NoSQL Databases Overview, Types and Selection Criteria - XenonStack

NoSQL Database Explained

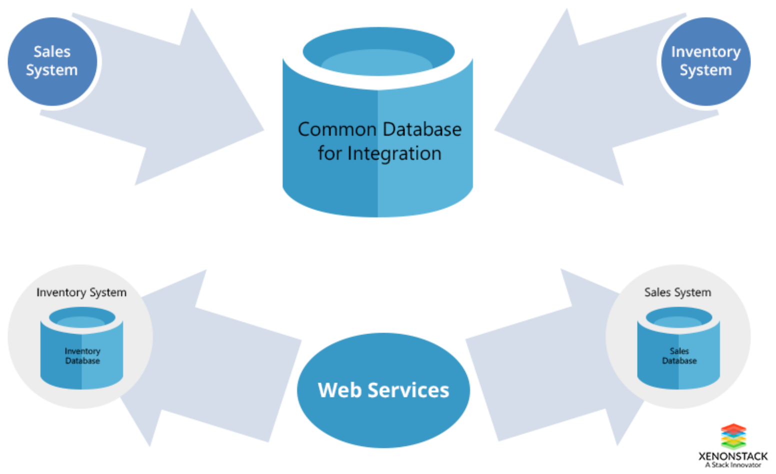
As the name implies NoSQL, also called Not-only-SQL are the databases that let the developers store/manage unstructured data and perform complex analytical operations on it as well.

Nowadays, a wide range of NoSQL databases are available and developers can choose according to their requirement. So, the companies and developers now do not need to stay confined to a single kind of database platform.

- NoSQL database was first adopted by companies such as Amazon DynamoDB, Google and others for solutions to real problems. These companies realized that SQL didn’t meet their requirement and decided that they needed NoSQL databases to this problem.
- Then they tried their traditional approach, they upgraded to faster hardware. When even that did not work, they tried to scale existing relational solutions by de-normalizing the schema. NoSQL databases store the data in the denormalized form and follow the different model to store the data depending upon requirements, which explained further in this blog.

Key Characteristics of NoSQL Database

Due to a mismatch between the in-memory data structure and relational data structure of applications, many problems were faced by application developers. By using NoSQL databases, developers do not need to convert in-memory structure to relational structure. Hence, they also use it as an integration point to the application.



- Relational databases were not designed in such a way that they can run perfectly on clusters.
- The storage needs of an ERP application are very different than the data storage needs of Facebook and other such applications.

The organizations are shifting to NoSQL databases to achieve higher scalability, higher speed, and continuous availability.

Features of NoSQL Database

- **The need for Speed –** Whenever a fast response time is required, the data should be placed in the memory. In this case, when the very fast response time is required we have to choose a database that stores the data in the memory.
- **The need of Scale –** With the increased number of users and data volumes organizations requires such databases which are easily scalable.
- **Need for Continuous Availability –** Slow performance can drive a customer away and nothing is worse than downtime. There is a difference between high scalability approach that RDBMS offer with master-slave architecture and the continuous availability that NoSQL databases like Cassandra offer no downtime with redundant copies of data are being spread throughout a cluster across multiple locations.
- **Need for Location Independence –** The ability to serve data quickly to multiple locations is critical. Because of fundamental master-slave design, RDBMS struggles to provide fast read access to many locations.

NoSQL databases can easily spread across multiple data centers and cloud availability.

For example, Adobe runs on Datastax enterprise using Apache Cassandra Database cluster between two data centers to ensure its customers can read and write data fast, no matter where they are located.

NoSQL database like Cassandra offers a much more flexible data model that can easily store structured, semi-structured and unstructured data.

Moving From Relational Database to NoSQL Database

- **New Applications**

Many applications which made in SQL begin with NoSQL by creating a new application and starting from the ground up, but it creates the issue of application rewrite.

