Transaction Processing

Enterprise Scale Data Management

Divy Agrawal
Department of Computer Science
University of California at Santa Barbara

The Transaction Concept

2

- Multiple online users:
  - Gives rise to the **concurrency problem.**
- Component unreliability:
  - Gives rise to the **failure problem.**
- Database designers confronted these problems in the context of managing persistent data:
  - Online transaction processing system
  - Design, implementation, and operation of large application system with hundreds of terminals, tens of computers, providing service with no downtime, guaranteeing application correctness and data consistency.

SPRING'2011:CMPSC 274 4/4/11

The Transaction Concept

3

- Transactions provide an integrative framework in the presence of many "moving parts".
- Distributed transaction-oriented systems are the enabling technology:
  - Distributed and Networked applications
  - E-commerce and Workflow systems
  - Large-scale Information Infrastructures
- Without transactions, distributed systems/ networked applications cannot be made to work.

SPRING'2011:CMPSC 274 4/4/11

The Transaction Concept

- Transactions were originally developed in the context of DBMS as a paradigm to deal with:
  - Concurrent access to shared data
  - Failures of different kinds/types.
- Typical and canonical application scenarios in the context of banking application: Debit/ Credit operations, and fund Transfers.
- The key problem solved in an elegant manner:
  - Subtle and difficult issue of keeping data consistent in the presence of concurrency and failures

while ensuring performance, reliability, and availability.

4

SPRING'2011:CMPSC 274 4/4/11

OLTP Example: Debit/Credit

```
void main ( ) {
  EXEC SQL BEGIN DECLARE SECTION
    int BAL, AID, amount;
  EXEC SQL END DECLARE SECTION;

  scanf (“%d %d”, &AID, &amount); // USER INPUT

  EXEC SQL Select Balance into :BAL From Account
  Where Account_Id = :AID; // READ FROM DB

  BAL = BAL + amount; // update BALANCE

  EXEC SQL Update Account
    Set Balance = :b Where Account_Id = :AID; // WRITE TO DB
  EXEC SQL Commit Work;
}
```

SPRING'2011:CMPSC 274 4/4/11

Concurrent Executions: Lost Update Anomaly

DEBIT($50)	Time	CREDIT($100)
Select Balance Into :b_1 From Account Where Account_Id = :a	**1**	
	/* b_1=100, a.Balance=100, b_2=0 */	
	2	**Select Balance Into :b_2 From Account Where Account_Id = :a**
	/* b_1=100, a.Balance=100, b_2=100 */	
b1 = b1-50	**3**	
	/* b_1=50, a.Balance=100, b_2=100 */	
	4	**b_2 = b_2 +100**
	/* b_1=50, a.Balance=100, b_2=200 */	
Update Account Set Balance = :b_1 Where Account_Id = :a	**5**	
	/* b_1=50, a.Balance=50, b_2=200 */	
	6	**Update Account Set Balance = :b_2 Where Account_Id = :a**
	/* b_1=50, a.Balance=200, b_2=200 */	

Observation: concurrency or parallelism may cause inconsistencies, requires concurrency control for “isolation”

SPRING'2011:CMPSC 274 4/4/11

Funds Transfer: Inconsistent DATA

```
void main ( ) {
  /* read user input */
  scanf ("%d %d %d", &srcid, &tgtid, &amount);
  /* subtract amount from source account */
  EXEC SQL Update Account
    Set Balance = Balance - :amount Where AccId = :srceid;
  /* add amount to target account */
  EXEC SQL Update Account
    Set Balance = Balance + :amount Where AccId = :tgtid;
  EXEC SQL Commit Work; }
```

CRASH

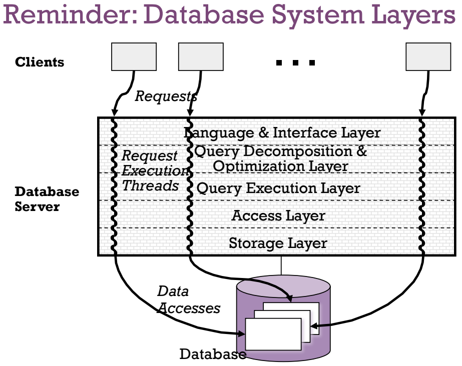
Observation: failures may cause inconsistencies, require recovery for "atomicity" and "durability"

Database Correctness

Database Fundamentals

Divy Agrawal
Department of Computer Science
UC Santa Barbara

Reminder: Database System Layers



Basic Ingredients

10

- Elementary Operations (read and write)
- Transactions (i.e., transaction program executions)
- Execution histories
- Characterization of correct executions
- Protocols (i.e., online algorithms to ensure correctness)

Transaction Page Model: Syntax

Page Model of Transaction:

A **transaction** T is a partial order of steps (actions) of the form r[x] or w[x], where $x \in D$ and reads and writes as well as multiple writes applied to the same object are ordered.
We write T = (op, <) for transaction T with step set op and partial order <.

