Gurjus Singh

MSDS 432 Foundations of Data Engineering

August 23rd, 2020

Module 9– Reading Comprehension

1. **What does ACID stand for, and what does each part mean? (2 pt)**

   ACID is the safety guarantees provided by transaction and stands for Atomicity, Consistency, Isolation, and Durability. The term was coined in 1983 by Theo Harder and Andreas Reuter. Their goal was to provide a way of describing fault-tolerance in databases. When analyzing database systems, it is good to make sure they are ACID-compliant. To check this, it is good to know the definition of each term.

   Atomicity base word is atom. When thinking about the base word atom the meaning is something that cannot be broken down further. In a database context it describes what happens if a client wants to make several writes, but a fault occurs after some of the writes have processed. If the writes have occurred in an atomic transaction this means if a fault occurs, then the whole transaction has to be discarded because it cannot be broken down. Atomicity prevents duplicating changes, and incorrect data.

   Consistency in ACID means that you have certain statements about the data that must always be true. The concept of consistency depends on the application; and the application should define its transactions correctly. Isolation is another term in ACID that means that concurrently executing transactions must be isolated from each other, so it prevents the race condition. Isolation prevents transactions from stepping on each other's toes. Isolation is known as serializability in some textbooks which is when the transaction should be thought of as the only transaction running on the machine and then serializing occurs where another transaction runs after this transaction has been completed. Durability is when data can be kept in a safe spot without the fear of losing it. It means that any data that has been written is not forgotten even when there is a hardware failure or software crash.

2. **What are dirty reads / dirty writes and ways to avoid them? (2 pt)**

   In a database dirty reads means you see transactions that have not been committed to a database while in dirty writes you overwrite data that has not been committed. Ways to fix dirty reads is by using locks and then require any transaction that wants to read an object to briefly acquire the lock then release it after reading. This guarantees that a read doesn't happen when an object has a dirty, uncommitted value because the lock is held by the transaction that has made the write. Although this does work, it does not work efficiently and a better way of doing it is by allowing the database to keep the old committed value and new committed value and while the uncommitted value has not been committed transactions can only read the old committed value. A way to fix dirty writes is by using row-level locks.

When a transaction wants to modify an object, it must first obtain the lock associated with the object. It will hold the lock until the transaction has been committed or aborted. One transaction can hold the lock for a given object; once it has been committed then the other transaction can have the lock.

3. **What is read skew / write skew and ways to avoid them? (2 pt)**

   Read skew is when a client views different parts of the database at different points in time. This can be prevented with snapshot isolation, which allows a transaction to read at one point in time. Write skew is when a transaction reads something, and makes a decision based on the value it saw and writes the decision, but by the time the transaction writes the decision, the value is no longer the same which makes the decision no longer true. Serializable isolation can be a way to avoid this anomaly.

   With read skew one way to fix this problem is to use snapshot isolation and is the most common solution to this problem. The idea behind it is the transaction reads from a consistent snapshot that is that the transaction sees all the data committed during the beginning of the transaction. During the transaction, the transaction only sees the old data from the snapshot. For write skew, serializable isolation works. This means that transactions write one at a time or serially.

4. **What is serializable isolation and what are three techniques used for its implementation? (2 pt)**

Serializable Isolation is when a transaction is running one at a time, or serially. Three techniques that are used for this implementation are serial order, two-phase locking and optimistic concurrency control technique such as serializable snapshot isolation. Serial order is when you use actual serial execution. What this means is you remove concurrency entirely, and this will help avoid problems of preventing conflicts between transactions. Another technique used is two phased locking which is when readers and writers are blocked by having a lock on each object. The lock is in shared mode or exclusive mode.

Deadlocks can occur in this technique which is where a transaction waits on another transaction. The third technique is called serializable snapshot isolation which is when all reads are made from a consistent snapshot of the database. It then uses an algorithm for detecting serialization conflicts among transaction writes and determines which transactions to abort. This a type of optimistic concurrency control because it is optimistic is it allows the transaction to try in the hope that everything is okay, and then makes sure something bad didn't happen and if it did then the transaction is aborted.

5. **What is Two-Phase Locking and how is it used? (2 pt)**

   Two-Phase locking makes lock requirements stronger. Transactions are allowed to concurrently read the same object as long as no transactions are currently writing to the object. If a transaction wants to write, then exclusive access is required. If Transaction A is reading and Transaction B wants to read it has to weight till A aborts or commits. Another

thing that can happen is the vice versa where A wants to write, and B wants to read then B has to wait. Two phased locking also provides serializability, it protects against race conditions, including lost updates and write skews.

The blocking of readers and writers works by having a lock on each object. The lock can be in shared mode or exclusive mode. If the transaction acquires a shared mode lock that means other transactions can also hold the lock in shared mode but once a transaction has an exclusive lock; those other transactions must wait. If a transaction wants to write, it can only write with an exclusive lock. If a transaction first reads and then writes, then the transaction's lock may be upgraded from shared lock to exclusive lock. If a transaction has a lock it will continue to hold the lock until it has been committed or aborted. Deadlock can also occur with so many locks being used. The database can automatically detect these deadlocks and can abort them so the others can continue.