>>

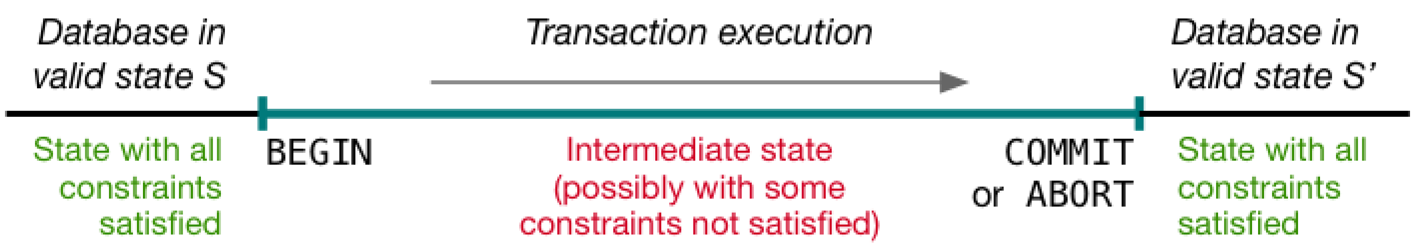
Transaction Processing

❖ Transaction Processing

A transaction (tx) is ...

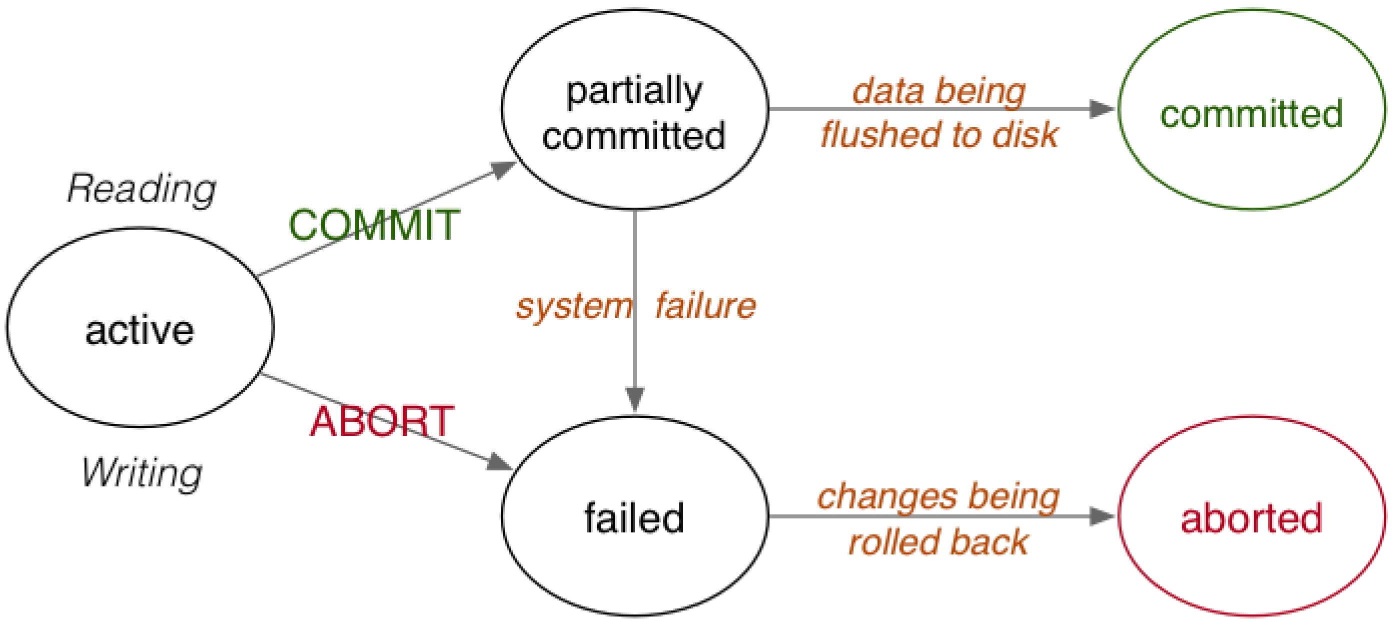
- a single application-level operation
- performed by a sequence of database operations

