Database Management Concepts II

Instructor: Dmitri A. Gusev

Fall 2007

CS 502: Computers and Communications Technology

Lecture 22, November 26, 2007

Data integrity

Integrity constraints: semantic conditions on the data

- Individual constraints on data items
- Uniqueness of the primary keys
- Dependencies between relations

Concurrency control

- Steps in executing a query
- Concurrent users of the database, interfering with the execution of one query by another
- Transaction: a set of operations that takes the database from one consistent state to another
- Solving the concurrency control problem: making transactions atomic operations (one at a time)
- Concurrent transactions: serializability theory (two-phase locking), read lock (many), write lock (one).
- Serializible transactions: first phase - accumulating locks, second phase - releasing locks.
- Deadlocks: deadlock detection algorithms.
- Distributed execution problems:
    - release a lock at one node (all locks accumulated at the other node?)
    - strict two-phase locking

The Transaction Model

Primitive	Description
BEGIN_TRANSACTION	Make the start of a transaction
END_TRANSACTION	Terminate the transaction and try to commit
ABORT_TRANSACTION	Kill the transaction and restore the old values
READ	Read data from a file, a table, or otherwise
WRITE	Write data to a file, a table, or otherwise

- Examples of primitives for transactions.

The Transaction Model

```
BEGIN_TRANSACTION
  reserve WP -> JFK;
  reserve JFK -> Nairobi;
  reserve Nairobi -> Malindi;
END_TRANSACTION
```

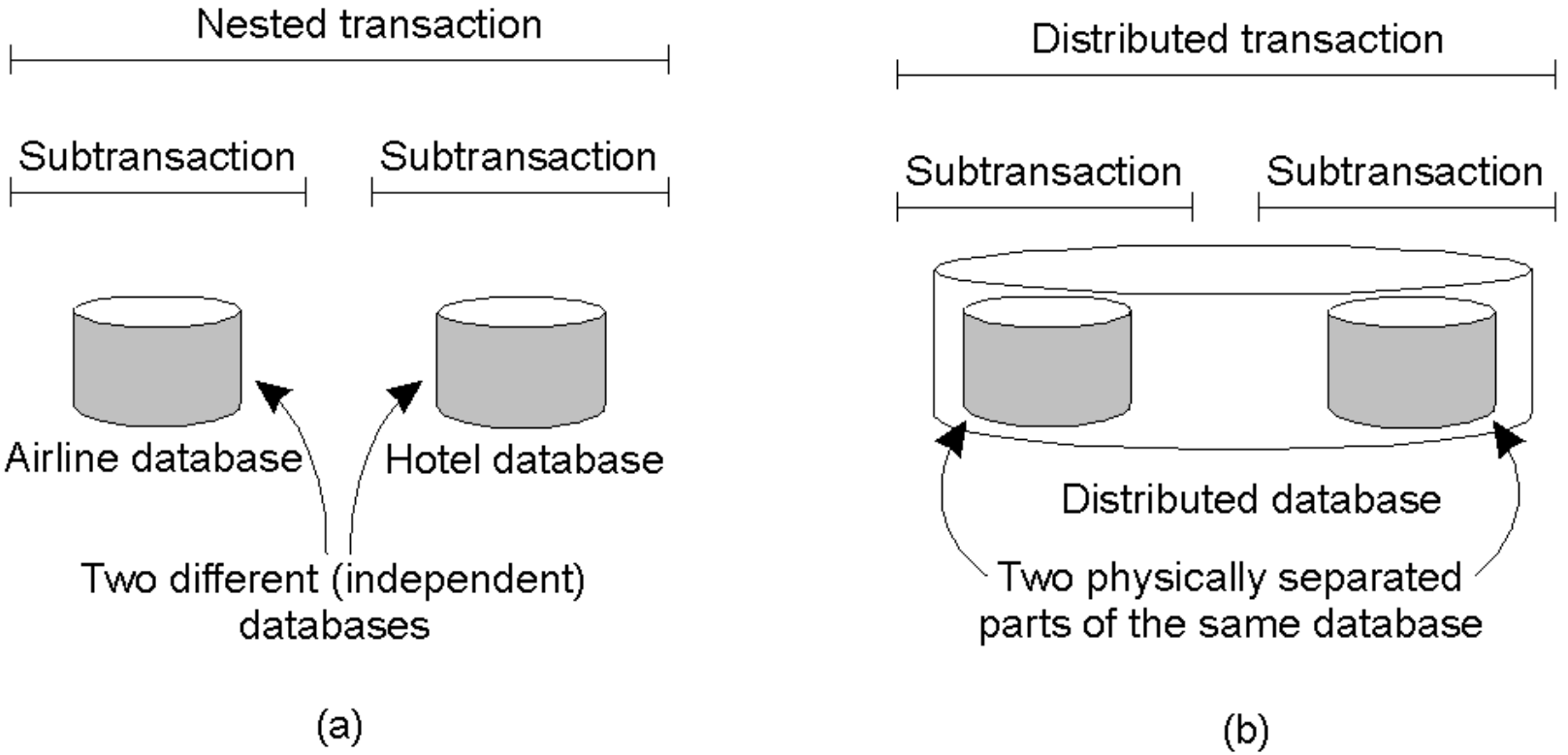
(a)

```
BEGIN_TRANSACTION
  reserve WP -> JFK;
  reserve JFK -> Nairobi;
  reserve Nairobi -> Malindi full =>
ABORT_TRANSACTION
```

