

Transactions

- Transactions, Concurrency, Recovery
- Transactions
- Example Transaction
- Transaction Concepts
- Transaction Consistency

❖ Transactions, Concurrency, Recovery

DBMSs maintain valuable information in an environment that is:

- **shared** - concurrent access by multiple users
- **unstable** - potential for hardware/software failure

Each user should see the system as:

- **unshared** - their work is not inadvertently affected by others
- **stable** - the data survives in the face of system failures

Ultimate goal: data integrity is maintained at all times.

❖ Transactions, Concurrency, Recovery (cont)

Transaction processing

- techniques for managing "logical units of work" which may require multiple DB operations

Concurrency control

- techniques for ensuring that multiple concurrent transactions do not interfere with each other

Recovery mechanisms

- techniques to restore information to a consistent state, even after major hardware shutdowns/failures

COMP3311 only looks at the first of these

❖ Transactions

A **transaction** is

- an atomic "unit of work" in an application
- which may require multiple database changes

Transactions happen in a multi-user, unreliable environment.

To maintain integrity of data, transactions must be:

- **A**tomic - either fully completed or completely rolled-back
- **C**onsistent - map DB between consistent states
- **I**solated - transactions do not interfere with each other
- **D**urable - persistent, restorable after system failures

❖ Example Transaction

Bank funds transfer

- move N dollars from account X to account Y
- `Accounts(id, name, balance, heldAt, ...)`
- `Branches(id, name, address, assets, ...)`
- maintain `Branches. assets` as sum of balances via triggers
- transfer operation is implemented by a function which
 - has three parameters: amount, source acct, dest acct
 - checks validity of supplied accounts
 - checks sufficient available funds
 - returns a unique transaction ID on success

❖ Example Transaction (cont)

Example function to implement bank transfer ...

```
create or replace function
  transfer(N integer, Src text, Dest text)
  returns integer
declare
  sID integer; dID integer; avail integer;
begin
  select id,balance into sID,avail
  from Accounts where name=Src;
  if (sID is null) then
    raise exception 'Invalid source account %',Src;
  end if;
  select id into dID
  from Accounts where name=Dest;
  if (dID is null) then
    raise exception 'Invalid dest account %',Dest;
  end if;
  ...
```

❖ Example Transaction (cont)

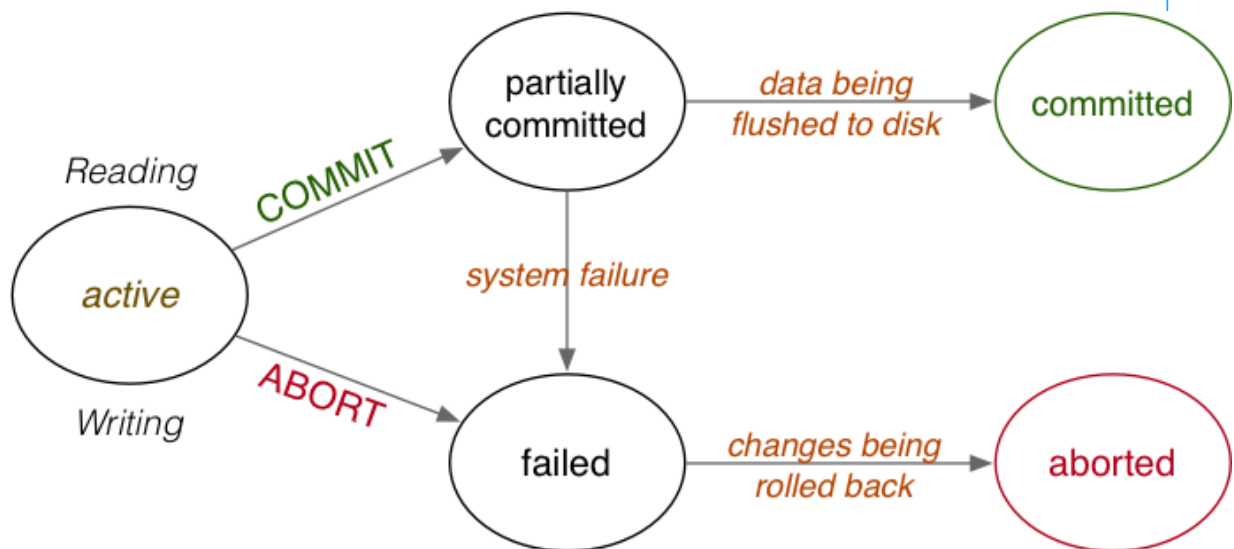
Example function to implement bank transfer (cont)...

```
...
  if (avail < N) then
    raise exception 'Insufficient funds in %', Src;
  end if;
  -- total funds in system = NNNN
  update Accounts set balance = balance-N
  where id = sID;
  -- funds temporarily "lost" from system
  update Accounts set balance = balance+N
  where id = dID;
  -- funds restored to system; total funds = NNNN
  return nextval('tx_id_seq');
end;
```

❖ Transaction Concepts

A transaction must always terminate, either:

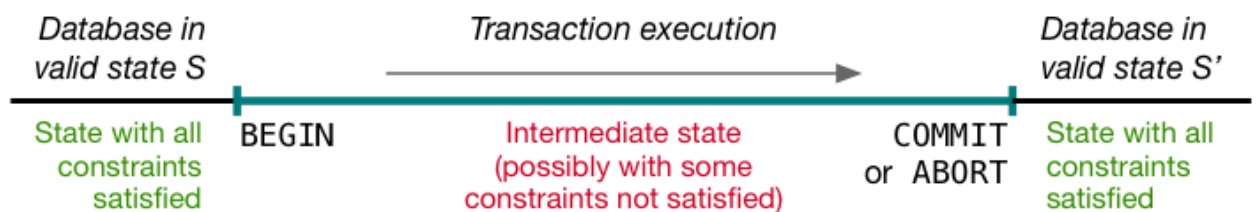
- successfully (COMMIT), with all changes preserved
- unsuccessfully (ABORT), with database unchanged



❖ Transaction Consistency

Transactions typically have intermediate states that are invalid.

However, states **before** and **after** transaction must be valid.



Valid = consistent = satisfying all stated constraints on the data

Produced: 25 Mar 2021