Deciding between SQL and No SQL databases can be a difficult task for which it is important to understand the key differences between their transaction models. In this article, we are going to discuss the two most popular database transaction models: ACID and BASE.

ACID(Atomicity, Consistency, Isolation, Durability)

ACID properties are used for maintaining data integrity during transaction processing. The term ACID stands for Atomicity, Consistency, Isolation and Durability. In order to maintain consistency before and after a transaction relational databases follow ACID properties. Let us understand the different terms in the word:

Atomicity:

The **all or none principle**, either the transaction takes place or it doesn’t. If one element of the transaction fails the entire transaction fails, they don’t occur partially.

Consistency:

The transactions should follow all rules defined in the system. The database must be in a consistent state before and after a transaction.

Isolation:

Each transaction is independent of the other. A transaction does not have access to other transactions which had not finished yet. The changes occurring during a transaction will not be visible to others until it is written in the main memory.

Durability:

Once the transaction has completed and the writes and updates have been written to the disk, it will remain in the system even if a system failure occurs.

BASE(Basically Available, Soft State, Eventual Consistency)

With the rise of No SQL databases, the BASE model was designed. No SQL databases follow the CAP theorem.

Basically Available:

Basically available means that there will be a response to every request even if the response is a system failure or does not successfully return the data.

Soft State:

The state of the system is always soft which means that it can change over time even if there are no reads or writes as the system keeps changing the data to make it consistent.

Eventual Consistency:

The data might not be consistent immediately but eventually, it becomes consistent. Reads in the system are still possible even though they may not give the correct response due to inconsistency.

ACID vs BASE: Which one should you choose?

Both the models have been designed to satisfy different requirements. While choosing a database you need to decide keeping in mind the properties of both the models and the requirements of your application. **SQL** databases like MySQL, SQLite, Microsoft SQL, Oracle follow ACID properties whereas **No SQL databases**MongoDB, Cassandra, Redis follow BASE.

No SQL databases like MongoDB are becoming increasingly popular these days and we can see an increasing number of companies using them. However there still some scenarios for which you need SQL databases. we will cover the important differences between No SQL and SQL databases and help you in deciding which one to choose for your next application.

# ****Difference between SQL and No SQL Databases****

In No SQL databases like MongoDB data is usually stored in JSON format whereas in SQL databases it is stored in the form of tables and relations. A simple example of this is to consider the schema of a user. In SQL databases user’s information like his name, age, gender and more will be stored in one table and his address will be stored in a separate table whereas in a No SQL database all this information of the person can be stored together in JSON format.

# Why should you go with No SQL?

# 1. **Insertions and retrievals are easier:** Consider the example of the user discussed above. In SQL you might need to insert the user’s data into multiple tables like the user’s address in the address table and other information in the user table. In No SQL information can be stored in the form of a block at one place making insertion easier. Similarly, if you trying to retrieve the person’s data you might need joins in SQL which are costly however in No SQL the entire block of the user’s data is stored at one place making retrieval faster and easier.

2. **Schema is flexible**: Maintaining consistency is difficult in SQL. When a new attribute like age is added to the user’s schema it will be added for all the previously registered and the new users. In No SQL you can simply choose to add the attribute for only the new users.

3. **Built for Scale**: In SQL databases we might need to use techniques like sharding for faster retrieval. Sharding is partitioning the data into smaller chunks based on some attributes for faster retrieval. In No SQL databases like MongoDB horizontal partitioning is inbuilt hence they are built for scale.

**4. Built for Analysis**: It is easier to find metrics in No SQL databases like finding the average age of all the registered users.

# Why should you go with SQL?

# 1. **Faster Updates**: If your data requires many updates then No SQL might not be the right choice for you. In No SQL for every row, you need to scan the entire block before coming to the attribute you need to update whereas in SQL you can directly access that column in the table. Thus No SQL database is not made for updates.

**2. Consistency**: In No SQL databases, you can have several users with the same key value whereas in SQL we have the concept of primary key which helps us to uniquely identify each entry. So Consistency is a problem in No SQL databases. SQL databases have ACID ( Atomicity, Consistency, Isolation, Durability) properties which are not guaranteed in No SQL. Thus No SQL databases are not used by financial systems because it is difficult to uniquely identify different transactions.

**3. Faster Reads**: No SQL databases are not read-optimized. For example, if you need to find the age of a user, No SQL database will read the entire block till it finds the age attribute. However, in SQL databases reader can directly get the age column from the table. Hence read times are comparatively slower in No SQL databases.

**4. Relationships are implicit**: In relational databases like MySQL, we know how different tables are related to each other and we can put referential integrity constraints to ensure that the corresponding value is present in the other table. In No SQL databases, we don’t have the concept of foreign key hence relationships are not implicit.

**5. Joins are easier in SQL**

Joins are more costly in No SQL as each time we need to find the attributes we are performing the operation on and then perform the cross product. However, in SQL we can easily access the columns making joins faster.