A transaction effects a state change on the DB



❖ Transaction Processing (cont)

Transaction states:



`COMMIT` ⇒ all changes preserved, **`ABORT`** ⇒ database unchanged

❖ Transaction Processing (cont)

Concurrent transactions are

- desirable, for improved performance (throughput)
- problematic, because of potential unwanted interactions

To ensure problem-free concurrent transactions:

- **A**tomic ... whole effect of tx, or nothing
- **C**onsistent ... individual tx's are "correct" (wrt application)
- **I**solated ... each tx behaves as if no concurrency
- **D**urable ... effects of committed tx's persist

<< ^ >>

❖ Transaction Processing (cont)

Transaction processing:

- the study of techniques for realising ACID properties

Consistency is the property:

- a tx is correct with respect to its own specification
- a tx performs a mapping that maintains all DB constraints

Ensuring this must be left to application programmers.

Our discussion focusses on: Atomicity, Durability, Isolation

<< ∧ >>

❖ Transaction Processing (cont)

Atomicity is handled by the commit and abort mechanisms

- **commit** ends tx and ensures all changes are saved
- **abort** ends tx and *undoes* changes "already made"

Durability is handled by implementing stable storage, via

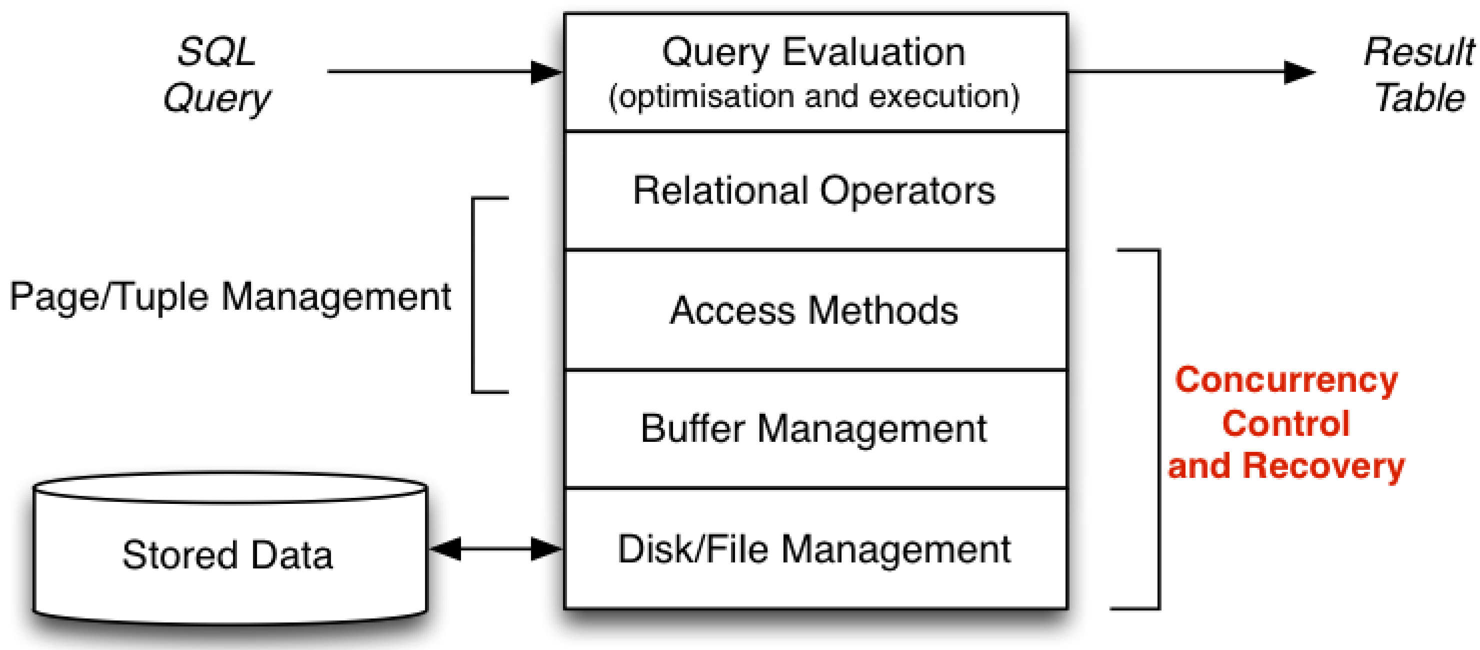
- redundancy, to deal with hardware failures
- logging/checkpoint mechanisms, to recover state