- **Augmentation (a process of making greater or larger in size)**

Some choose to augment an existing by adding a NoSQL component to it. This often happens with applications than having outgrown RDBMS due to scaling issues, the need for better availability or other issues. Part of the application continues to use existing RDBMS, but the other components of an application are modified to utilize the NoSQL database.

- **Full Rip-Replace**

The system that simply is proving too costly from an RDBMS perspective to keep or increase of users concurrency. A full replacement is done with NoSQL databases.

Requirements To Move From RDBMS To NoSQL Database

- RDBMS systems are made such that they don’t scale.
- Handle things like foreign keys, maintain relations over the entire data set. The problem with this is to handle the data on a large set of machines with their foreign key relationships.
- According to CAP only two properties out of three can be achieved. If the consistency is the absolute requirement we have to give up the other two. Because the RDBMS follow ACID(Atomicity, Consistency, Isolation, Durability), so it is difficult to scale the RDBMS. Almost all data stores handle things like-
  - Concurrency
  - Queries
  - Transactions
  - Schema
  - Replication
  - Scaling

Performance and scalability – Two are odd to each other, increasing one would decrease the other. For performance, how we execute the same set of requests, over the same set of data with –

- Shorter time
- Few resources usage
- There is also a tradeoff between resource usage and processing time. In general, we can say that we can reduce the processing time by consuming more resources. Conversely, we can reduce

the processing time by consuming more resources.

Types of NoSQL Databases

There are different types of data stores under NoSQL databases available which allow storage of data. These have different ways to store data. Some of the data stores that come under NoSQL databases are explained below :

Key-Value Database

- The Key-value store or key-value based database is a database that uses an associative array(such as a map) where each key is associated with one and only one value in a collection. This kind of relationship is referred to as a key-value pair.
- In a Key-value pair, each key value is represented as an arbitrary string such as a hash value.
- The value is stored as a blob.
- The storage of value as BLOB removes the need to index the data to improve performance so that we cannot control what's returned from a request by value.
- Key-value stores do not have any query language. They only allow to store, retrieve and update data using simple get, put and delete commands and the data can be retrieved by making a direct request to the object in memory or on disk.
- Some examples of key-value store Databases are –
  - Aerospike
  - Apache Cassandra
  - Berkeley DB
  - Couchbase Server
  - Redis
  - Riak
  - Memcached

There is a difference between the databases which come under key-value databases; all databases are not the same.

For example, Memcached data is not persistent while Riak is. Using Memcached to implement the caching of user preferences will load all the data when the node goes down and refresh required from the source system.

For example, If we use Riak we may not need to worry about losing data but we need to focus only on how to update data. It is important to not only choose a key-value database based on your requirements but also to choose which key-value database to be used.

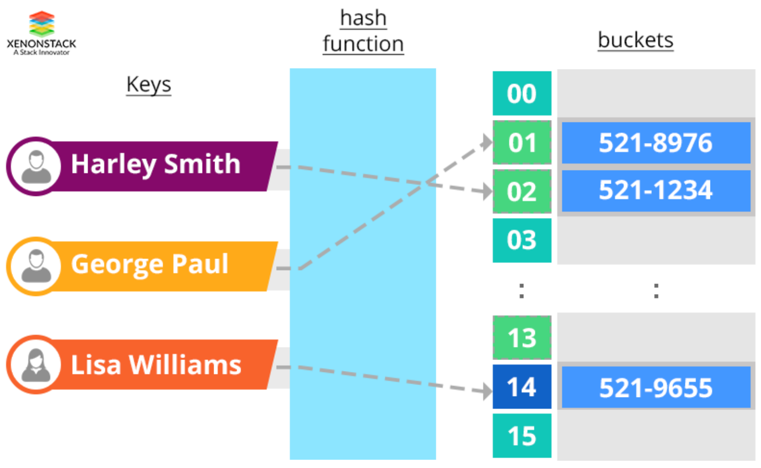
- **Queries –** Queries are performed only on the basis of the key.
- **Schema –** It stores the data on the basis of the key-value pair. It stores the data corresponding to the key in the format of a BLOB.
- **Scaling up –** Keyspace is shared means that key starting with A data for this key will go to one server. Key starting with B data for this key go to another server. But it exposes the system to data loss if a server goes down.
- **Replication –** When we write data to multiple machines. If there are two servers in the cluster then the value of key "ABC" are two different things for two different servers. Resolving this is a complex issue and during updates it creates problems.

