14 Transactions: models

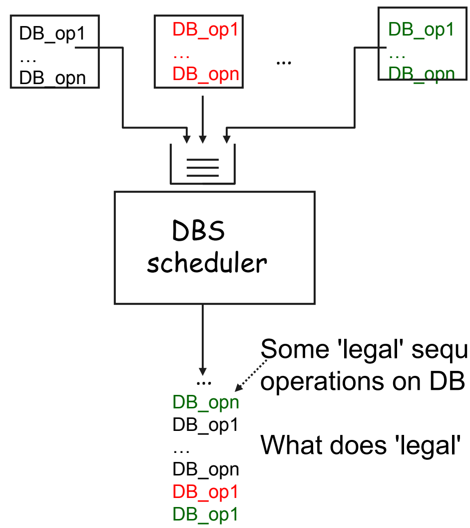
Kemper / Eickler chap 11.1-11.5, Elmasri/Navathe chap. 19

14.1 Concept: ACID properties

- A transaction is
  - A unit of work...
  - … which consists of a sequence of one or more operations...
  - is executed with the following guarantees::
    - **A**tomic: the sequence of operations is executed completely or it has no effect (on the database)
    - **C**onsistent: if the database was in a consistent state before transaction execution it will be after
    - **I**solated: concurrently executed transactions (TA's) do not interfere
    - **D**urable: (persistent): all effects of a TA are permanent

Transactions: DBS perspective

- System point of view



Some 'legal' sequence of operations on DB

What does 'legal' mean?

14.2 Modeling Transactions

– Main concern: concurrency.
  Model should enable the study of isolation properties

– Model should be most general - since nothing is known about the particular transaction programs

– Model should be independent of
  - particular actions in the TA programs
  - particular DB language
  - of the granularity of objects to be read / written

Note however: the scheduler could do the better, the more information it has – e.g. " t1 is a 'Read-only' – TA"

Modeling TAs

- Modeling TAs: The Read/Write model

  Atomic DB-operations of TA i are

  - $READ_i[x]$ - TA i reads Object x $r_i[x]$
  - $WRITE_i[y]$ - TA i writes Object x $w_i[x]$,
    ⇨ the DB state is changed
  - $Commit_i$ - TA i wants to terminate successfully
  - $Rollback_i$ - TA i wants to abort without leaving any effects in the DB

  – Operations of different TAs interleaved

* as an abstraction

The Model

– A transaction is a sequence of reads and writes, e.g.:

$TA_j = r_j[x], r_j[y], w_j[y], r_j[z], w_j[x], w_j[s], w_j[z], c_j$

– c_j means "successful commit ", a_j "abort TA_j",
may be sometimes omitted

– The sequence reflects the sequence (time and logic) of DB operations of a single transactional program, the subscript i of op_i identifies the transaction this operation belongs to.

– no TA reads or writes the same item twice
no TA reads an item it has written

$TA_j = r_j[x], w_j[x], r_j[z], r_j[x], w_j[x], c_j$

redundant — final effect

Transactions and transaction sets

Data dependencies: written data item dependent on all previous read items

$TA_j = r_j[x], r_j[y], w_j[y], r_j[z], w_j[x], w_j[s], w_j[z], c_j$

Interleaved transactions

TA1: $r_2[x], w_2[y], w_2[x], r_2[s]\ c_2$

TA2: $r_1[x], r_1[y], w_1[x], c_1$

"Blind writer"

One of many interleaved transaction executions

$r_1[x], r_2[x], w_2[y], r_1[y], w_1[x], w_2[x], r_2[s], c_1, c_2$

Correctness criteria

- Main concern: given a set of TAs
  What is a correct execution sequence of their atomic operations?
- Potential problems during interleaved execution
  - Lost update
  - Dirty read: read uncommitted data
  - Non-repeatable read: different result when reading the same object more than once in a transaction
  - Phantoms: a kind of non-repeatable read caused by insertions or deletions

Example: Correctness violation

Lost update

T

T1:r[x] , T2:r[x] ,T1: x=x+1, T1:w[x], T2:x=x-1, T2:w[x], T2:c, T1:c

Read not repeatable

T

T1:r[x], T2:r[x] ,T2: x=x+1 , T2:w[x], T1:r[x], T2:c, T1:c

Transactions Phantom

Phantoms

TA1

```
Exec sql select balance into :bbal from
branch_totals
        where branch_id = :bid
Exec sql select  sum(balance) into :total
from accounts
             where branch_id = :bid
If (bbal <> total)  {print "something
´seriously wrong")
```