Isolation is handled by concurrency control mechanisms

- possibilities: lock-based, timestamp-based, check-based
- various levels of isolation are possible (e.g. serializable)

❖ Transaction Processing (cont)

Where transaction processing fits in the DBMS:



❖ Transaction Terminology

To describe transaction effects, we consider:

- **READ** - transfer data from "disk" to memory
- **WRITE** - transfer data from memory to "disk"
- **ABORT** - terminate transaction, unsuccessfully
- **COMMIT** - terminate transaction, successfully

Relationship between the above operations and SQL:

- **SELECT** produces **READ** operations on the database
- **UPDATE** and **DELETE** produce **READ** then **WRITE** operations
- **INSERT** produces **WRITE** operations

<< ^ >>

❖ Transaction Terminology (cont)

More on transactions and SQL

- **`BEGIN`** starts a transaction
  - the **`begin`** keyword in PLpgSQL is not the same thing
- **`COMMIT`** commits and ends the current transaction
  - some DBMSs e.g. PostgreSQL also provide **`END`** as a synonym
  - the **`end`** keyword in PLpgSQL is not the same thing
- **`ROLLBACK`** aborts the current transaction, undoing any changes
  - some DBMSs e.g. PostgreSQL also provide **`ABORT`** as a synonym

In PostgreSQL, tx's cannot be defined inside functions (e.g. PLpgSQL)

❖ Transaction Terminology (cont)

The **`READ`**, **`WRITE`**, **`ABORT`**, **`COMMIT`** operations:

- occur in the context of some transaction *T*
- involve manipulation of data items *X, Y, ...* (`READ` and `WRITE`)

The operations are typically denoted as:

$R_T(X)$	read item *X* in transaction *T*
$W_T(X)$	write item *X* in transaction *T*
A_T	abort transaction *T*
C_T	commit transaction *T*

<< ^ >>

❖ Schedules

A schedule gives the sequence of operations from $\geqslant 1$ tx

Serial schedule for a set of tx's $T_1 .. T_n$

- all operations of T_i complete before T_{i+1} begins

E.g. $R_{T_1}(A)$ $W_{T_1}(A)$ $R_{T_2}(B)$ $R_{T_2}(A)$ $W_{T_3}(C)$ $W_{T_3}(B)$

Concurrent schedule for a set of tx's $T_1 .. T_n$

- operations from individual T_i's are interleaved

E.g. $R_{T_1}(A)$ $R_{T_2}(B)$ $W_{T_1}(A)$ $W_{T_3}(C)$ $W_{T_3}(B)$ $R_{T_2}(A)$

❖ Schedules (cont)

Serial schedules guarantee database consistency

- each T_i commits before T_{i+1} starts
- prior to T_i database is consistent
- after T_i database is consistent (assuming T_i is correct)
- before T_{i+1} database is consistent ...

Concurrent schedules interleave tx operations arbitrarily

- and may produce a database that is not consistent
- after all of the transactions have committed successfully

<< ^

❖ Transaction Anomalies

What problems can occur with (uncontrolled) concurrent tx's?

The set of phenomena can be characterised broadly under:

- dirty read:
  reading data item written by a concurrent uncommitted tx
- nonrepeateable read:
  re-reading data item, since changed by another concurrent tx
- phantom read:
  re-scanning result set, finding it changed by another tx

Produced: 11 Apr 2021