Transactions and Concurrency Control

Transactions

• Unit of update/change
  – Viewed as indivisible
  – Database can be inconsistent during transaction
    • Add to relations with mutual foreign keys
    • Constraints on values
      – Debit of bank savings + credit of bank checking
  – Commit transaction/ Abort transaction
    • Aborts by User
    • Aborts by Error
Concurrency

- Must be able to execute multiple transactions on DB together
  - Multiple users
    - Reservations, billing, banking, …
  - Long transactions
    - Reports, analysis, …
- Interleave transactions
- Each committed transaction must leave DB in consistent state
- Each aborted transaction must leave DB in state as if it never happened

Modeling transactions

- Only reads and writes to DB tables relevant
- Consider actions READ, WRITE, COMMIT, ABORT
- How interleave these actions correctly?
  - Actions of different transactions can interact
- Around these actions a transaction does local computation: not affect DB
- Next time: crash recovery
  - make sure DB in consistent state w.r.t transactions after crash
Example

Transaction T1: debit savings; credit checking
Transaction T2: get savings bal.; get checking bal.

T1: debit savings          credit cking
T2: bal chking?            Bal savings?
   time
   BAD

Transaction T1: debit savings; credit checking
Transaction T3: get savings bal.; get checking bal.

T1: debit savings          credit cking
T2: bal saving?            Bal chking?
   time
   GOOD

Read/Write diagrams

T1: R(V) W(V) R(K)W(K) C
T2: R(K) R(V) C X
T3: R(V) R(K) C √
T4: R(K)R(V)C X

R(object): read the DB object
W(object): write the DB object
C: transaction commits
V represents savings account
K represents checking account
Equivalence of schedules

Two schedule are equivalent if:
For any starting state of the DB for both schedules
The effect of executing the 1\textsuperscript{st} schedule is identical to the effect of executing the 2\textsuperscript{nd} schedule

Effect refers to the state of the DB as well as other results (e.g. a nasty letter that you are overdrawn)

Serializability

- **Serial schedule**: schedule for a set of transactions that does not interleave actions of different transactions
- A schedule is **serializable** if it is equivalent to some serial schedule for the same set of transactions
Conflict Serializable

- Conflicting actions by different transactions
  - Read and write to same DB object
  - Two writes to the same DB object
- Only non-conflicting actions
  - Two reads to the same DB object

A schedule is conflict serializable if the non-conflicting actions of the schedule can be reordered to get a serial schedule
- Strong condition!

Our Examples

T1: R(V) W(V) R(K)W(K) C
T2: R(K) R(V) C
T3: R(V) R(K) C

Graph:

Time

T1: R(V) W(V) R(K)W(K) C
T2: R(K) C
T3: R(V) R(K) C
Our Examples

T1: R(V) W(V)               R(K)W(K)               C
T2:                     R(K)                                 R(V)       C   X
T3:                      R(V)                                 R(K)       C   √

Precedence Graph

• Each node represents a transaction $T_i$
• Edge from $T_i$ to $T_j$ if some action of $T_i$ precedes and conflicts with an action of $T_j$

THEOREM: A schedule is conflict serializable if and only if the precedence graph for the schedule is acyclic
Locking

- Locks maintained by **transaction manager**
- Transaction **requests lock**
- Manager **grants/denies lock**
- Lock types:
  - **Shared:** need to have before **read** object
  - **Exclusive:** need to have before **write** object
- Object locked?
  - Different **levels granularity**
    - Tables and indexes
    - expense

Locking protocols

- **Strict 2-phase locking:**
  - Transaction requests lock at any time before action
  - Transaction **releases locks when commits**

- **2-phase locking** (not strict)
  - Transaction requests lock at any time before action
  - Transaction **releases locks at any time, BUT cannot request additional locks** once released **any** lock
    - Can release before commit but must have all locks ever need when release 1st
- **Strict 2-phase locking satisfies 2-phase locking constraints**
Theorem

• 2 phase locking (2PL) allows only schedule with acyclic precedents graph

=>

• 2 phase locking allows only conflict serializable schedules

• Corollary: Strict 2-phase locking allows only conflict serializable schedules