Uses of Key-Value Store

Key/value stores are used when we have to access the following data –

- session
- shopping cart info

For example – In a shopping mall, information regarding a particular product is stored on the basis of a particular key. So, when the product is scanned, on the basis of barcode all the information for a particular product is accessed.



This key/value database allows us to read and write values as follows –

- Get (key) returns the value associated with the provided key.
- Put (key, value) associates the value with the key.
- Multi-get (key1, key2, .., keyN) returns the list of values associated with the list of keys.
- Delete(key), removes the entry for the key from the data store.

Document Database

- Document-based databases are similar to key/values databases. They store data on the basis of the key/value which is similar to a key-value database. But the only difference is that it stores the values in the form of XML, JSON(javascript object Notation), BSON (Binary encoding of JSON objects).
- The database understands the format of data so that the operations can be performed easily.
- It allows the storage of complex data. If we want to store trees, collections, and dictionaries, then it is a good choice.
- It does not support relations. Each document is standalone. It can refer to other documents by storing their key, corresponding to the particular document.
- Document-based databases do not support the joins, so it almost overcomes the problem of sharing the data across multiple nodes.
- Some of the document-based databases are –
  - MongoDB
  - CouchDB
  - Terrastore
  - OrientDB
  - RavenDB

Queries – There is no other way to query the data except the key-value stores. We can also perform range queries on the basis of a key.

Transactions – Mostly document-based database support transaction for a single document.

Schemaless – Schemaless means it does not require any schema to store the data. Each document can differ in the number of columns. It understands the data of JSON format only.

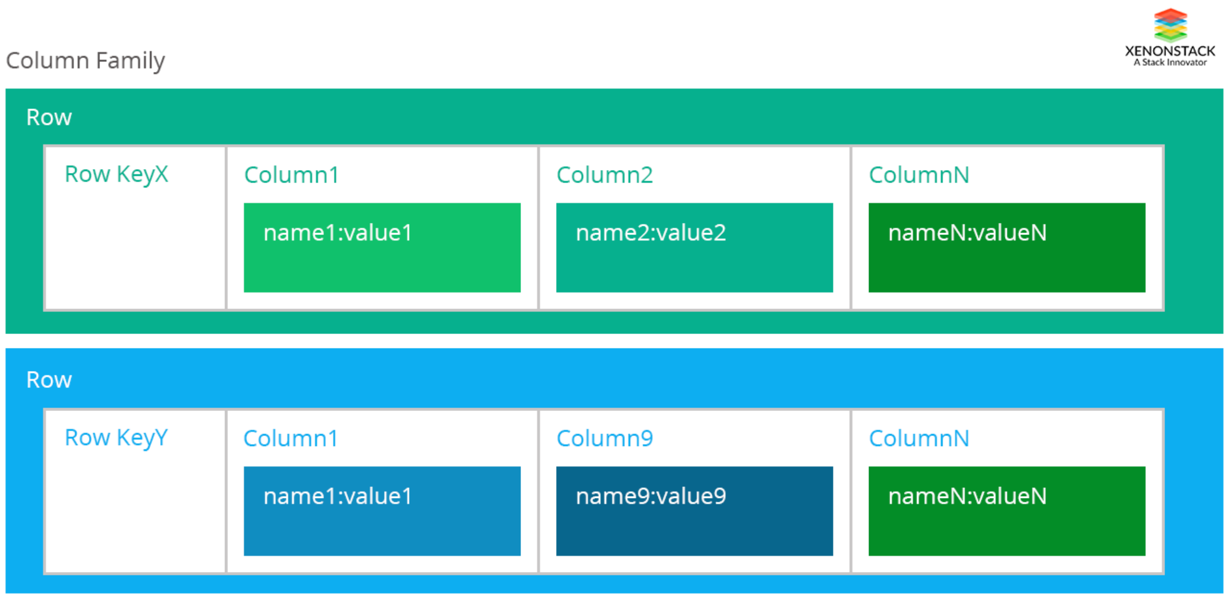
Scaling up – In this database, each document is an independent document. It does not support joins. So it is easily possible to share the data across multiple nodes independent of each other.