(b)

a) Transaction to reserve three flights commits
b) Transaction aborts when third flight is unavailable

Distributed Transactions



a) A nested transaction
b) A distributed transaction

ACID

- *ACID* (Atomicity, Consistency, Isolation, Durability) is a set of properties that guarantee that database transactions are processed reliably.
  - *Atomicity* refers to the ability of the DBMS to guarantee that either all of the tasks of a transaction are performed or none of them are.
  - *Consistency* refers to the database being in a legal state when the transaction begins and when it ends. This means that a transaction cannot break the rules, or *integrity constraints*, of the database.
  - *Isolation* means that no operation outside the transaction can ever see the data in an intermediate state. More formally, isolation means the transaction history (or schedule) is *serializable*.
  - *Durability* refers to the guarantee that once the user has been notified of success, the transaction will persist, and not be undone. This means it will survive system failure, and that the database system has checked the integrity constraints and won't need to abort the transaction. Many databases implement durability by writing all transactions into a log that can be played back to recreate the system state right before the failure. A transaction can only be deemed committed after it is safely in the log.

Write Ahead Logging (WAL)

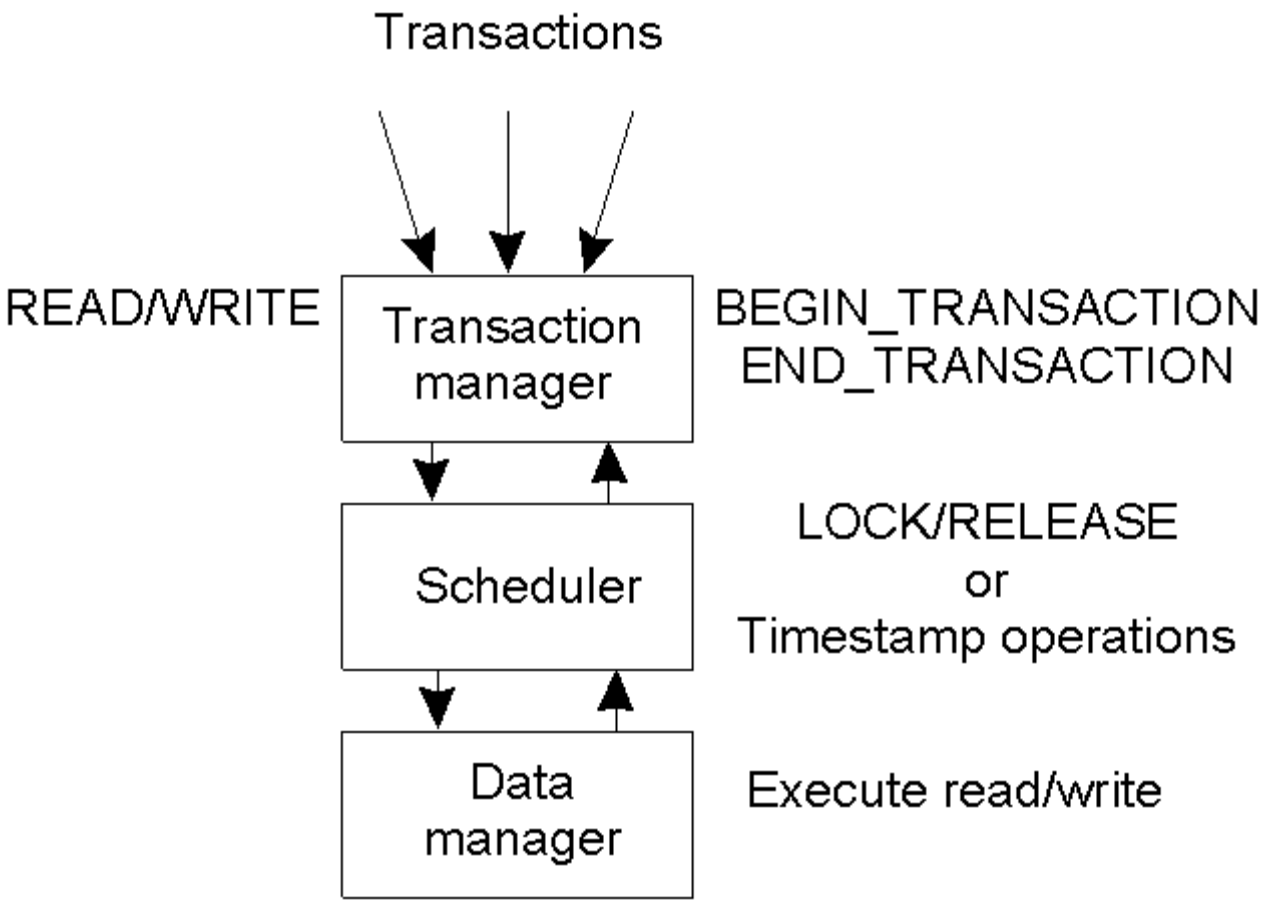
- *Write Ahead Logging (WAL)* is a family of techniques for providing atomicity and durability (two of the ACID properties) in database systems. In a system using WAL, all modifications are written to a log before they are applied to the database. Usually both redo and undo information is stored in the log.The motivation for WAL is to allow updates of the database to be done *in-place*.

Write Ahead Log

x = 0;	Log	Log	Log
y = 0;			
BEGIN_TRANSACTION;			
x = x + 1;	[x = 0 / 1]	[x = 0 / 1]	[x = 0 / 1]
y = y + 2		[y = 0/2]	[y = 0/2]
x = y * y;			[x = 1/4]
END_TRANSACTION;			
(a)	(b)	(c)	(d)

- a) A transaction
- b) – d) The log before each statement is executed

Concurrency Control



- General organization of managers for handling transactions.

Serializability

- A schedule (transaction history) is *serializable* if its outcome (the resulting database state) is equal to the outcome of its transactions executed sequentially without overlapping.

