**NoSQL**

**Interview Questions**

**► What is NoSQL?**

NoSQL stands for ‘Not Only SQL’. NoSQL encompasses different database technologies that were developed in response to a rise in the volume of data stored about users, objects, products, the frequency in which this data is accessed, and performance and processing needs.

NoSQL databases are designed with “**Big Data**” needs in mind. Since they are not bound by fixed schema model, this makes them suitable for a **large volume of non-uniform data** (Big Data).

**►► Compare / differences between NoSQL & RDBMS**

|  |  |  |
| --- | --- | --- |
| **Criteria** | **NoSQL** | **RDBMS** |
| **Data format / Schema / Flexibility** | No schema or dynamic schema, store **structured, semi-structured, unstructured** data with very few things to be predefined. Data could have different formats and changed on the fly. | Predefined data schema, organized and **structured**. Any schema changes could require hours of maintenance. |
| **Scalability** | Very good, scale out **horizontally** | Average, need to buy separate solution |
| **Querying** | No joins | Using SQL with different kinds of joins |
| **Performance / Indexes** | Great performance, limited indexing, **sharding**. | Multiple indexes to improve searches |
| **Transactions / ACID** | Can’t modify multiple documents in a single transaction, distributed data is usually “eventually consistent”,  **BASE transactions**. | Full support for transactions and ACID. |
| **Storage mechanism** | Depends on DB type: key-value pair, document, column storage, etc.  Large distributed volumes. | Data tables, indexes, relationships, SQL, stored procedures stored in database. |

**►► What is CAP theorem? How is it applicable to NoSQL systems?**

The CAP theorem was proposed by Eric Brewer in early 2000. In this, three system attributes have been discussed within the **distributed databases**:

* **Consistency** - all the nodes see the same data at the same time.
* **Availability** - a guarantee that there will be a response for every request made whether it was successful or not.
* **Partition tolerance** - quality of DB management system which states that the system will work even if a part of the system has failed or is not working.

A distributed database system might provide only 2 of the 3 above qualities.

**► Can you explain transaction support in NoSQL? What is BASE?**

The CAP theorem states that distributed systems cannot achieve all three properties at the same time; consistency, availability and partition tolerance. The **BASE** system gives up on **consistency** while maintaining the other two:

* Basically Available
* Soft state
* Eventual consistency

**►► What is sharding?**

Sharding is NoSQL feature of partitioning data into smaller databases to have faster access to data.

**►► What is eventual consistency?**

Eventual consistency in NoSQL means that when all the service logics have been executed, the system is left in a consistent state for achieving high availability. It’s possible that if you create an item, update that item, and then get that item, you may see the **old version**. The reason for this is that the item is persisted on multiple servers in the background. But it gives a guarantee that eventually all accesses to that item will return the last updated value. Most of NoSQL databases – MongoDB, Amazon Dynamo DB, Cassandra are eventually consistent in some of the configurations.

**►► What are kinds / types of NoSQL databases?**

There are several types of NoSQL databases, each with its own uses and applications.

|  |  |  |
| --- | --- | --- |
| **Key-value** | Uses (hash) keys to access different values – string, JSON, BLOB, etc | Amazon **Dynamo DB**, Redis, Oracle NoSQL, Cassandra |
| **Cache** | Similar to key-value, store frequently used data in memory. | Redis, Memcache |
| **Document** | Store hierarchical / **nested data** structures directly in the database. Many different key-value pairs. A collection contains sets of documents and each doc can contain diverse fields. | **MongoDB**, Couch DB |
| **Graph** | For relationships-intensive / network of data | Neo4J |
| **Columnar** | Data is stored in cells grouped as **columns** rather than in the form of **rows**. Sparse matrix, uses columns and rows as keys. | Google’s Big Table, HBase, Apache **Cassandra** |

**► What are the pros and cons of a Graph type of NoSQL DB?**

**Pros** of using graph database:

* These are tailor-made for the networking applications. A social network is a good example of this.
* They can also be perfect for an object-oriented programming system.