Column-family Database

Column-family databases store data in column families as rows. These rows have many columns associated with a particular row. Column families basically contain the group of correlated data which we can access together.

- Each column family can be compared to a container of rows in an RDBMS table where the key identifies the row and the row consists of multiple columns.
- Rows do not need to have the same columns, and columns can be added to any row at any time without having to add it to other rows.
- When a column consists of a map of columns, we have a super column. A super column consists of a name and a value which is a map of columns. Think of a super column as a container of columns.
- Some Column-family databases are –
  - Cassandra
  - Hbase
  - Hypertable
  - Amazon DynamoDB

Cassandra is more fast and scalable as compared to other column-family databases with write operations because data is spread across the cluster.



4 Major Benefits of Column Family Database

- **Compression –** Column-based data storage stores data efficiently through data compression and by using data partitioning.
- **Aggregation Queries –** Due to the structure of the column family data structure, they perform particularly well with aggregation queries (such as SUM, COUNT, AVG etc).
- **Scalability –** Column databases are more scalable as compared to other databases. They are well suited for a data structure where data is spread on a large cluster of machines, often thousands of machines.
- **Fast to load and query –** Columnar stores can be loaded fast. A table containing millions of rows can be loaded within a few seconds. We can start querying and analyzing immediately on the loaded data.

Understanding Graph Database

Graph databases allow you to store data in the form of nodes and edges in which nodes are represented as entities and relationships are represented in the form of edges. Node is an instance of an object in an application. Relations which are known as edges can also have their properties. Edges have directional significance to represent the relationship between the edges.

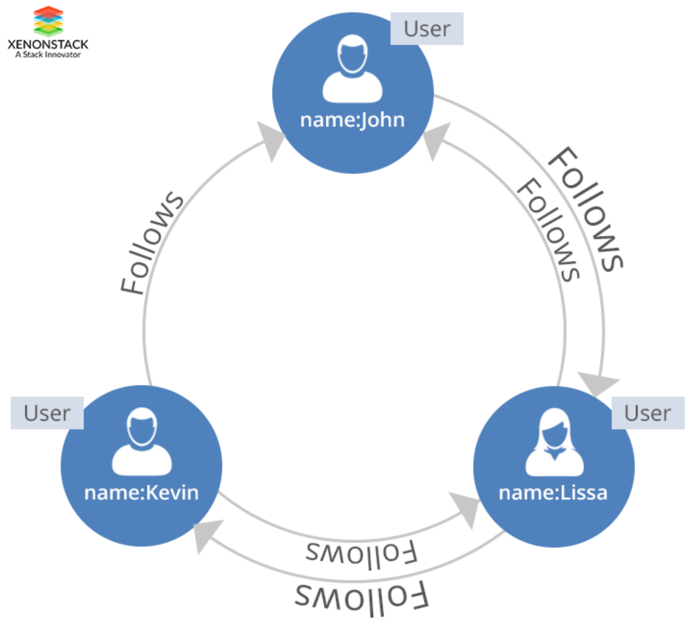
- The graph database allows you to store the data only once and a number of different types of relationships can be stored in these nodes.
- Relationships which are represented in the form of edges can be unidirectional, bi-directional which is same as the one to one, one to many, many to one and many to many relationship types of Relational Database Management System.
- In RDBMS, adding another relation after the schema creation results in a lot of schema changes and data movement. But this problem of RDBMS is overcome by graph databases. It requires only storing data once in the form of nodes, then after a number of different types of relationships in form of edges can be specified to the already stored data (data which is stored in the form of nodes).
- In graph databases, relationships between the nodes are not calculated at query time because it is persisted as a relationship. Traversing persisted relationships are faster as compared to calculating the relationship at query time.
- Relationships are an important part of the graph database. By adding properties to the edges (relationships), we can add some level of intelligence to the graph database.
- Adding new relationships to the graph databases is easy. But changing existing relationships to the graph databases is a difficult task because changes have to be made on each node and for each relationship in the existing data. So changing the existing node and their relationships is similar to data migration.
- Since most of the queries to the graph database are answered on the basis of relationships and its properties, it is mandatory to choose the relationship properly.
- There are different types of graph databases. Some graph databases, support only single-depth relationships. With some graph databases, we can not traverse more than one level of relationship.

