L15-L16: Transactions & Concurrency Control

CS3200 Database design (sp18 s2)

3/12/2018

L15: Transactions & Concurrency Control
Part 1: Transactions & Logging

CS3200 Database design (sp18 s2)

3/12/2018

Goals this part of lectures

- **Transactions** are a programming abstraction that enables the DBMS to handle recovery and concurrency for users.

- Application: Transactions are critical for users
  - Even casual users of data processing systems!

- Fundamentals: The basics of how TXNs work
  - Transaction processing is part of the debate around new data processing systems

  - Give you enough information to understand how TXNs work, and the main concerns with using them

balance? $500

balance? $500

withdraw $300

withdraw $300

Some example of what can go wrong

Dirty Reads

Write-Read Conflict

T_2 reads a data object previously written but <u>uncommitted</u> by T_1

T_1: WRITE(A)	
	T_2: READ(A)
T_1: ABORT	

Inconsistent Read

Write-Read Conflict

T_2 reads a data object previously written but not finished by T_1

```
T1:   A := 20;   B := 20;
T1:   WRITE(A)


T1:   WRITE(B)
```

```

T2:   READ(A);
T2:   READ(B);

```

Unrepeatable Read

Read-Write Conflict

T_1: WRITE(A)	T_2: READ(A); T_2: READ(A);

Lost Update

T_2 overwrites uncommitted data by T_1

Write-Write Conflict

T_1: READ(A)	
T_1: A := A+5	
T_1: WRITE(A)	
	T_2: READ(A);
	T_2: A := A*1.3
	T_2: WRITE(A);

Intro to Transactions & Logging

1) Transactions
2) Properties of Transactions: ACID
3) Logging

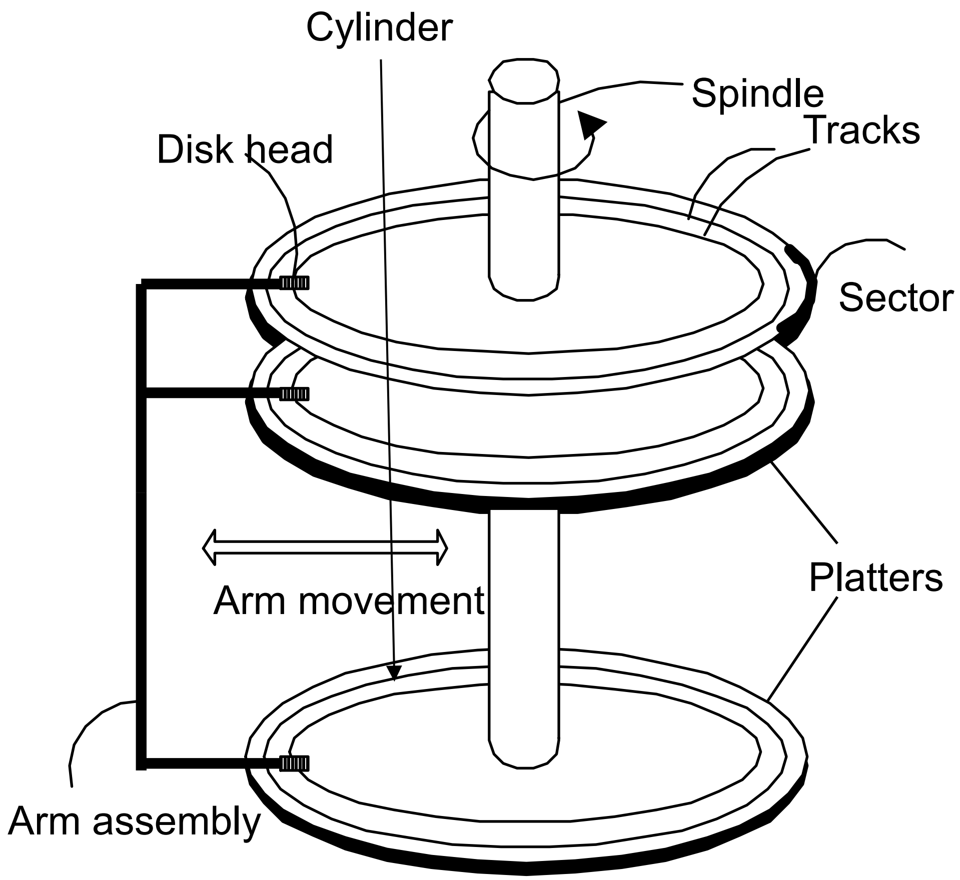
1. Transactions

What we will learn about next

- Our “model” of the DBMS / computer
- Transactions basics
- Motivation: Recovery & Durability
- Motivation: Concurrency

