Recitation 18: Databases

MIT - 6.033

Spring 2022

Henry Corrigan-Gibbs

[Based on Sam Madden's 2007 lecture notes]

Plan

* DBs & Transactions
* Durability
* Concurrent access

Logistics

* DP pres grades out next week
* DB hands on out 4/14
* No lecture 4/18 for Patriots Day

Database

- Collection of tables (rows, cols)
- High-level language (SQL) for reading/writing data in tables

If you haven't yet used it, you will be surprised & delighted!

Students

name	ID	course	...

piece of data

- Client can group together a sequence of actions into a transaction

```
BEGIN TRANSACTION

A = 50
B = A
B = B + 1

COMMIT / ABORT
```

In reality, use SQL

- All that the DB system cares about is the reads & writes.

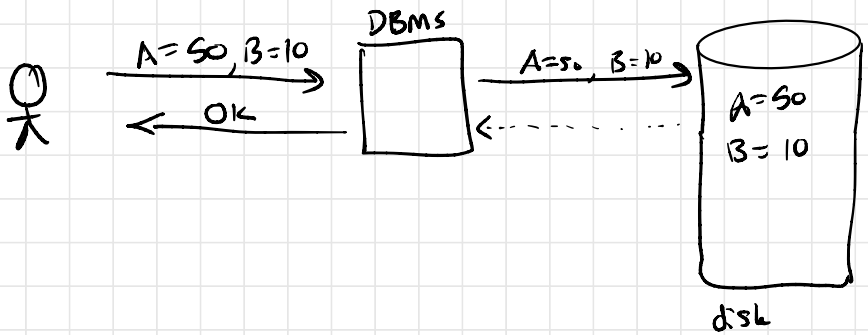
Two things to worry about

1. Crashes / Durability
   - ↳ Want committed data to persist on disk ("non-volatile" storage)

2. Concurrent access to data
   - ↳ Want each transaction to appear to execute in sequence (sort of) ***

Durability

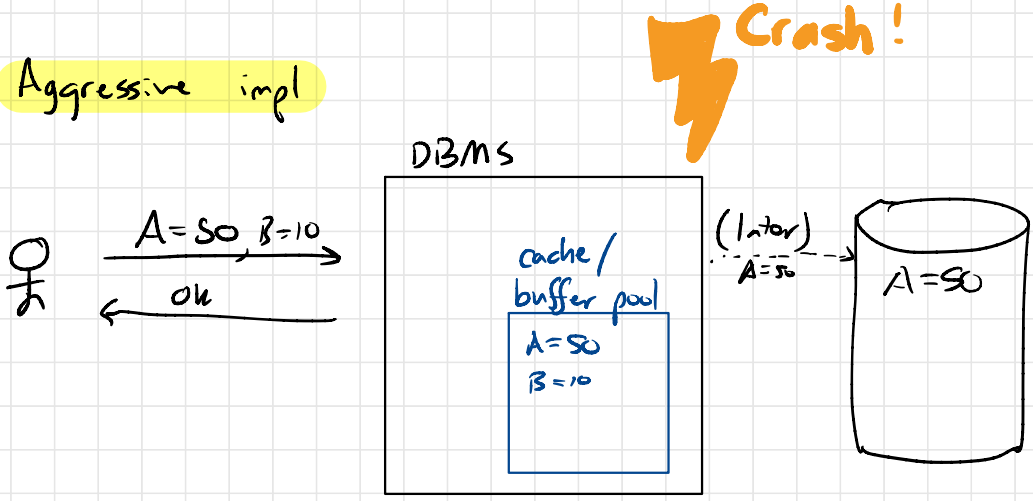
Simplest implementation



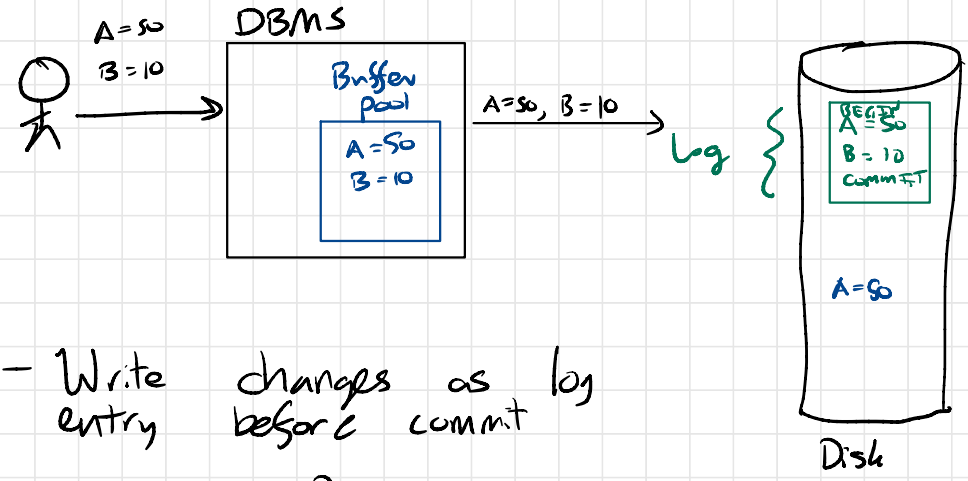
Problem: Performance!

