CS162
Operating Systems and Systems Programming
Lecture 18
Transactions

November 6, 2013

Anthony D. Joseph and John Canny
http://inst.eecs.berkeley.edu/~cs162

Quiz 18.1: Flow-Control

- Q1: True _ False _ Flow control is responsible for detecting packet losses and retransmissions
- Q2: True _ False _ Flow control always allows a sender to resend a lost packet
- Q3: True _ False _ With TCP, the receiving OS can deliver data to the application out-of-sequence (i.e., with gaps)
- Q4: True _ False _ Flow control makes sure the sender doesn't overflow the receiver

Quiz 18.1: Flow-Control

- Q1: True _ False X Flow control is responsible for detecting packet losses and retransmissions
- Q2: True X False _ Flow control always allows a sender to resend a lost packet
- Q3: True _ False X With TCP, the receiving OS can deliver data to the application out-of-sequence (i.e., with gaps)
- Q4: True X False _ Flow control makes sure the sender doesn't overflow the receiver

Goals for Today

- What is a database?
- Transactions (ACID semantics)

Note: Some slides and/or pictures in the following are adapted from lecture notes by Mike Franklin.

What is a Database

- A large **integrated collection** of data
- Models real world, e.g., enterprise
  - Entities (e.g., teams, games)
  - Relationships, e.g.,
    Cal plays against Stanford in The Big Game

Key Concept: Structured Data

- A data model is a collection of entities and their relationships
- A schema is an instance of a data model
  - E.g., describes the fields in the database; how the database is organized
- A relational data model is the most used data model
  - Relation, a table with rows and columns
  - Every relation has a schema which describes the fields in the column

Example: University Database

- Conceptual schema:

  Students(sid*: string*, name*: string*, email*: string*, age*: integer*, gpa*:real*)

  Courses(cid*: string*, cname*:string*, credits*:integer*)

  Enrolled(sid*:string*, cid*:string*, grade*:string*)

  FOREIGN KEY sid REFERENCES Students

  FOREIGN KEY cid REFERENCES Courses

- External Schema (View):

  Course_info(cid:string,enrollment:integer)

  Create View Course_info AS

  SELECT cid, Count (*) as enrollment

  FROM Enrolled

  GROUP BY cid

Example: An Instance of Students Relation

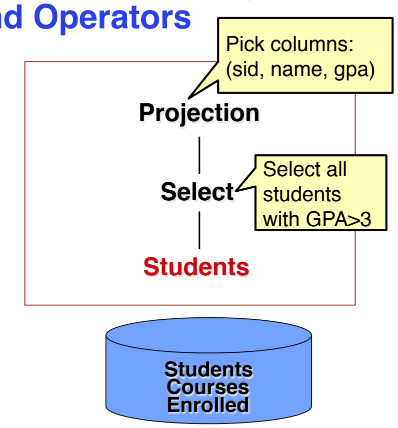
sid	name	email	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

What is a Database System?

- A **Database Management System (DBMS)** is a software system designed to store, manage, and facilitate access to databases.
- A DBMS provides:
  - Data Definition Language (DDL)
    - » Define relations, schema
  - Data Manipulation Language (DML)
    - » Queries – to retrieve, analyze and modify data.
  - Guarantees about durability, concurrency, semantics, etc

Key Concepts: Queries, Query Plans, and Operators

```
SELECT sid, name, gpa
FROM Students S
WHERE S.gpa > 3
```

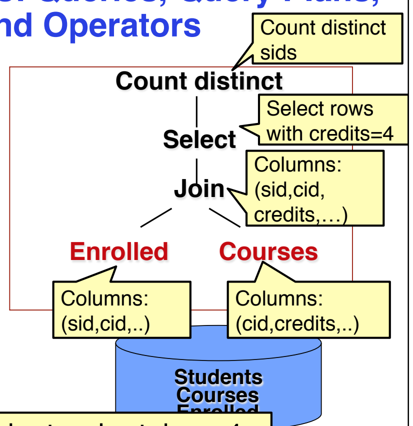


System handles query plan generation & optimization; ensures correct execution.

Select all students with GPA > 3.0

Key Concepts: Queries, Query Plans, and Operators

```
SELECT
   COUNT DISTINCT (E.sid)
FROM Enrolled E, Courses C
WHERE E.cid = C.cid
   AND C.credits = 4
```



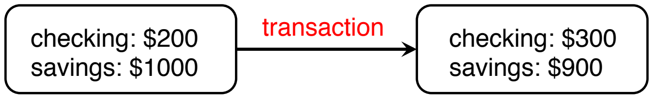
System handles query plan generation & optimization; ensures correct execution.

Number of students who take a 4 credit class

Key concept: Transaction

- An atomic sequence of database actions (reads/writes)
- Takes DB from one consistent state to another

Example



- Here, *consistency* is based on our knowledge of banking "semantics"
- In general, up to writer of transaction to ensure transaction preserves consistency
- DBMS provides (limited) automatic enforcement, via integrity constraints (IC)
  - e.g., balances must be >= 0

From Multiprogramming to Transactions

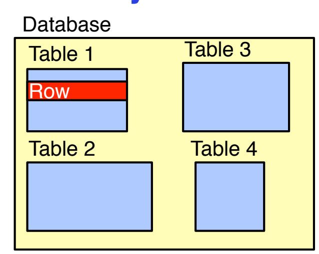
- Users would like the illusion of running their programs on the machine alone
  - Why not run the entire program in a critical section?
- Users want fast response time and operators want to increase machine utilization → increase concurrency
  - Interleave executions of multiple programs
- How can DBMS help?