High-level: Disk vs. Main Memory

- Disk:
  - Slow
    - Sequential access
      - (although fast sequential reads)
  - Durable
    - We will assume that once on disk, data is safe!
  - Cheap



High-level: Disk vs. Main Memory

- Random Access Memory (RAM) or Main Memory:
  - Fast
    - Random access, byte addressable
      - ~10x faster for sequential access
      - ~100,000x faster for random access!
  - Volatile
    - Data can be lost if e.g. crash occurs, power goes out, etc!
  - Expensive
    - For $100, get 16GB of RAM vs. 2TB of disk!

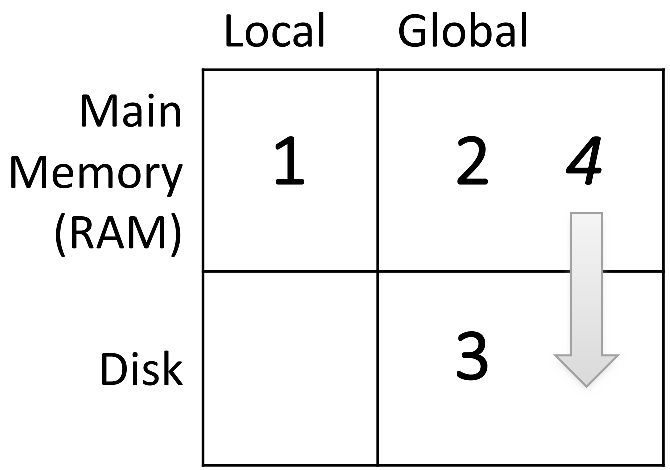
Our model: Three Types of Regions of Memory

1. <u>Local</u>: In our model each process in a DBMS has its own local memory, where it stores values that only it "sees"

2. <u>Global</u>: Each process can read from / write to shared data in main memory

3. <u>Disk</u>: Global memory can read from / flush to disk

4. *<u>Log</u>: Assume on stable disk storage- spans both main memory and disk…*



Log is a *sequence* from main memory -> disk

"<u>**Flushing**</u> to disk" = writing to disk from main memory

High-level: Disk vs. Main Memory

- Keep in mind the tradeoffs here as motivation for the mechanisms we introduce
  - Main memory: fast but limited capacity, volatile
  - Vs. Disk: slow but large capacity, durable

How do we effectively utilize ***both*** ensuring certain critical guarantees?

Transactions

Transactions: Basic Definition

A **transaction ("TXN")** is a sequence of one or more ***operations*** (reads or writes) which reflects ***a single real-world transition***.

In the real world, a TXN either happened completely or not at all

```
START TRANSACTION
    UPDATE Product
    SET Price = Price – 1.99
    WHERE pname = ‘Gizmo’
COMMIT
```

Transactions: Basic Definition

A <u>**transaction ("TXN")**</u> is a sequence of one or more ***operations*** (reads or writes) which reflects ***a single real-world transition***.

In the real world, a TXN either happened completely or not at all

<u>Examples:</u>

- Transfer money between accounts
- Purchase a group of products
- Register for a class (either waitlist or allocated)

Transactions in SQL

- In “ad-hoc” SQL:
  - Default: each statement = one transaction

- In a program, multiple statements can be grouped together as a transaction:

```
START TRANSACTION
     UPDATE Bank SET amount = amount – 100
     WHERE name = ‘Bob’
     UPDATE Bank SET amount = amount + 100
     WHERE name = ‘Joe’
COMMIT
```

Model of Transaction in our class

- Note: we assume that the DBMS only sees reads and writes to data
  - User may do much more
  - In real systems, databases do have more info...

