There are three side effects of acid.
Enhanced long term memory,
decreased short term memory,
and I forget the third.
- Timothy Leary

**Concurrency Control & Recovery**

- **Concurrency Control**
  - Provide correct and highly available access to data in the presence of concurrent access by large and diverse user populations
- **Recovery**
  - Ensures database is fault tolerant, and not corrupted by software, system or media failure
  - 24x7 access to mission critical data
- **Existence of CC&R allows applications be be written without explicit concern for concurrency and fault tolerance**

**Roadmap**

- **Overview (Today)**
- **Concurrency Control (2 lectures)**
- **Recovery (1-2 lectures)**

**Structure of a DBMS**

These layers must consider concurrency control and recovery (Transaction, Lock, Recovery Managers)

**Transactions and Concurrent Execution**

- **Transaction** - DBMS's abstract view of a user program or activity:
  - A sequence of reads and writes of database objects.
  - Unit of work that must commit and abort as a single atomic unit
- **Transaction Manager controls the execution of transactions.**
- **User program may carry out many operations on the data retrieved from the database, but the DBMS is only concerned with what data is read/written from/to the database.**
- **Concurrent execution of multiple transactions essential for good performance.**
  - Disk is the bottleneck (slow, frequently used)
  - Must keep CPU busy with many queries
  - Better response time

**ACID properties of Transaction Executions**

- **Atomicity**: All actions in the Xact happen, or none happen.
- **Consistency**: If each Xact is consistent and the DB starts consistent, it ends up consistent.
- **Isolation**: Execution of one Xact is isolated from that of other Xacts.
- **Durability**: If a Xact commits, its effects persist.
Atomicity and Durability

- A transaction might **commit** after completing all its actions, or it could **abort** (or be aborted by the DBMS) after executing some actions. Also, the system may crash while the transaction is in progress.

  - **Important properties**:
    - **Atomicity**: Either executing all its actions, or none of its actions.
    - **Durability**: The effects of committed transactions must survive failures.

  - **DBMS ensures the above by logging all actions**:
    - **Undo** the actions of aborted/failed transactions.
    - **Redo** actions of committed transactions not yet propagated to disk when system crashes.

Transaction Consistency

- A transaction performed on a database that is internally consistent will leave the database in an internally consistent state.

  - **Consistency of database is expressed as a set of declarative Integrity Constraints**
    - CREATE TABLE/ASSERTION statements
    - Application-level
      - E.g. Each CS106 student can only register in one project group. Each group must have 3 students.
    - **Transactions that violate ICs are aborted.**

Isolation (Concurrency)

- Concurrency is achieved by DBMS, which interleaves actions (reads/writes of DB objects) of various transactions.

  - **DBMS ensures transactions do not step onto one another**.
  - **Each transaction executes as if it was running by itself**.
    - Transaction's behavior is not impacted by the presence of other transactions that are accessing the same database concurrently.
    - **Net effect must be identical to executing all transactions for some serial order**.
    - Users understand a transaction without considering the effect of other concurrently executing transactions.

Example

- Consider two transactions (Xacts):
  - T1: BEGIN A=A+100, B=B-100 END
  - T2: BEGIN A=1.06*A, B=1.06*B END

  1st xact transfers $100 from B's account to A's
  2nd credits both accounts with 6% interest.

  Assume at first A and B each have $1000. What are the legal outcomes of running T1 and T2?

  - T1; T2 (A=1166, B=954)
  - T2; T1 (A=1160, B=960)

  In either case, A+B = $2000 *1.06 = $2120
  There is no guarantee that T1 will execute before T2 or vice-versa, if both are submitted together.

Example (Contd.)

- Consider a possible interleaved schedule:
  - T1: A=A+100, B=B-100
  - T2: A=1.06*A, B=1.06*B

  ❖ This is OK (same as T1; T2). But what about:
  - T1: A=A+100, B=B-100
  - T2: A=1.06*A, B=1.06*B

  - Result: A=1166, B=960; A+B = 2126, bank loses $6!