Concurrent Execution & Transactions

- Concurrent execution essential for good performance
  - Disk slow, so need to keep the CPU busy by working on several user programs concurrently
- DBMS only concerned about what data is read/written from/to the database
  - Not concerned about other operations performed by program on data
- **Transaction** – DBMS's abstract view of a user program, i.e., a sequence of reads and writes.

Locking Granularity

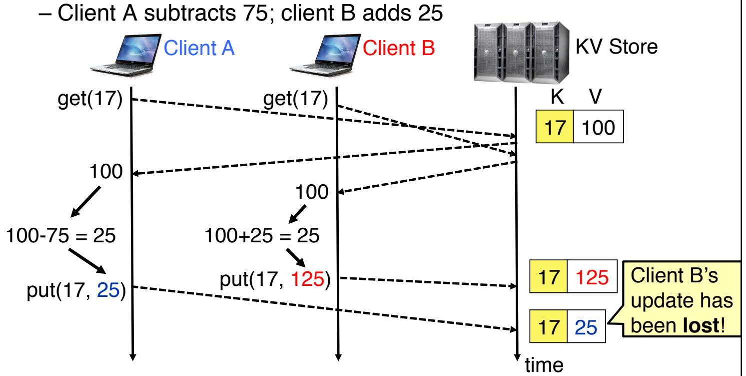
- What granularity to lock?
  - Database
  - Tables
  - Rows



- Fine granularity (e.g., row) → high concurrency
  - Multiple users can update the database and same table simultaneously
- Coarse granularity (e.g., database, table) → simple, but low concurrency

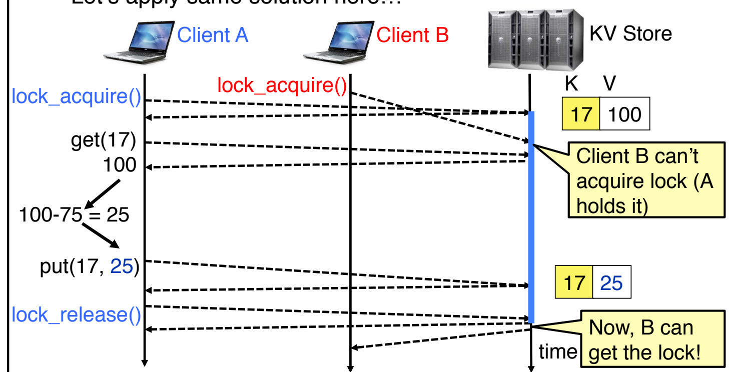
Need for Transactions in Distributed Systems

- Example: assume two clients updating same value in a key-value (KV) store at the same time
  - Client A subtracts 75; client B adds 25



Solution?

- How did we solve such problem on a single machine?
  - Critical section, e.g., use locks
  - Let's apply same solution here…



Discussion

- How does client B get the lock?
  - Polling: periodically check whether the lock is free
  - KV storage system keeps a list of clients waiting for the lock, and gives the lock to next client in the list
- What happens if the client holding the lock crashes?
- Network latency might be higher than update operation
  - Most of the time in critical section spent waiting for messages
- What is the lock granularity?
  - Do you lock every key? Do you lock the entire storage?
  - What are the tradeoffs?

Better Solution

- Interleave reads and writes from different clients
- Provide the same semantics as clients were running one at a time
- **Transaction** – database/storage sytem's abstract view of a user program, i.e., a sequence of reads and writes

“Classic” Example: Transaction

```
BEGIN;    --BEGIN TRANSACTION
 UPDATE accounts SET balance = balance -
   100.00 WHERE name = 'Alice';

 UPDATE branches SET balance = balance -
   100.00 WHERE name = (SELECT branch_name
   FROM accounts WHERE name = 'Alice');

 UPDATE accounts SET balance = balance +
   100.00 WHERE name = 'Bob';

 UPDATE branches SET balance = balance +
   100.00 WHERE name = (SELECT branch_name
   FROM accounts WHERE name = 'Bob');
COMMIT;    --COMMIT WORK
```

Transfer $100 from Alice’s account to Bob’s account

The ACID properties of Transactions

- **Atomicity:** all actions in the transaction happen, or none happen
- **Consistency:** transactions maintain data integrity, e.g.,
  - Balance cannot be negative
  - Cannot reschedule meeting on February 30
- **Isolation:** execution of one transaction is isolated from that of all others; no problems from concurrency
- **Durability:** if a transaction commits, its effects persist despite crashes

Atomicity

- A transaction
  - might *commit* after completing all its operations, or
  - it could *abort* (or be aborted) after executing some operations
- Atomic Transactions: a user can think of a transaction as always either *executing all its* operations, or *not executing any* operations at all
  - Database/storage system *logs* all actions so that it can *undo* the actions of aborted transactions