Example: r[x] w[x] r[y] w[y]

Transaction Page Model: Semantics

Interpretation of j^{th} step, p_j, of T:

If p_j=r[x], then interpretation is assignment v_j := x to local variable v_j

If p_j=w[x] then interpretation is assignment x := f_j (v_{j1}, ..., v_{jk}) with unknown function f_j and j_1, ..., j_k denoting T's prior read steps.

Lost Update Problem

P1	Time	P2
	/* x = 100 */	
r (x)	1	
	2	r (x)
x := x+100	3	x := x+200
w (x)	4	
	/* x = 200 */	
	5	w (x)
	/* x = 300 */	

update "lost"

Observation: problem is the interleaving $r_1(x)$ $r_2(x)$ $w_1(x)$ $w_2(x)$

SPRING'2011:CMPSC 274 4/4/11

Dirty Read Problem

P1	Time	P2
r (x)	1	
x := x + 100	2	
w (x)	3	
	4	r (x)
	5	x := x - 100
failure & rollback	6	
	7	w (x)

cannot rely on validity
of previously read data

Observation: transaction rollbacks could affect concurrent transactions

SPRING'2011:CMPSC 274 4/4/11

Correctness Requirements: ACID

15

- ATOMICITY:
  - All-or-none property of user programs
- CONSISTENCY
  - User program is a consistent unit of execution
- ISOLATION
  - User programs are isolated with the side-effects of other user programs
- DURABILITY:
  - Effects of user programs are persistent forever

SPRING'2011:CMPSC 274 4/4/11

Transactions Executions: History

16

- History:
  - Contains all operations from all transactions
  - Distinct termination for every transaction
  - Preserves the order of operations of all transactions
  - Termination is the final step
  - Conflicting operations are ordered

SPRING'2011:CMPSC 274 4/4/11

Notion of Transaction Histories

17

- Goal:
  - A technique/algorithm/scheduler that prevents incorrect or bad execution.
- Develop the notion of correctness – or characterize what does correct execution means.
- This characterization will be based on the histories of transaction execution:

$H \to$ Good

$H \to$ Bad

SPRING'2011:CMPSC 274 4/4/11

Transaction Executions: Histories

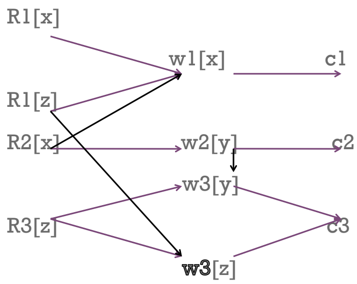
Let $T=\{T_1, \ldots, T_n\}$ be a set of transactions, where each $T_i \in T$ has the form $T_i=(op_i, <_i)$.

A **history** for T is $H=(op(H),<_H)$ such that:

1. $op(s) \subseteq \cup_{i=1..n}\, op_i \cup \cup_{i=1..n} \{a_i, c_i\}$
2. for all i, $1 \le i \le n$: $c_i \in op(s) \Leftrightarrow a_i \notin op(s)$
3. $\cup_{i=1..n} <_i \subseteq <_s$
4. for all i, $1 \le i \le n$, and all $p \in op_i$: $p <_H c_i$ or $p <_H a_i$
5. for all $p, q \in op(s)$ s.t. at least one of them is a write and both access the same data item: $p <_s q$ or $q <_s p$

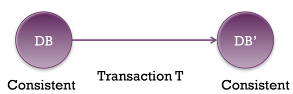
SPRING'2011:CMPSC 274 4/4/11

History Example



Correctness

- Syntactical semantics for schedules based on an intuitive notion:
  - Each transaction is a correct mapping, i.e.,



Hence, serial execution of transactions will be correct.

Serial History

- A history H is serial if for any two transactions T_i and T_j in H, all operations of T_i are ordered in H before all operations of $T_{j\ or}$ vice-versa.

General Idea

22

- Notion of equivalence of two histories H_1 and H_2.
- Use this notion of equivalence to accept all histories which are "equivalent" to some serial history as being correct.
- How to establish this equivalence notion?

Semantics

23

- Equivalence via a notion of semantics:
  - We do not know the semantics of transaction programs
  - We need a general notion that can capture all potential transaction semantics

➔ Need a general enough and powerful notion that can capture all possible semantics of transactions.