Motivation for Transactions

Grouping user actions (reads & writes) into transactions helps with two goals:

1. Recovery & Durability: Keeping the DBMS data consistent and durable in the face of crashes, aborts, system shutdowns, etc.

2. Concurrency: Achieving better performance by parallelizing TXNs without creating anomalies

Our first focus

Motivation

1. Recovery & Durability of user data is essential for reliable DBMS usage

- The DBMS may experience crashes (e.g. power outages, etc.)
- Individual TXNs may be aborted (e.g. by the user)

Idea: Make sure that TXNs are either **durably stored in full**, **or not at all**; keep log to be able to "roll-back" TXNs

Protection against crashes / aborts

```
Client 1:
        INSERT INTO SmallProduct(name, price)
                SELECT pname, price
                FROM Product
                WHERE price <= 0.99

        DELETE Product
                WHERE price <=0.99
```

Crash / abort!

What goes wrong?

Protection against crashes / aborts

```
Client 1:
        START TRANSACTION
                INSERT INTO SmallProduct(name, price)
                        SELECT pname, price
                        FROM Product
                        WHERE price <= 0.99

                DELETE Product
                        WHERE price <=0.99
        COMMIT OR ROLLBACK
```

Now we'd be fine! We'll see how / why this lecture

Motivation

- 2. Concurrent execution of user programs is essential for good DBMS performance.

  - Disk accesses may be frequent and slow- optimize for throughput (# of TXNs), trade for latency (time for any one TXN)

  - Users should still be able to execute TXNs as if in isolation and such that consistency is maintained

Idea: Have the DBMS handle running several user TXNs concurrently, in order to keep CPUs humming…

Multiple users: single statements

```
Client 1: UPDATE Product
          SET Price = Price – 1.99
          WHERE pname = ‘Gizmo’



Client 2: UPDATE Product
          SET Price = Price*0.5
          WHERE pname = ‘Gizmo’
```

Two managers attempt to discount products *concurrently*-
What could go wrong?

Multiple users: single statements

```
Client 1: START TRANSACTION
                  UPDATE Product
                  SET Price = Price – 1.99
                  WHERE pname = ‘Gizmo’
          COMMIT

Client 2: START TRANSACTION
                  UPDATE Product
                  SET Price = Price*0.5
                  WHERE pname = ‘Gizmo’
          COMMIT
```

Now works like a charm- we’ll see how / why next lecture…

2. Properties of Transactions

ACID

What we learn about next: ACID

- Atomicity
- Consistency
- Isolation
- Durability

Transaction Properties: ACID

a-tomos: undividable

- Atomic
  - State shows either all the effects of txn, or none of them
- Consistent
  - Txn moves from a state where integrity holds, to another where integrity holds
- Isolated
  - Effect of txns is the same as txns running one after another (ie looks like batch mode)
- Durable
  - Once a txn has committed, its effects remain in the database

ACID continues to be a source of great debate!
BASE (Basic Availability, Soft-state, Eventual Consistency)

ACID: Atomicity

- TXN's activities are atomic: all or nothing
  - Intuitively: in the real world, a transaction is something that would either occur completely or not at all

- Two possible outcomes for a TXN
  - It commits: all the changes are made
  - It aborts: no changes are made

ACID: Consistency

- The tables must always satisfy user-specified integrity constraints
  - Examples:
    - Account number is unique
    - Stock amount can't be negative
    - Sum of debits and of credits is 0

- How consistency is achieved:
  - Programmer makes sure a txn takes a consistent state to a consistent state
  - System makes sure that the txn is atomic

ACID: Isolation

- A transaction executes concurrently with other transactions
- Isolation: the effect is as if each transaction executes in isolation of the others.
  - E.g. Should not be able to observe changes from other transactions during the run

Isolation failure

Write-Write Conflict

T_1: A := A−1

T_2: B := B−2

T_2: A := A+2

Crash / abort!

T_1: B := B+1

ACID: Durability

- The effect of a TXN must continue to exist ("persist") after the TXN
  - And after the whole program has terminated
  - And even if there are power failures, crashes, etc.
  - And etc…
- Means: Write data to disk

Change on the horizon?
Non-Volatile Ram (NVRam).
Byte addressable.