TA2

```
Exec sql insert into accounts ... values
(.....);
```

Causes phantom,
if executed here

Transactions Correctness

14.2.1 Correctness criteria

– If transactions are scheduled in arbitrary sequential order e.g.
  TA1; TA2 or TA2; TA1 (for two TAs)
  ⇨ no resource conflicts
  ⇨ no concurrency issues

  if all resources are released after commit
  ⇨ no concurrency at all
  ⇨ nondeterministic state at the end of execution if order of execution is arbitrary

Transaction indeterminism

Example

TA1: r1[x], x==x+1, w1[x]
TA2: r2[x], x==x*10, w2[x]
State after executing TA1; TA2 : x_new ==(x_old +1)*10
State after executing TA2; TA1 : x_new ==x_old*10 +1

Serial Execution

An execution of transaction in an arbitrary sequential order is called a serial execution

T1 then T2:
r1[x], r1[y], w1[y], r1[z], w1[x], c1, r2[y], r2[z], w2[y],r2[x], w2[x], r2[s], c2

T2 then T1 :
r2[y], r2[z], w2[y], r2[x], w2[x], r2[s], c2, r1[x], r1[y], w1[y], r1[z], w1[x], c1
,
are both serial executions

Note: the order of operations within a transactions is unchanged

14.2.3 Transactions History

Wanted:

a more efficient interleaved execution sequence which guarantees a correct final database state

History (schedule, execution sequence)

Informally an interleaved sequence of atomic actions of two or more transactions

Find histories which guarantee a correct final state

History

A history S of a (finite) set of transactions T
is a sequence <a> of atomic actions a if the following conditions hold:
(1) An atomic action of a TA $\in$ T occurs exactly once in S
(2) No other action occurs in S
(3) If a < a' in some TA, then a < a' in S (*)
where "<" is the canonical ordering induced by the sequence of operations in TA and S rsp.

A schedule is a prefix of S.

(*) Does a more general approach make sense?

History

Example

TA $_1$ = r1[x], r1[y], w1[y], r1[z], w1[x], c_1
TA $_2$ = r2[y], r2[z], w2[y], r2[x], w2[x], r2[s], c_2

r1[x], r2[y], r2[z], w2[y], r2[x], r1[y], w2[x], w1[y], r1[z], r2[s], c2, w1[x], c1
is a history (schedule) of TA_1, TA_2 (see above)

r1[x], r2[y], r1[y], w2[y], w1[y], r1[z], r2[z], r2[x], w2[x], c2, w1[x], c1

is not, why?

- Obvious: every serial execution is a history
- Goal: find correct schedules
- what is "correct"?

14.3 Serializability

- Correctness criterion for schedules: Serializability

> Informal: A history (schedule) S of the transaction set T is called serializable, if its effects are equivalent to a serial execution of T
>
> Serial schedules: correct by definition

- What does "equivalent" mean?

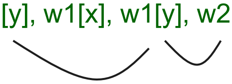
  Same DB state at the very end?   Indeterminism!

  Plausible but not effective

Serializability

- Well known from concurrency theory:
  - No conflict if only concurrent READs
  - lost update, dirty read etc. can only occur, if different transactions operate on the same object and at least one is a write operation
- Analyze conflicting operations of different TAs

H: r1[x], r2[y], w1[x], w1[y], w2[y]



Conflict pairs

Intuitive equivalence of schedules: same order of conflicts

In the example: first conflict: TA2 < TA1, second TA1 < TA2

Serializability

14.3.1 Conflict Relation

$op_i[x]$ and $op'_j[y]$ are in conflict, if

x = y AND

i != j AND

op = w $\vee$ op' = w

> Conflict relation of a schedule S:
>
> C(S) = {(op,op') | op and op' are conflicting and op < op' in S}

Example:

TA 1 = r1[x], r1[y], w1[y], r1[t], w1[x], c1

TA 2 = r2[y], r2[z], w2[z], r2[x], w2[x], r2[s], c2

C(S) = {(r2[y], w1[y]), (r1[x],w2[x]) (w1[x], w2[x]), (r2[x], w1[x])}

> What is S?? can you find an S with this C(S)?

