CMPT 321
FALL 2017

Transactions

Lecture 05.04

By Marina Barsky

Integrity or correctness of data

We would like data to be “accurate” or “correct” at all times

EMP table

Name	Age
White	52
Green	3421
Gray	1

Integrity or consistency constraints

- Predicates that data must satisfy

For example:

- x is key of relation R
- Domain(x) = {Red, Blue, Green}
- no employee should make more than twice the average salary

Definition:

- *Consistent state*: satisfies all constraints
- *Consistent DB*: DB in consistent state

Integrity constraints may not capture "full correctness"

Implicit (business) constraints:

- When salary is updated,

  new salary > old salary

- When account record is deleted,

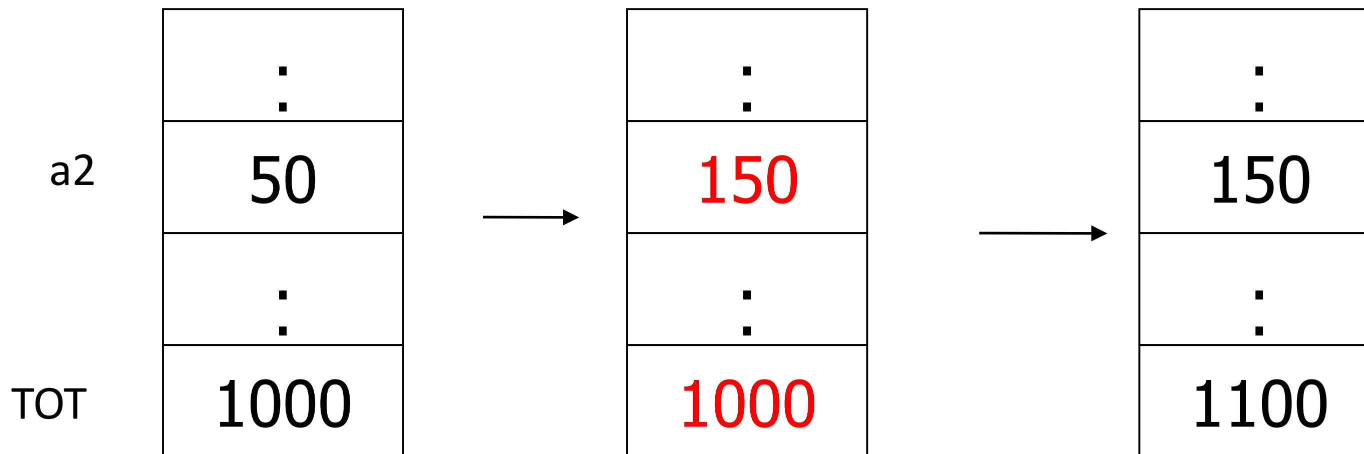
  balance = 0

Observation:
DB cannot be consistent always

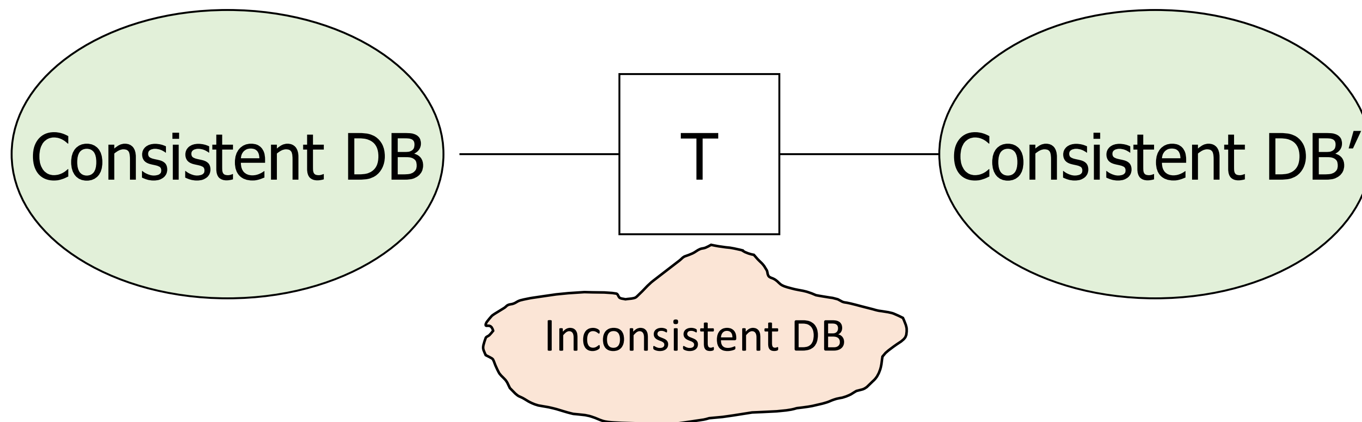
Example: $a1 + a2 + \ldots . \; an = TOT$ (constraint)