  - The DBMS's view of the second schedule:
    - T1: R(A), W(A), R(B), W(B)
    - T2: R(A), W(A), R(B), W(B)

Scheduling Transactions

- **Serial schedule**: Schedule that does not interleave the actions of different transactions.
- **Equivalent schedules**: For any database state, the effect (on the set of objects in the database) of executing the first schedule is identical to the effect of executing the second schedule.
- **Serializable schedule**: A schedule that is equivalent to some serial execution of the transactions.

(Note: If each transaction preserves consistency, every serializable schedule preserves consistency.)
Anomalies with Interleaved Execution

- Reading Uncommitted Data (WR Conflicts, “dirty reads”):
  - T1: R(A), W(A), R(B), W(B), Abort
  - T2: R(A), W(A), C

- Unrepeatable Reads (RW Conflicts):
  - T1: R(A), R(A), W(A), C
  - T2: R(A), W(A), C

Anomalies (Continued)

- Overwriting Uncommitted Data (WW Conflicts):
  - T1: W(A), W(B), C
  - T2: W(A), W(B), C

Lock-Based Concurrency Control

- Here’s a simple way to allow concurrency but avoid the anomalies just described...

- Two-phase Locking (2PL) Protocol
  - Each Xact must obtain a S (shared) lock on object before reading, and an X (exclusive) lock on object before writing.
  - If an Xact holds an X lock on an object, no other Xact can get a lock (S or X) on that object.
  - System can obtain these locks automatically
  - Two phases: acquiring locks, and releasing them
    - No lock is ever acquired after one has been released
    - “Growing phase” followed by “shrinking phase”.
  - Lock Manager keeps track of request for locks and grants locks on database objects when they become available.

Strict 2PL

- 2PL allows only serializable schedules but is subjected to cascading aborts.
- Example: rollback of T1 requires rollback of T2!
  - T1: R(A), W(A), Abort
  - T2: R(A), W(A), R(B), W(B)

Introduction to Crash Recovery

- Recovery Manager
  - When a DBMS is restarted after crashes, the recovery manager must bring the database to a consistent state.
  - Ensures transaction atomicity and durability.
  - Undoes actions of transactions that do not commit.
  - Redoes actions of committed transactions during system failures and media failures (corrupted disk).
  - Recovery Manager maintains log information during normal execution of transactions for use during crash recovery.

The Log

- Log consists of “records” that are written sequentially.
  - Typically chained together by Xact id.
  - Log is often duplexed and archived on stable storage.
- Log stores modifications to the database.
  - If T writes an object, write a log record with:
    - If UNDO required need “before image”
    - IF REDO required need “after image”.
  - T commits/aborts: a log record indicating this action.
- Need for UNDO and/or REDO depend on Buffer Mgr.
  - UNDO required if uncommitted data can overwrite stable version of committed data (STEAL buffer management).
  - REDO required if xact can commit before all its updates are on disk (NO FORCE buffer management).
Logging Continued

- **Write Ahead Logging (WAL) protocol**
  - Log record must go to disk before the changed page!
  - Implemented via a handshake between log manager and the buffer manager.
  - All log records for a transaction (including its commit record) must be written to disk before the transaction is considered “Committed”.
- All log related activities (and in fact, all CC related activities such as lock/unlock, dealing with deadlocks etc.) are handled transparently by the DBMS.

ARIES Recovery

- There are 3 phases in ARIES recovery:
  - **Analysis**: Scan the log forward (from the most recent checkpoint) to identify all Xacts that were active, and all dirty pages in the buffer pool at the time of the crash.
  - **Redo**: Redoes all updates to dirty pages in the buffer pool, as needed, to ensure that all logged updates are in fact carried out and written to disk.
  - **Undo**: The writes of all Xacts that were active at the crash are undone (by restoring the before value of the update, as found in the log), working backwards in the log.
- At the end --- all committed updates and only those updates are reflected in the database.
- Some care must be taken to handle the case of a crash occurring during the recovery process!

Summary

- Concurrency control and recovery are among the most important functions provided by a DBMS.
- **Concurrency control is automatic.**
  - System automatically inserts lock/unlock requests and schedules actions of different Xacts in such a way as to ensure that the resulting execution is equivalent to executing the Xacts one after the other in some order.
- Write-ahead logging (WAL) and the recovery protocol are used to undo the actions of aborted transactions and to restore the system to a consistent state after a crash.