Properties are added to the edges of the graph database which helps us to query the graph database. For example: what is the distance between two cities, the current age of a particular person. Then the above queries can be answered with the properties of edges. The distance between two cities can be found out with the help of start node, an end node, the distance property which links these two vertices help to find the distance between two nodes.

The current age of a person can be found with the help of the birth date and the current date. So, the age of a person can be answered with the help of age property which links two vertices, which contain the birth date and current date of the person.

Some of the Graph databases are mentioned here –

- Neo4J
- Infinite Graph
- OrientDB
- FlockDB



Choosing Right NoSQL Database

- **The Data Model**

It involves the type of data that you need to store. NoSQL Databases only differ in the data model that they use. A mismatch between the NoSQL databases solution data model and target application can make or break the success of the project which you are building.

- **Data-Scaling expectations**

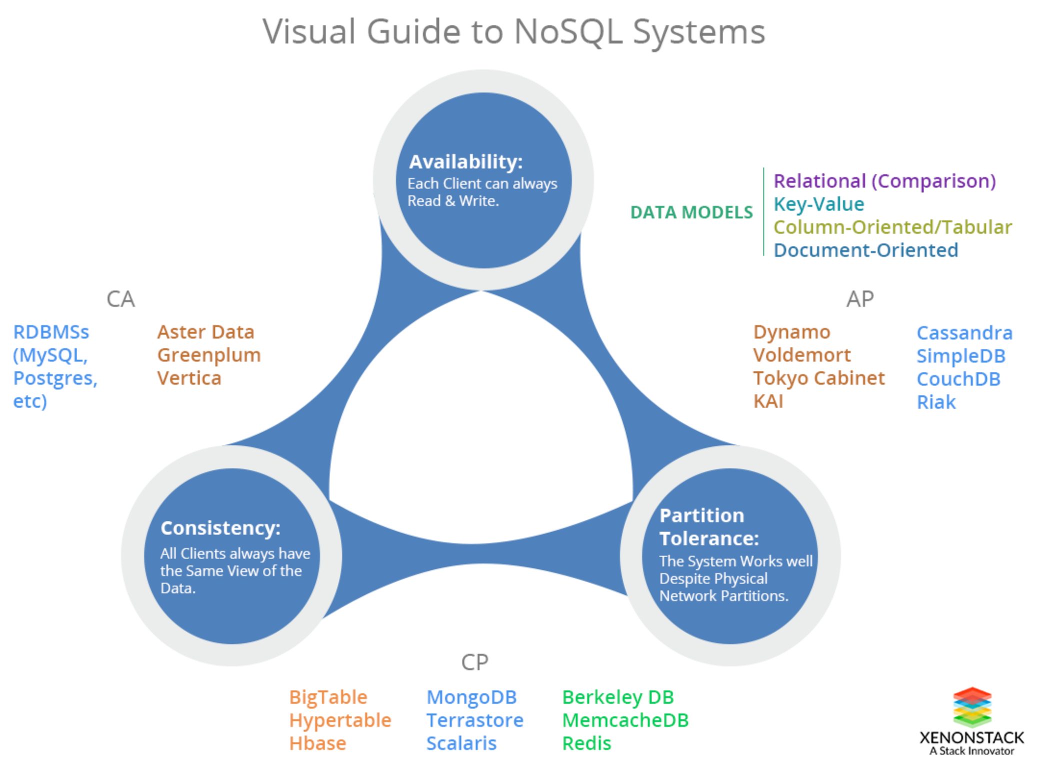
The next question is how large an application is expected to grow and how much data scale support will be needed. Some NoSQL databases are memory-based and do not scale across multiple machines where others like Cassandra, scale linearly across many applications.