Deposit $100 in a2: $a2 \leftarrow a2 + 100$

$TOT \leftarrow TOT + 100$



Transaction: collection of actions that bring DB from one consistent state to another



If T starts with consistent state + T executes in isolation

$\Rightarrow$T leaves consistent state

Concurrent transactions

- In production environments, it is unlikely that we can limit our system to just one user at a time.
  - Consequently, it is possible for multiple queries to be submitted at approximately the same time.

- If all of the queries were very small (i.e., in terms of time), we could probably just execute them serially, on a first-come-first-served basis.

SERIALLY – ONE AFTER ANOTHER

Queries are executed “simultaneously”

- However, many queries are both complex and time consuming.
  - Executing these queries would make other queries wait a long time for a chance to execute.
  - Disk usage can be optimized for several queries running in parallel
- So, in practice, the DBMS may be running many different queries at about the same time.

INTERLEAVING QUERY PROCESSING

Concurrent Transactions

- Unlike operating systems, which *support* interaction of processes, a DMBS needs to keep processes from troublesome interactions.

- Even when there is **no "failure", several** transactions can **interact** to turn a
  consistent state
  into an
  inconsistent state.

Example: two people - one bank account

- Before withdrawing money, each needs to check if the balance is sufficient
- Initially there is **100**$ on the account

Ryan	Monica
READ(X, b)	
b=100	
	READ(X, c)
	c = 100
	c - = 50
	WRITE (X, c)
b - = **100**	
WRITE (X, Ryan)	

Monica: thinks 50 $ left

Ryan: thinks 0 $ left

In fact, the withdrawn amount is 150$

Example: two people - one bank account

- Before withdrawing money, each needs to check if the balance is sufficient
- Initially there is **100**$ on the account

Ryan	Monica
READ(X, b)	
b=100	
	READ(X, c)
	c = 100
	c - = 50
	WRITE (X, c)
b - = **100**	
WRITE (X, b)	

The problem is that the reading and writing operations should be performed as one ***transaction***, their combination should be **atomic**

Transaction

- DBMS groups your SQL statements into ***transactions***.
- The ***transaction*** is the atomic unit of execution of database operations
- By default, each query or DML statement is a transaction
- User can group multiple SQL statements into a single transaction

Transactions with SQL

START TRANSACTION; (BEGIN;)

…SQL statements

COMMIT; (END;)

End of a transaction

- The transaction ends when one of the following occurs:
  - A COMMIT or ROLLBACK are issued
  - A DDL (CREATE, ALTER, DROP …) or DCL (GRANT, REVOKE) statement is issued
  - A user properly exits (COMMIT)
  - System crashes (ROLLBACK)

COMMIT and ROLLBACK

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

- 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 did not request it.

Banking example: DB terminal

Assuming we defined a CHECK constraint on balance >=0

Ryan

```
BEGIN;
 SELECT balance
 FROM accounts
 WHERE
 account_name = "Monica and Ryan";
 UPDATE accounts
 SET balance = balance – 100
 WHERE
 account_name = "Monica and Ryan";
COMMIT;
```

Monica

```
BEGIN;
 SELECT balance
 FROM accounts
 WHERE
 account_name = "Monica and Ryan";
 UPDATE accounts
 SET balance = balance – 50
 WHERE
 account_name = "Monica and Ryan";
COMMIT;
```

Failure – constraint violated

Transaction should have ACID

- ***A**tomicity*: Whole transaction or none is done.
- ***C**onsistency*: Database constraints preserved. Transaction, executed completely, takes database from one *consistent* state to another
- ***I**solation*: It appears to the user as if only one process executes at a time.
  - That is, even though actions of several transactions might be interleaved, the net effect is identical to executing all transactions one after another in some **serial order**.
- ***D**urability*: Effects of a process survive a crash.