25 Transactions

Application programs that use multiple files may have their own invariants that must hold about the state (contents) of these files. After a system crash, the system cannot restore these invariants, since it has no knowledge of them. Need to have a high-level tools that allows applications to specify legal (consistent) states of their file data.
Transactions are a high-level mechanism that allows applications to specify transitions from one legal state to another legal state. They can be used by the system to both ensure synchronization (in the event of concurrent access), and all-or-nothing semantics (in the event of system crashes.)
Transaction Primitives:

> tid = **startTransaction()**;
> {sequence of read/write operations to multiple files}
> if (anythingGoesWrong)
>   **abortTransaction(tid)**;
> else {
>   {more read/write operations}
>   **commitTransaction(tid)**;
> }

Properties of transactions:

- serializability: concurrent transactions do not interfere with each other
- atomicity: either all or no part of a transaction completes
- permanence: once a transaction completes, the changes are permanent

Implementation of transactions:

- atomicity, permanence:
  - shadowing: keep separate, private copies of each modified file blocks, leaving the original file unchanged. When the transaction commits, atomically copy the shadow pages to the original pages.

  - intention log: modify original files directly, but keep a log (diff) of all changes that were made. If the transaction aborts, use that log to undo all the changes that were made ("rollback").

- serializability: two-phase locking

There is much more to transactions. Transactions play a central role in databases.