Transactions

Why Transactions?

- Database systems are normally being accessed by many users or processes at the same time.
  - Both queries and modifications.
- Unlike operating systems, which support interaction of processes, a DMBS needs to keep processes from troublesome interactions.

Transactions

- A single "unit of work" in a DBMS.
- Can comprise more than one SQL command, but each individual command does not stand on its own.

Statement of Problem

- How do we allow concurrent running of independent transactions while preserving database integrity?
- Additionally, we want
  - good response time and minimal waiting.
  - correctness and fairness.

Another example: "lost update" problem

time

T1	T2
Read(N)	
	Read(N)
N=N-1	
	N= N-1
Write(N)	
	Write(N)

Concurrency

- Arbitrary interleaving can lead to
  - Temporary inconsistency (unavoidable)
  - "Permanent" inconsistency (bad!)

Example: Bad Interaction

- You and friend each take $100 from different ATMs at about the same time.
  - The DBMS had better make sure one account deduction doesn't get lost.
- Compare: An OS allows two people to edit a document at the same time. If both write, one's changes get lost.

Remember ACID?

Remember ACID?

ACID Transactions

- *We want transactions to be*:
  - ***Atomic***: Whole transaction or none is done.
  - ***Consistent***: Database constraints preserved.
  - ***Isolated***: It appears to the user as if only one transaction executes at a time.
  - ***Durable***: Effects of a transaction survive a crash.

SQL Transactions

- `BEGIN TRANSACTION`
- `// do SQL here`
- `either COMMIT or ROLLBACK`

COMMIT

- The SQL statement COMMIT causes a transaction to complete.
  - Any database modifications are now permanent in the database.

ROLLBACK

- The SQL statement ROLLBACK also causes the transaction to end, but by ***aborting***.
  - No effects on the database.
- Failures like division by 0 or a constraint violation can also cause rollback, even if the programmer does not request it.