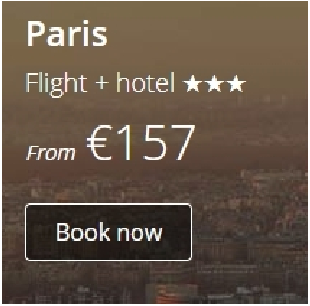
Transaction Processing
Concurrency

Hans Philippi

November 25, 2025

Transactions

Transactions are about a group of database operations (reads, inserts, deletes, updates) that should be regarded as one logical unit and one atomic action:



Transactions

Applications often require the possibility to execute transactions in a concurrent mode:

> The Seattle-based company said customers ordered 34.4 million items during Prime Day, or to put it another way, 398 items per second. In the U.S. alone it sold 47,000 TVs, marking a 1300 percent increase on last year; 41,000 Bose headphones, compared to eight the previous Wednesday; and 14,000 iRobot Roomba 595 Pet Vacuum Cleaning Robots, compared to one the

Uncontrolled concurrency may lead to incorrect executions

A concurrent ATM teller transaction on the same account x

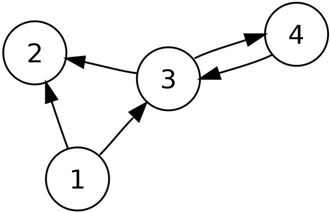
T1	T2
Read(x)	
x -= 500	
	Read(x)
	x -= 500
Write(x)	
	Write(x)

Transaction concept

We have a problem with concurrency, so ...

- We will start formalizing transaction executions
- We will define a formal notion of correct concurrent executions
- We will propose methods that ensure correct concurrent executions
- ...
- We will need the notion of a *directed graph*

Directed graph



This (directed) *graph* G is defined by

- a set of *nodes* $N = \{1, 2, 3, 4\}$
- a set of *edges* $E = \{(1, 2), (1, 3), (3, 2), (3, 4), (4, 3)\}$

By the way, it is *cyclic*

Graphs (directed and undirected) arguably are the most frequently applied discrete mathematical structure in computing science

Transaction concept

Definition:

Atomic database action

$R(x)$: read data element x

$W(x)$: write data element x

data element: attribute, tuple, block, table

Definition:

A transaction is a collection of atomic database actions considered to be one logical unit, with respect to:

- concurrency
- recovery

Transaction concept

Termination of a transaction

- *commit*: positive, complete, irreversible, persistent;
  effects to real world will become visible
- *abort* or *rollback*: negative, nihil;
  no effects to real world;
  no partial results visible

ACID-properties:

A tomicity

C onsistency preservation

I solation

D urability

Transaction concept: ACID-properties

- Atomicity: a transaction runs either completely (commit) or not at all (abort)
- Consistency preservation: a transaction must respect the integrity constraints; if not, it must be aborted
- Isolation: each transaction must run without visible interference with other transactions
- Durability: after commitment, the persistency of the updated data must be guaranteed

Schedules

The (concurrent) execution order of a sequence of database operations generated by a set of transactions is represented by a *schedule* or *history*

Example:
an interleaved schedule $S1$ and a serial schedule $S2$

S1

T1	T2	T3
	W(y)	
		R(x)
W(x)		
	W(z)	
		R(y)

↓ time

S2

T1	T2	T3
	W(y)	
	W(z)	
		R(x)
		R(y)
W(x)		

Problem 1: lost update

S3

T1	T2
R(x)	
	R(x)
W(x)	
	W(x)

Inconsistent retrieval

Problem 2: inconsistent retrieval

S4

T1	T2
	$R(x_1)$
	$R(x_2)$
	$\vdots$
	$R(x_m)$
$R(x_m)$	
$W(x_m)$	
$R(x_{m+1})$	
$W(x_{m+1})$	
	$R(x_{m+1})$
	$\vdots$
	$R(x_n)$

Correctness criterion

Formalizing correctness

- serial schedules (schedules without concurrency) are correct
- two schedules are equivalent if
  their effects visible to the outside world
  are exactly the same
- an interleaved schedule is correct if
  it is equivalent to a serial schedule;
  such a schedule is called <u>serializable</u>