Consistency

- Data follows integrity constraints (ICs)
- If database/storage system is consistent before transaction, it will be after
- System checks ICs and if they fail, the transaction rolls back (i.e., is aborted)
  - A database enforces some ICs, depending on the ICs declared when the data has been created
  - Beyond this, database does not understand the semantics of the data (e.g., it does not understand how the interest on a bank account is computed)

Isolation

- Each transaction executes as if it was running by itself
  - It cannot see the partial results of another transaction
- Techniques:
  - Pessimistic – don't let problems arise in the first place
  - Optimistic – assume conflicts are rare, deal with them *after* they happen

Durability

- Data should survive in the presence of
  - System crash
  - Disk crash → need backups
- All committed updates and only those updates are reflected in the database
  - Some care must be taken to handle the case of a crash occurring during the recovery process!

Quiz 18.2: Databases

- Q1: True _ False _ A relational data model is the most used data model
- Q2: True _ False _ Transactions are not guaranteed to preserve the consistency of a storage system
- Q3: True _ False _ A DBMS uses a log to implement atomicity
- Q4: True _ False _ Durability isolates the reads and writes of a transaction from all other transactions

5min Break

Quiz 18.2: Databases

- Q1: True X False _ A relational data model is the most used data model
- Q2: True _ False X Transactions are not guaranteed to preserve the consistency of a storage system
- Q3: True X False _ A DBMS uses a log to implement atomicity
- Q4: True _ False X Durability isolates the reads and writes of a transaction from all other transactions

This Lecture

- Deal with **(I)solation**, by focusing on **concurrency control**
- Next lecture focus on (A)tomicity, and partially on (D)urability

Example

- Consider two transactions:
  - T1: moves $100 from account A to account B

```
T1:A := A-100; B := B+100;
```

  - T2: moves $50 from account B to account A

```
T2:A := A+50;  B := B-50;
```

- Each operation consists of (1) a read, (2) an addition/ subtraction, and (3) a write
- Example: A = A-100

```
Read(A); // R(A)
A := A - 100;
Write(A); // W(A)
```


Example (cont'd)

- Database only sees reads and writes

		Database View
`T1: A:=A-100; B:=B+100;`	→	`T1:R(A),W(A),R(B),W(B)`
`T2: A:=A+50; B:=B-50;`	→	`T2:R(A),W(A),R(B),W(B)`

- Assume initially: A = $1000 and B = $500
- What is the legal outcome of running T1 and T2?
  - A = $950
  - B = $550

Example (cont'd)

```
T1: A:=A-100; B:=B+100;
T2: A:=A+50; B:=B-50;
```

Initial values:
A:=1000
B:=500

- What is the outcome of the following execution?

```
T1:R(A),W(A),R(B),W(B)
T2:                      R(A),W(A),R(B),W(B)
```

A=900, B=600, A=950, B=550

- What is the outcome of the following execution?

```
T1:                      R(A),W(A),R(B),W(B)
T2:R(A),W(A),R(B),W(B)
```

A=1050, B=450, A=950, B=550

Example (cont'd)

```
T1: A:=A-100; B:=B+100;
T2: A:=A+50; B:=B-50;
```

Initial values:
A:=1000
B:=500

- What is the outcome of the following execution?

```
T1:R(A),W(A),                         R(B),W(B)
T2:           R(A),W(A),R(B),W(B)
```

A=900, A=950, B=450, B=550

- What is the outcome of the following execution?

```
T1:R(A),                      W(A),R(B),W(B)
T2:     R(A),W(A),R(B),W(B)
```

A=1050, B=450, A=900, B=550

Lost $50!

Transaction Scheduling

- Why not run only one transaction at a time?
- Answer: low system utilization
  - Two transactions cannot run simultaneously even if they access different data

Goals of Transaction Scheduling

- Maximize system utilization, i.e., concurrency
  - Interleave operations from different transactions
- Preserve transaction semantics
  - Semantically equivalent to a **serial schedule**, i.e., one transaction runs at a time

T1: R, W, R, W **T2:** R, W, R, R, W

Serial schedule (T1, then T2):
R, W, R, W, R, W, R, R, W

Serial schedule (T2, then T1):
R, W, R, R, W, R, W, R, W

Two Key Questions

1) Is a given schedule equivalent to a serial execution of transactions?

Schedule: R, R, W, W, R, R, R, W, W

≡ ? ≡ ?

Serial schedule (T1, then T2):
R, W, R, W, R, W, R, R, W

Serial schedule (T2, then T1):
R, W, R, R, W, R, W, R, W

2) How do you come up with a schedule equivalent to a serial schedule?

Summary

- Transaction: a sequence of storage operations
- ACID:
  - Atomicity: all operations in a transaction happen, or none happens
  - Consistency: if database/storage starts consistent, it ends up consistent
  - Isolation: execution of one transaction is isolated from another
  - Durability: the results of a transaction persists
- **Serial schedule:** A schedule that does not interleave the operations of different transactions
  - Transactions run serially (one at a time)