RAM read: 100 ns
Disk seek: 10,000,000 ns

Aggressive impl



Write-ahead log (Very common/useful idea!)



- Write changes as log entry before commit
- Why better?
  - (a) Writes are large & sequential
  - (b) Repeated access to same obj doesn't touch disk

After crash, inspect log
- → "Undo" uncommitted partial txns
- → "Redo" committed txns

To undo, need to store old & new value of each record.

[Similar ideas show up in other contexts...

Recovery

- REDO actions from log
  - ↳ Disk now in state as before crash.
- UNDO aborted txns
  - ↳ Find first such one, roll back

Log

```
BEGIN T1
BEGIN T2
A = 50    (old: 40) - T1
BEGIN T3
C = 20    (old: 0) - T2
B = 10    (old: 0) - T3
COMMIT T1
ABORT T2
```

In groups, walk through this recovery process.

What happens if you crash during recovery?

Log can grow LARGE

- ↳ Checkpoints to speed up recovery.

Concurrent access

- Two different transactions should ideally appear to execute serially (conflict serializability)

```
Abal = Read(A)
Abal -= 50
Write(A, Abal)
```

```
Abal = Read(A)
Bbal = Read(B)
Print(Abal + Bbal)
```

```
Bbal = Read(B)
Bbal += 50
Write(B, Bbal)
```

- Most common technique: Locking
  - ↳ Covered in lecture.

- Two-phase locking
  - → Each data item has lock

I. Growing: Ack all locks needed → In fixed order! o.w. deadlock
   Do stuff
II. Shrinking: Release all locks

Locking

Why are we not done? Performance!

If you txn sums value of all rows in a table, you will prevent any other action on table for a while.

↳ Must hold locks on all table rows

↳ Action of locking & unlocking can be costly.

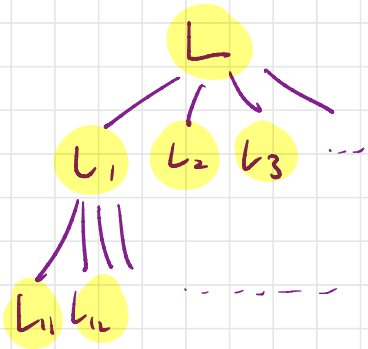
How to address this?

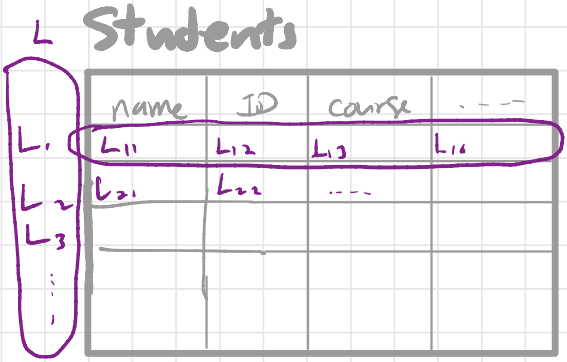
↳ Discuss in groups

Two ideas:

1. Coarse-grained locks (hierarchical)
   ↳ Reduce # of locks you need to acquire
2. Lock less.
   ↳ Release locks early to increase amt of concurrency

Hierarchical Locking





Can lock/unlock entire region at once
↳ Typically a "page" of values

"Intent locks" (see paper) allow extra concurrency

Relaxed Isolation

Postgres DB uses "read committed" by default

↳ Txn sees values of writes committed by other txns

```
BEGIN
Aval = Read(A)
Write(B,10)
Aval = Read(A)
COMMIT
```

could return diff vals

→ Useful b/c allows releasing read locks early

Wrap up

Two challenges:

- Recovering from failures
- Doing many things at once

Both are easy if you don't care about performance.

If you do, ... trade-offs!