Serializability (example)

```
BEGIN_TRANSACTION
  x = 0;
  x = x + 1;
END_TRANSACTION
```

(a)

```
BEGIN_TRANSACTION
  x = 0;
  x = x + 2;
END_TRANSACTION
```

(b)

```
BEGIN_TRANSACTION
  x = 0;
  x = x + 3;
END_TRANSACTION
```

(c)

Schedule 1	x = 0; x = x + 1; x = 0; x = x + 2; x = 0; x = x + 3	Legal
Schedule 2	x = 0; x = 0; x = x + 1; x = x + 2; x = 0; x = x + 3;	Legal
Schedule 3	x = 0; x = 0; x = x + 1; x = 0; x = x + 2; x = x + 3;	Illegal

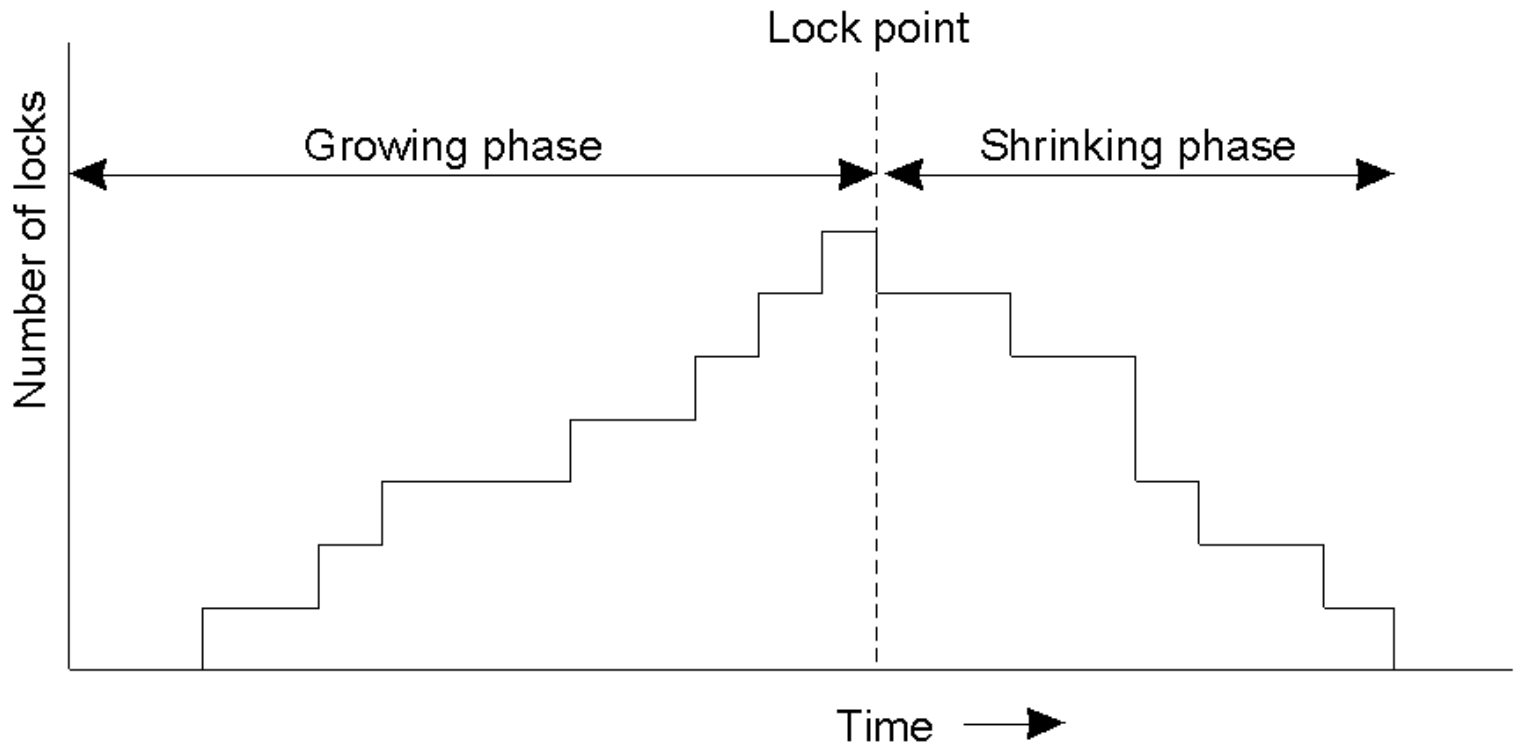
(d)

