Concurrency Architecture

<https://dl.acm.org/doi/10.1145/356842.356846>

<https://dspace.mit.edu/bitstream/handle/1721.1/16279/05331643-MIT.pdf>

TODOS:

* insert diagrams,
* Format citations,

The main concurrency architecture for postgres is called Multiversion Concurrency Control, herein referred to as MVCC. This architecture was first described in *Naming and synchronization in a decentralized computer system*. MVCC solved many of the issues of concurrent access to a database. In addition to this postgres, also permits finer grained concurrency control mechanisms on top of MVCC, in order to get the desired results. The MVCC architecture is more of a procedural architecture, than a structural one, and focuses more on a high level how concurrency issues are generally handled.

MVCC, Multiversion Concurrency Control

In general for concurrent systems that read and write a shared element, there needs to be some mechanism in order to avoid race conditions. A simple solution is to use a locking mechanism, to control, when process can read or write to an element. This locking mechanism functionally blocks access to some element under some conditions. A popular solution for databases is Two-Phase Locking which uses two types of locks. there are write-locks (Exclusive locks), in which any transaction that would write to an element in a database would obtain the lock, blocking all access to that element. The other type of lock is a read-lock (shared lock), a transaction that would read an element in the database would acquire this lock, and any other transactions that read the element would be able to access it, but all write transactions would be locked out. The main disadvantage of this solution is that in a concurrent database system, a large number of transactions can occur simultaneously and having transactions wait for lock access will cause delay and thus performance issues, especially for more complex longer running transactions.

MVCC seeks to solve this problem of performance caused by two-phase locking by forgoing locks altogether. Instead MVCC uses something called a snapshot. A snapshot is the state of the database at a particular point in time. When any transaction is sent to the database it receives a time stamped snapshot of the database when it begins. This means that multiple concurrent transactions will see different versions of the database depending when the transaction began. This makes it so that at a particular time stamp stamp when the transaction began the data would be consistent. Each transaction would then have its own snapshot of the database. When a snapshot is associated with a transaction via MVCC, it is then possible to control how concurrent transactions take place via isolation levels.

Components:

These architectural components are purely conceptual used to more generally define, MVCC architecture. They may or may not exist as some class or process within postgres. They are named in a way that makes it clearer to understand what it does with respect to the MVCC architecture. They were derived by reviewing the functional components necessary in order to process an entire transaction and commit it. Firstly there must exist a mechanism for getting the snapshot of the database. There must be some mechanism in that watches other commits occurring during a transaction, or else, read committed cannot occur. There must be some validation mechanism in order to ensure that the transaction is valid before it is committed. And lastly in order to fully process the transaction there must be some mechanism to Commit these changes to the database.

MONITOR: monitors other commits during a transaction, represents the conceptual component that check concurrent commits within a transaction

VALIDATOR: Represents the conceptual component, that checks if a transaction, is valid. If not performs some action

COMMIT: commits to database

Isolation Levels

Isolation levels for MVCC allow for finer grained control of what is expected, during concurrent transactions. The main phenomena of interest are those that act on the same logical element. The SQL specification has defined what these phenomena are:

Dirty read:

Occurs when one uncommitted transaction makes a write, to an element, and another transaction reads what it wrote before that transaction commits

[DIAGRAM]

Sequence diagram

Nonrepeatable read:

An initial transaction reads an element then at some point a separate transaction commits a write to that element. When the transaction preforms a read again, before it commits, it will get the newly update element from the other transaction’s commit

[DIAGRAM]

Phantom read:

During an aggregate transaction, that reads multiple rows, another transaction commits an update that would alter the set members such that it changes the elements that agree with the predicate.

[DIAGRAM]

Serialization Anomaly:

The order of the successfully committed transactions results in different possibilities

[DIAGRAM]

Transaction

Currently (v13.4) postgres satisfies the following isloations modes for concurrency control:

Even Finer grained control: Locks

Sometimes the MVCC solution and its isolation levels may not exactly match the business requirements. Postgres allows a variety of different locks that can help achieve the concurrent behaviour that is desired. Although it is not useful to completely understand the functionality of all the types of locks for the conceptual architecture, it is useful to know that this is a lower level control mechanism. All the available locks,are documented in ????.

[LAYER DIAGRAM?]

---------------------------------------------------------------------------------------------------------------------

[SNAPSHOT DIAGRAM OVERALL]

Shows multiple concurrent transactions, and how they interact with each other.

Concurrency Phenomena

When concurrent transactions occur on the same element, several phenomenon can occur.

Postgres can control which phenomena can occur via setting an isolation level:

Read committed:

Serial:

Although the sql specification states these phenomenon, postgres only supports the following:

ACID Compliance

Atomicity

Consistency

Isolation

Durability

Generally for concurrent architectures locks are required when writing data. A writer that obtains a lock will not allow other processes to access it. When a more complex system like a database, which can have a large number of transactions with reads and writes, traditional locks, will cause a back log of transactions, as each batch of transactions that need to write to some element are scheduled waiting for each other to release the lock, one by one. This leads to much slower performance in general. The MVCC architecture was meant to be more generalized, and its main selling point is that it does not require any locks. MVCC is centered on the concept of snapshots. Applied to a database, this meands that reads will never block writes and writes will never block reads.  
Its main advantage is that it

In concurrent systems there can be multiple simultaneous transactions containing reads and writes, many phenomena can occur when reading and writing to the same element.

use snapshots at specific times in order to handle its concurrency. This means that read and writes do not lock each other up, which allows for performance gains. Each time a transaction starts, it can refer to a snapshot of the current database.

13.1. Introduction, pg 486

MVCC DIAGRAM (draw.io)

With this concurrency control system, postgres allows you multiple different ways to finely tune what occur within the MVCC upon a read and write.

STATE DIAGRAM EXAMPLE

Isolation Levels

Controls how concurrent reads and writes are processed

Concurrent Read and Write Phenomenon

When there are multiple concurrent transactions that can read and write there are a few phenomenon that can occur:

Dirty read: A transaction reads data written by a concurrent uncommitted transaction.

Nonrepeatable read: A transaction re-reads data it has previously read and finds that data has been modified by another transaction.

Phantom Read: A transaction re-executes a query returning a set of rows that satisfy a search condition and finds that the set of rows satisfying the condition has changed due to another recently-commited transaction.

Serialization Anomaly: Ther

3 distinct isolation levels:

Read Uncommitted, Read Committed

Repeatable Read

Serializable