**Cons** of using graph database:

* Since the degree of interconnection between nodes is high, so it is not suitable for network partitioning.
* Don’t scale out well.

**►► What are some NoSQL databases (that you used, example)?**

* Amazon Dynamo DB
* MongoDB
* HBase
* Cassandra
* Couch DB
* Oracle NoSQL

**► How does NoSQL DB use / budget memory?**

The Replication Node manages the data in a NoSQL DB store and is the main consumer of memory. The Java heap and cache size used by the Replication Node can be important performance factors. By default, the Replication Node heap and cache are calculated by NoSQL DB based on the amount of memory available to the Storage Node. If the number of Replication Nodes on a Storage Node changes, the per-RN memory will be recalculated dynamically.

**► How to script NoSQL DB configuration?**

NoSQL database usually provides some CLI to script DB creation and configuration to be used for HA / clouds.

**► Could NoSQL database interact with RDBMS, e.g. Oracle?**

Usually NoSQL DB supports retrieving records from RDBMS using CLI and performing queries from RDBMS to retrieve records from NoSQL Database.

# Apache Cassandra

**► What is Cassandra?**

Cassandra is one of the most favored open source hybrid **columnar / key-value NoSQL** distributed DBMS efficiently designed to store and manage **large volumes of data**. Highly scalable for **Big Data,** originally designed by Facebook, written in Java, comprising flexible schemas, established on **peer-to-peer architecture** ensuring no failures.

Great flexibility as it allows insertion of multiple nodes to any Cassandra cluster in any datacenter. Any client can forward its request to any server. Strong data **replication** on nodes capability as it allows data storage at multiple locations enabling users to retrieve data from another location if one node fails. Supports **schema-free** / **schema-optional** data model.

<https://intellipaat.com/interview-question/cassandra-interview-questions/>

**► Explain Cassandra’s data model.**

The Cassandra data model has 4 main concepts which are:

* **Cluster** - contains many nodes (machines) and can contain multiple keyspaces.
* **Keyspace** – outermost container for an application, namespace to group multiple column families, typically one per application.
* **Column** - contains a name, value and timestamp.
* **Column family** - contains multiple columns referenced by a row keys.

**► Explain “tunable consistency”.**

Tunable consistency is a phenomenal characteristic that makes Cassandra a favored database choice. It refers to the up-to-date and synchronized data rows on all their replicas and allows users to select the consistency level best suited for their use cases. It supports two consistencies:

* Eventual Consistency
* Strong Consistency.

The Eventual guarantees consistency when no new updates are made on a given data item, all accesses return the last updated value eventually. Systems with eventual consistency are known to have achieved replica convergence.

For Strong consistency, Cassandra supports the following condition:

R + W > N, where

N – Number of replicas

W – Number of nodes that need to agree for a successful write

R – Number of nodes that need to agree for a successful read

**► How does Cassandra write?**

Cassandra performs the write function by applying **two commits** - first it writes to a commit log on disk and then commits to an in-memory structured known as memtable. Once the two commits are successful, the write is achieved. Writes are written in the table structure as SSTable (sorted string table). Cassandra offers faster write performance.

**► What is CQL?**

CQL is Cassandra Query language to access and query the Apache distributed database. It consists of a CQL parser that incites all the implementation details to the server. The syntax of CQL is similar to SQL but it does not alter the Cassandra data model.

**► Explain Cqlsh**

Cqlsh expands to CQL Shell that configures the CQL interactive terminal. It is a Python-based CLI used on Linux or Windows and exequte CQL commands like ASSUME, CAPTURE, CONSITENCY, COPY, DESCRIBE and many others. With cqlsh, users can define a schema, insert data and execute a query.

**► Does Cassandra support ACID transactions?**

No, Cassandra does not support ACID transactions.

**► Define composite type?**

In Cassandra, composite type allows to define a key or a column name with a concatenation of data of different type. You can use two types of composites:

* Row key
* Column name

**► What are the collection types provided by CQL?**

There are three collection data types:

* **List** - one or more ordered elements.
* **Map** - key-value pairs.
* **Set** - one or more unique elements.