**Week 8: Recovery Systems**

**Overview**

Database are expected to be extraordinarily reliable. Failures can occur at many different stages of data changes, and the database must be able to handle these failures in a way that maintains the consistency of the database.

Databases use recovery systems to reliably restore databases to a consistent state after a failure. Recovery systems are also the basis for implementing transaction rollbacks and concurrency control mechanisms.

Although there are multiple approaches, the most common approach is to use a log, often called a write-ahead log (WAL). All data changes are appended to the log so the database can re-read them when restarting from a failure. Logs are effectively a data structure that stores events in order by time and only allows append, seek, and read operations. They do not allow modifying data once written. Older data is usually deleted through an expiration mechanism. When we get to distributed systems, we'll see that logs are also used there as a key part of handling failures and restoring multiple nodes to a common state.

The main downside of a log-recovery mechanism is the increase in the number of disk writes.

**Reflection Questions**

* What does the <Ti, Xi, V1, V2> for log records syntax mean?
* What are the differences between deferred and immediate modifications?
* How would undo and redo operations be implemented using a log?
* What happens when transactions are in the pre-commit and commit stages?
* What is the practical utility of checkpoints?
* How does the process for recovering from a failure work?
* How does logging complicate database buffering? What changes are needed to the database buffering approach to support recovery?
* How can log mechanisms be used to implement a backup system? What are the tradeoffs of the one-safe, two-very-safe, and two-safe commit approaches?
* What optimizations does ARIES make?