# 10 Transactions

Saturday, January 12, 2019 2:04 PM

Describe local and distributed ACID transactions Discuss some of the methods used in local and distributed transactions.
[Coulouris et al. 16.1 – 16.2, 16.4 – 16.5, 17.2, 17.3.1]
---

- Atomic: indivisible - all or nothing
  - Atomic: x=7
  - Non-atomic: x+=7 (that's two operations: addition, then updating x)
- Consistent: all constraints are true at commit
- Isolated: transactions don't influence each other (until they're committed)
- Durable: transaction is saved at commit

## Problems

- Lost update: an update is lost because the transaction is working on old data
- Inconsistent read: reading from different versions of data
- Dirty read: reading from a transaction that was later rolled back (cancelled)

## Fixing the problems

- Pessimistic locking
  - Lock everything you touch (read or write)
  - Only release it when you're completely done
  - Examples
    - Windows file system
    - Database on serialized isolation level
- Optimistic locking
  - Pessimistic locking can lead to bottlenecks where the system bogs down
  - Data is versioned
  - You're not allowed to write if you have an old version
    - Do over if wrong version
    - Ex: git, database with read committed isolation level
  - A global version number is more 'sound' for the data, usually it's not necessary to track the version of each table or row separately.
  - You don't need to download the entire database if the database on the server is ahead, you just download the data you need to affect
- Avoid deadlocks:
  - lock all editable resources at transaction start (simple but not effective)
  - Use timeouts

### 2 Phase Locking

- Phase 1: grab resource (where you read from)
- Phase 2: use resource (write your data)

### Conservative 2 phase locking (used in pessimistic locking)

- P1: grab all resources (needs to be synchronized in Java)
- P2: use all resources

Two databases

| DB1 | DB2 |
|---|---|
| Begin transaction | |
| | Begin transaction |

| | |
|---|---|
| UPDATE | |
| | UPDATE |
| Commit | |
| | commit |

## One-phase commit

1. Do the work
2. While not all changes are committed
    a. For all uncommitted participant p
        i. p.commit()

- If one of the commits will not satisfy the DB constraints, this will keep on failing; this is fixed by two-phase commit

## Two-phase commit

1. Do the work
2. For all participants
    a. Ask db: "can you commit?"
3. If yes: while all changes are committed
    a. For uncommitted participants p
        i. P.commit();

## Java Transaction API (JTA)

- Has all the code that happens in the coordinator
- Participant databases need to be jta ready (drivers)