- a) – c) Three transactions T_1, T_2, and T_3
- d) Possible schedules

Two-Phase Locking

- According to the *Two phase locking* protocol, locks are handled by a transaction in two distinct, consecutive phases during the transaction's execution:
  - Phase 1: Locks are acquired and no locks are released.
  - Phase 2: Locks are released and no locks are acquired.
- The *Strict two phase locking* (S2PL) class of schedules is the intersection of the 2PL class with the class of schedules possessing the Strictness property. To comply with the S2PL protocol a transaction needs to comply with 2PL, and release its *write (exclusive) locks* only after it has ended, i.e., being either committed or aborted.

Two-Phase Locking



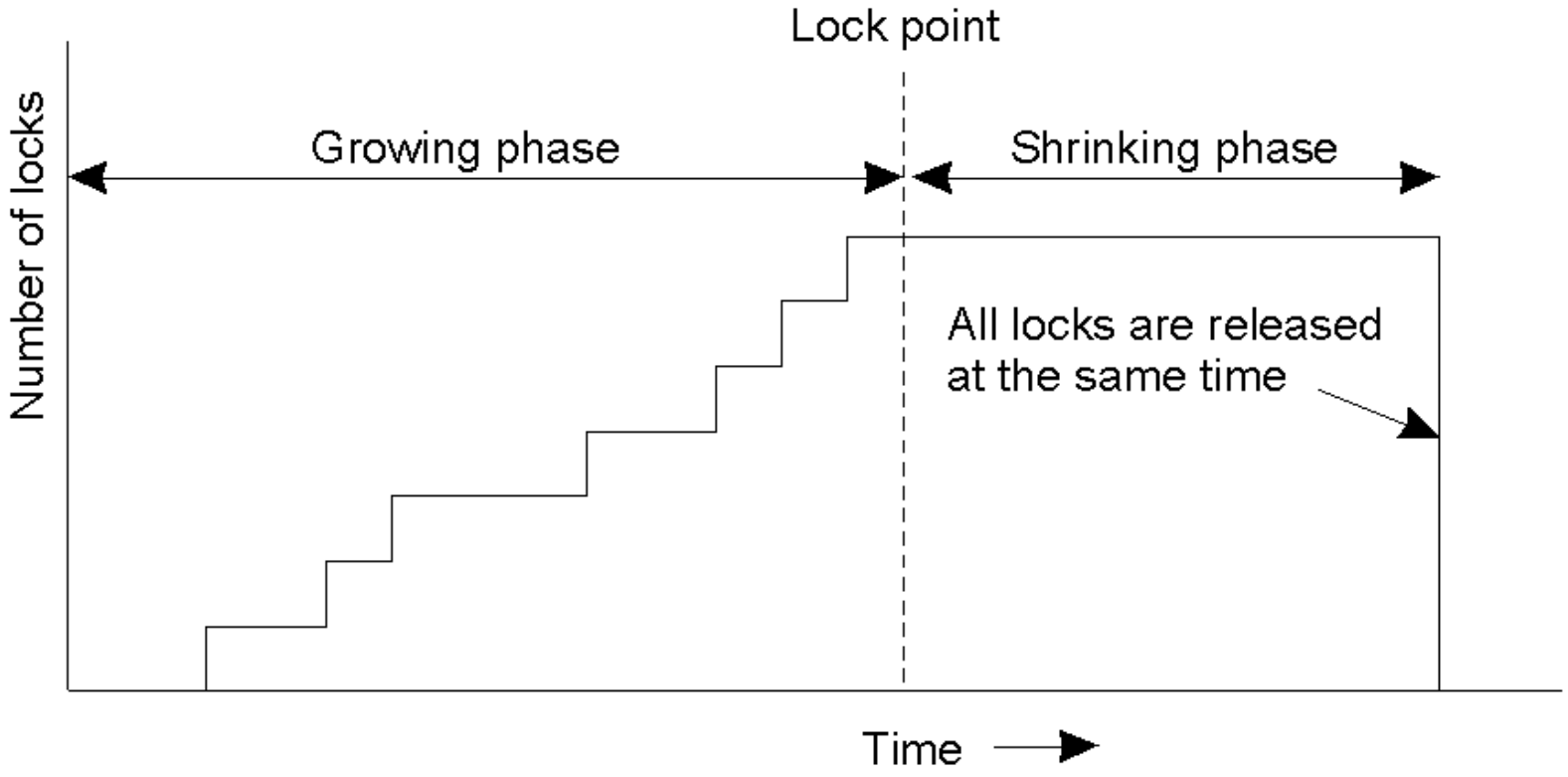
- Two-phase locking.

