Consistency Levels in Database Management Systems - Aditi Shashank Joshi (1222838916)

In distributed database systems, some tradeoff has always been made to calculate and predict the correctness for the performance. This category is called consistency levels. There are **many consistency levels** which a user can choose from depending on the application and its extensibility. Similar to database isolation levels, weaker consistency maximizes the performance of the systems. Consistency levels are discussed in this article in reference to ACID database systems that act on a single data item at a time.

What is a consistency level? When a system follows an established set of rules, it is said to be consistent. As consistency levels are directly related to Context, the set of rules changes from system to system. These rules are **interpreted differently** in ACID (Atomicity, Consistency, Isolation, and Durability) and CAP (Consistency, Availability and Partition Tolerance) theorem. In **ACID** - C refers to consistency level which depends on application. It is ensured by a system that does not violate referential integrity requirements, foreign key constraints, or any other application specific constraints. In ACID, only an application developer can ensure if the code inside the transaction **does not violate application semantics** when the transaction is run in isolation. In **CAP** - C refers to the rules for making a distributed, concurrent system appear to be a single threaded, centralized system. The reads in the system reflect the most recently completed write, regardless of the server which is processed. In CAP, the consistency implies that the system **ensures that all the writes are reflected by the reads**, regardless of the location and time of these writes. The types of consistency which are covered in this blog are:

1. **Sequential Consistency:** All the writes are **globally ordered**. All single threads must see the **writes that are occurring in the same order.** Sequential Consistency is executing the instructions of processors in some sequential order. If it happens the other way around, consistency is violated. The sequential consistency doesn't concern the order of writes. It allows the official history to be different, provided every thread agrees to see the changes in their write order.
2. **Strict Consistency:** The events must be ordered in the same real-time sequence in which they occurred, that is they **agree to global ordering**. An **earlier write/read always has to be seen before a later write/read to the other systems**. Every read action in real time must read the value of the most recent write. In a distributed system, it is difficult to track the

precise current time due to which strict consistency is hard to be applied in such cases. This is also known as the highest level of consistency.

3. **Linearizability / Atomic Consistency:** It is similar to the strict consistency as it imposes real time constraint on writes, and an extension of sequential consistency. All threads agree to global ordering. Although this is similar to Strict Consistency, it takes into consideration **that there's some time gap between when an operation is submitted to the system and when the system acknowledges it**. This is the highest level of consistency obtained in practice.
4. **Causal consistency:** This is a most popular and useful consistency level. It does not concern the orderings of unrelated writes. It only enforces the order constraints on related writes. Hence, all threads in context do not have to agree to global ordering. When a thread performs **a read operation on a data item and then writes it, it assumes that the read triggered the following write.**
5. **Eventual consistency:** In this consistency, if there are **no writes done for a long period of time then the threads in due time agree upon the value of the last write.** Hence, the system will have the values of last write.

What is strong consistency and weak consistency?

Strong consistency levels are the ones where the state of the database goes through a system level agreed sequence of state changes. This **hides the reality of duplication** from the end user, that is the user can see only one copy of the database which keeps on changing in a forward direction. Strong consistency includes Strict consistency, linearizability and sometimes Sequential Consistency. **Weaker consistency levels** are the ones which allow different views of databases to allow the end user to see the development during each stage of read/write. The user must be aware that the **data has been divided into views**, this increases the complexity of the system. Weak consistency includes Causal Consistency and Eventual Consistency.

In the context of distributed database systems, consistency can be applied to individual data items as well as entire transactions. The **reads and writes requests are grouped together** in a transaction with a transaction identifier that started each request. Within the transaction, all reads and writes have the same outcome: success or failure. They're separated from other transactions that are ongoing at the same time. The atomicity and durability guarantee of transactions deal with different concepts than consistency guarantees due to the isolation which places limitations on when writes become visible.