Choosing a Data Model

- To choose a data model, it is first required that we check what type of data is required to be stored. Depending on that, we choose the data model.
- If the data is to be represented in the form of a graph, then the graph database is used.
- If we have to store the data in the form of the key-value pair then we can choose a key-value database or document database is chosen.
- Different data models are used to solve different problems. For example: if we are solving the graph problem with the relational database, so it is better to resolve the graph problem with the graph database.

What is the CAP Theorem?

The concept of consistency(C), availability and partition tolerance(P) across distributed systems gives rise to the need for CAP theorem. But CAP theorem demonstrates that any distributed system cannot guarantee C, A, and P simultaneously.



- **Consistency in CAP Theorem**

When data is stored on multiple nodes, all the nodes should see the same data, meaning, that when the data is updated at one node then the same update should be made at the other nodes storing the same data also.

For example, if we perform a read operation, it will return the value of the most recent write operation causing all nodes to return the same data.

A system is said to be in a consistent state, if the transaction starts with the system in a consistent state, and ends with a system in a consistent state. In this model, a system can shift into an inconsistent state during a transaction but, in this case, the entire transaction gets rolled back if there is an error at any stage in the process.

- **Availability in CAP Theorem**

To achieve a higher order of availability, it is required that the system should remain operational 100% all the time. So we can get a response at any time. So according to this whenever a user makes a request, a user should be able to get the response regardless of the state of a system.

- **Partition Tolerance in CAP Theorem**

According to this, a system should work despite message loss or partition failure. A system that is partition-tolerant can sustain any amount of network failure. A system that is partition tolerant can sustain any amount of network failure that does not result in a failure of the entire network.

A storage system that falls under CP (partition tolerance with consistency) are MongoDB, Redis, AppFabric caching and Memcached DB.

Databases that come under the partition tolerance are those which store their data on multiple nodes.

As in relational data models, it is required that it should follow the ACID (Atomicity, Consistency, Isolation, Durability) properties. But with NoSQL databases, it is not possible for data storage structures to follow all the C, A and P.

Data storage models which come under the NoSQL databases of the following but it is not possible to follow all –

- CA(Consistency and Availability)
- AP(Availability with partition Tolerance)
- CP(consistency with partition Tolerance)

Consistent, Available (CA) Systems have trouble with partitions. Examples of CA systems include –

Traditional RDBMSs like Postgres, MySQL etc (relational)

- Vertica (column-oriented)
- Aster Data (relational)
- Greenplum (relational)

Consistent, Partition-Tolerant (CP) Systems have trouble with availability while keeping data consistent across partitioned nodes. Examples of CP systems include –

- BigTable (column-oriented/tabular)
- Hypertable (column-oriented/tabular)
- HBase (column-oriented/tabular)
- MongoDB (document-oriented)
- Terrastore (document-oriented)
- Redis (key-value)
- Scalaris (key-value)
- MemcacheDB (key-value)
- Berkeley DB (key-value)

Available, Partition-Tolerant (AP) Systems achieve “eventual consistency” through replication and verification. Examples of AP systems include –

- Dynamo (key-value)
- Voldemort (key-value)
- Tokyo Cabinet (key-value)
- KAI (key-value)
- Cassandra (column-oriented/tabular)
- CouchDB (document-oriented)
- SimpleDB (document-oriented)
- Riak (document-oriented)

Column Family vs Column Store Database

Bigtable, HBase, Hypertable, and Cassandra come under column-store databases due to their ability to store and access column families separately. This makes them appear in the column store like :

Sybase IQ, C-Store, Vertica, Vectorwise, MonetDB, ParAccel and Infobright which are also able to access columns separately.