Correctness criterion

- *Intuition:* two atomic database actions are conflicting if the effects of reversing the relative order of their execution are visible to the outside world
- *Definition:* two atomic database actions are <u>conflicting</u> if they affect the same data element and at least one of the operations is a write
- *Definition:* two schedules are <u>equivalent</u> if they order all the conflicts the same way

Correctness criterion

S5

T1	T2	T3
		W(x)
R(x)		
		R(z)
	W(x)	
W(y)		
	W(y)	

S6

T1	T2	T3
		W(x)
		R(z)
R(x)		
W(y)		
	W(x)	
	W(y)	

Correctness criterion

S5

T1	T2	T3
		W(x)
R(x)		
		R(z)
	W(x)	
W(y)		
	W(y)	

S6

T1	T2	T3
		W(x)
		R(z)
R(x)		
W(y)		
	W(x)	
	W(y)	

S5 orders all the conflicts the same way S6 does
... so S5 is equivalent to S6 ...
... and S6 is serial ...
... so S5 is correct: *serializable*

INTERMEZZO

S7

T1	T2	T3	T4
		W(x)	
	R(x)		
W(z)			
			R(z)
			W(y)
		W(y)	
	R(y)		
W(y)			
	W(x)		

S8

T1	T2	T3	T4
		W(x)	
	R(x)		
			R(z)
			W(y)
W(z)			
		W(y)	
	R(y)		
W(y)			
	W(x)		

is *S7* serializable ?

is *S8* serializable ?

Testing serializability

So, testing serializability of a schedule S means

considering ... serial schedules of n transactions?

Testing serializability

Definition: a precedence graph $G(S)$ is a directed graph connected to a schedule S such that:

- the set of nodes is the set of transactions in S
- there exists an edge $T_i \rightarrow T_j$ in G if
  there is a conflicting pair of operations o_i, o_j in S
  such that o_i occurs before o_j

Testing serializability

Theorem:
a schedule S is serializable $\Leftrightarrow$ $G(S)$ is acyclic

Enforcing serializability: 2PL

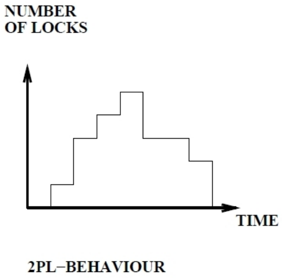
Two phase locking (2PL)

Scheduling technique based on locking principle

- before executing an atomic operation, the requesting transaction has to acquire an exclusive lock on the data element; if the data element is locked by another transaction, the requesting transaction has to wait until release of the lock by the other transaction
- after executing the operation, the scheduler has to release the lock

Enforcing serializability: 2PL

- the transaction is not allowed to acquire new locks after the release of a lock; so locking is executed in two consecutive phases: the growing phase and the shrinking phase (with respect to the number of locks held by the transaction)



Enforcing serializability: 2PL

Theorem:
S is a 2PL-schedule $\Rightarrow$ S is serializable

Enforcing serializability: 2PL

Refinement

- concurrent execution of read-operations should be allowed
- exclusiveness of locks is only required in case of conflicting operations
- we distinguish read-locks and write-locks (or exclusive locks)

Lock compatibility matrix

	read	write
read	yes	no
write	no	no

Note that 2PL behaviour is enforced on *every* transaction separately!

Enforcing serializability: 2PL

Problem: deadlock

Solution:

- prevention
- detection by time out
- detection by wait-for-graphs

Enforcing serializability: timestamping

Alternative to 2PL

- Every transaction receives a timestamp at birth
- Timestamps are emitted in some order (typically a counter or system time)
- Conflicting operations are executed in timestamp order
- Application: ad hoc concurrency control in loosely coupled databases

Enforcing serializability: timestamp ordering principle

Timestamp rule:

For each conflicting pair of operations o_i, o_j in S,
if $o_i < o_j$ then $ts(T_i) < ts(T_j)$

where $o_i < o_j$ denotes time order

Implementation:
timestamp manager maintains a table of recently used data entries

Enforcing serializability: timestamping

Theorem:

If schedule S obeys the timestamp rule
then S is serializable

Enforcing serializability: other methods

Optimistic concurrency control by a posteriori validation

no further details