SS2PL

- *Strong strict two phase locking (SS2PL)* is a common mechanism utilized in database systems to enforce both conflict serializability and strictness of a schedule. SS2PL is the name of the resulting schedule property as well, which is also called *rigorousness*. In this mechanism each data item is locked by a transaction before accessing it (any read or write operation). As a result, access by another transaction may be blocked, typically upon conflict, depending on lock type and the other transaction's access operation type. **All** locked data on behalf of a transaction (i.e., both its *write (exclusive)* and *read (shared)* locks) are released only after the transaction has ended (either committed or aborted).

- Mutual blocking between transactions results in a *deadlock*, where execution of these transactions is stalled, and no completion can be reached. A deadlock is a reflection of a potential cycle in the conflict graph, that would occur without the blocking. Deadlocks are resolved by aborting a transaction involved with such potential cycle. It is often detected using a *wait-for graph* that indicates which transaction is "waiting for" a lock release by which transaction, and a cycle means a deadlock. Aborting one transaction per cycle is sufficient to break the cycle.

SS2PL (cont'd)



- Strong strict two-phase locking.

Data integrity

Backup and recovery

- The problem of keeping a transaction atomic: successful or failed

  What if some of the intermediate steps failed?
- Log of database activity: use the log to undo a failed transaction.
- More problems: when to write the log, failure of the recovery system executing the log.

Security and access control

- Access rules for relations or attributes. Stored in a special relation (part of the data dictionary).
- Content-independent and content-dependent access control
- Content-dependent control: access to a view only or query modification

  (e.g. and-ing a predicate to the WHERE clause)
- Discretionary and mandatory access control

Knowledge Bases and KBS (and area of AI)

- Information, Data, Knowledge (data in a form that allows reasoning)
- Basic components of a KBS
    - Knowledge base
    - Inference (reasoning) mechanism (e.g. forward/backward chaining)
    - Explanation mechanism/Interface
- Rule-based systems (medical diagnostics, credit evaluation etc.)

Rule Base:

1. IF (lecturing X)
   AND (grading-tests X)
   THEN (overworked X)

2. IF (month february)
   THEN (lecturing alison)

3. IF (month february)
   THEN (grading-tests alison)

4. IF (overworked X)
   THEN (bad-mood X)

5. IF (slept-badly X)
   THEN (bad-mood X)

6. IF (month february)
   THEN (weather cold)

7. IF (year 1993)
   THEN (economy bad)

Backward chaining:

Given the facts:

(month february)
(year 1993)

we can prove

(bad-mood alison)

Forward chaining:

Given the facts:

(month february)
(year 1993)

we can infer

(lecturing alison)
(grading-tests alison)
(overworked alison)
(bad-mood alison)
(economy bad)

Backward chaining by asking questions:

Given no facts we can try to prove
(bad-mood alison)

Then the system asks:
Is (month february) true?

If the answer is "yes", then the system proves successfully answers yes the initial question.
If the answer is "no" the system tries to find an alternative prove and asks:

Is (slept-badly alison) true ?