In NoSQL, column store databases are categorized into two groups –

- **Group A:** Bigtable, HBase, Hypertable, and Cassandra. These four systems are not intended to be a complete list of systems in Group A.
- **Group B**: Sybase IQ, C-Store, Vertica, VectorWise, MonetDB, ParAcce, and Infobright.

Difference between column-based store databases which are divided into two parts Group A and Group B on the basis of following parameters –

- **Data Model**

Group A uses a multi-dimensional map. It can be row-name,column-name and timestamp are sufficient to uniquely map value in the database. Group A does not use a relational database model.

Group B databases do not use a relational data model. This results in many people saying that column store databases are not relational.

- **Independence of columns**

Databases which come under Group A stores parts of a data entity/row in separate column-families. They have the ability to access these column families separately. Because of column family databases, consisting many columns and the columns within column-families are not independently accessible.

Group B: Databases come under this group, stores columns separately, so the columns are accessible independently.

Group A, it is useful for queries that only access a subset of table attributes in any particular query. The only difference is that each column is stored separately instead of column families. The interface used by these two column family stores: Group A is a part of NoSQL, not have a SQL interface. Group B supports the standard SQL interfaces.

- **Optimized workload**

Group B has optimized for reading based analytical workloads. These systems have a fast load time, but poor performance for making updates.

Because the data warehouses require complex queries, they require mainly read operations and rarely updations are performed in data warehouses.

Group A databases come under group A can handle the much higher rate of updates.

Group B stores the data in the database for the above image in the following form –

(ID): 1 2 3 4 5 6

(First Name): Joe, Jack, Jill, James, Jasmine, Justin

(Last Name):Smith,Williams,Davis,Miller,Wilson,Taylor

(Phone):555-1234,555-5668,555,5432,NULL,555-6527,55-8247

(E-mail): jsmilth@gmail.com, jwilliams@gmail.com, NULL, jmiller@gmail.com, NULL, jtaylor@gmail.com

Each value is stored by itself, without information about what row or column it came from. Therefore, we can figure out from which row it is coming by counting the number of rows above it corresponding to the same column.

For example, if we want to check the last name for an id whose value is 4, we have to check the value in the fourth row in the last name column to find out the name. So it becomes compulsory to fill the fields with null values to maintain the order and to get the correct value.

So the group B takes much less space on storage than Group A.

So by storing just column values without storing column-names or row-names, databases under group B optimizes performance by reading the data for each column and after that applying aggregations on them.

NoSQL Challenges

- **Data Model Differences**

Companies struggle with the mental switch from the relational to NoSQL data model. Projects can be made or broken depending on whether the team has correctly modeled the data for the NoSQL databases to maximize its capabilities so it is required that database professionals should be trained and acquired with the new NoSQL data model in the database they choose.

- **The Distribution Model**

Some NoSQL databases use the master-slave architecture which can only somewhat scale read operations as compared to peer-to-peer architecture that can scale both read and write.

- **No Advanced Expertise**

NoSQL databases are still new, so most of the developers are learning how to use it, but over time this situation will resolve. But it is easier to find an RDBMS expert than a NoSQL developer.

- **Lack of Security**

In 2012 in an article, it was said that “NoSQL” equals no security. In this article, the author cited the lack of security feature in NoSQL databases.

Conclusion NoSQL Database

NoSQL database provides us many benefits, consistency, availability and partition tolerance. It also provides facilities to easily store the graph data, which is not available with SQL databases. Instead of that, there are some problems with NoSQL databases.

One of the drawbacks of NoSQL databases follows CAP, according to this database which follows CAP can obtain only two out of three and have to skip the third one. Because with the CAP only two properties can be achieved. But it provides us with the facility to store the data in the form of denormalized form.

- Explore the Difference between “**SQL vs NoSQL vs NewSQL**“
- Learn more about “**Database Testing**” in this insight
- Get in Touch with us for “**Database Migration Services**“