Serializability

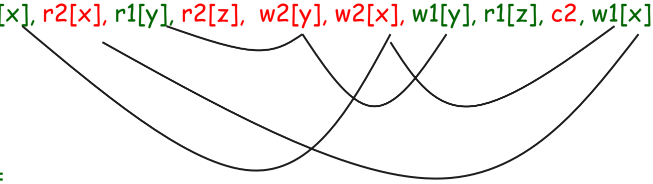
- Conflict serializable schedules
  - Basic idea: correct schedules should have the same conflict pairs as *some* serial schedule
  - Means: if there are conflicting pairs of operations in TA1 and TA2 they should be executed in the same order: TA1 – ops before TA2 – ops or the other way round

> A Schedule S of a transaction set T is conflict serializable (or serializable),
>
> if it has the same conflict relation as some serial execution SER of T: C(S) = C(SER)

Serializability

- Example

S: r1[x], r2[x], r1[y], r2[z], w2[y], w2[x], w1[y], r1[z], c2, w1[x], c_1



C(S) =
{ (r1[x], w2[x]), (r2[x], w1[x]), (r1[y], w2[y]) (w2[y], w1[y]), (w2[x], w1[x]) }

T1 before
T2 in a serial Schedule

T2 before
T1 in serial Schedule

NOT
conflict
serializable

Serializability

Conflict Graph (Precedence | dependency graph)

- Conflicts graph:

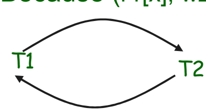
  (a) Nodes: Transactions {T1, ….Tn}

  (b) Directed Edges E :

  (Tj,Tk) $\in$ E :$\Leftrightarrow$
  exists a conflicting pair (op$_j$ [x] , op'$_k$ [x])

Example:

Because (r1[x], w2[x]),



Because (r2[x], w1[x]),

Conflict graph and serializability

- Conflict graph   CG(S)
  - Represents the conflict relations between transactions
  - Note: Commit does not have an influence on the graph
    Therefore commit – operation c, may be omitted.
    Why exactly?
  - How does the conflict graph of a serializable schedule look like?

Serializability

14.2.2 Serializability Theorem:

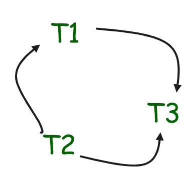
> A schedule S is conflict serializable, if and only if its conflict graph does not contain a cycle

Example:

S: r1[y], r3[u], r2[y], w1[y], w2[x], w1[x], w2[z], w3[x]

Intuitive correctness idea:
Determine from graph the "conflict-equivalent" serial schedule. (Example: T2 before T1 and before T3, T1 before T3, therefore T2,T1,T3)

Exchange operations in S without switching conflict pairs until serial schedule is established



Serializable

Serializability

- Proof of Serializability Theorem:

  " ⇐" " no cycle --> serializable"

  The nodes of a connected directed graph without cycles can be sorted topologically: a < b iff there is a path from a to b in the graph. Results in a serial schedule TAi, ...TAk if non-conflicting TAs are added arbitrarily.

  " ⇒ " "Serializable → no cycle"

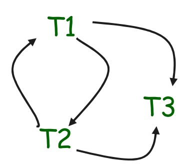
  Suppose there is a cycle TA i -> TAj in CG(S) .
  Then there are conflicting pairs (p,q) and (q',p'), p,p' from TAi, q,q' from TAj. No serial schedule will contain both (p,q) and (q',p').
  Induction over length of cycle proves the "only if"

Transactions Serializability

- Conflict serializability is restrictive

  S1: w1[y], w2[x], r2[y], w2[y], w1[x], w3[x]

  C(S1) = {(w1[y],r2[y]), (w1[y],w2[y]), (w2[x], w1[x]), (w2[x],w3[x]),
  (w1[x], w3[x])}



- But effect is the same as from the serial Schedule: T1, T2, T3
  since T3 is a "blind writer":
  writes x independent of previous state

Summary of the TA model

- Summary (serializability theory)
  - Serial executions of a fixed set of transactions T trivially have isolation properties
  - Schedules of T with the same effects as an (arbitrary) serial execution are intuitively correct
  - If all conflicting pairs of atomic operations are executed in the same order in some schedule S' as in the schedule S, the effects of S and S' would be the same
  - Conflict graph is a simple criterion to check conflict serializability
  - Conflict serializability is more restrictive than necessary (see view serializability -> literature)
  - Serializability is a theoretical model which defines correctness of executions.