

6 TRANSACTION MANAGEMENT AND CONCURRENCY CONTROL

6.1 Transactions

- A **transaction** is a sequence of operations such that the database is in a consistent (or correct) state both before and after the transaction, but may be in an inconsistent state during the transaction.
- **Example:** Transfer \$100 from account A to account B.

```
Begin transaction
  Read A
  A.amount := A.amount - 100
  Write A
  Read B
  B.amount := B.amount + 100
  Write B
End transaction
```

- Required properties:

- **Atomicity**

Either all the operations of a transaction are performed or none are.

- **Durability**

All (committed) changes to the database persist - no matter what.

- Typical transaction structure:

```
Begin transaction
  Read input message
  Perform processing against the database
  If successfully completed
    then send output message(s) and COMMIT
    else ROLLBACK and send error message
  EndIf
End transaction
```

- **COMMIT**

Signals successful completion of a transaction to the DBMS, which frees all locks, if any, and makes all changes permanent and visible to other users.

– **ROLLBACK**

Signals unsuccessful completion of a transaction to the DMBS, which **undoes** all changes made by the transaction.

If a transaction is aborted by the system, an implicit rollback must be issued on its behalf.

- The transaction management subsystem of a DBMS normally consists of

– **Concurrency control manager**

Keeps track of active transactions, schedules the reading and writing of records so that transactions do not interfere with each other, and coordinates COMMITs and ROLLBACKs.

– **Recovery manager**

Maintains a log of changes made to the database, performs transaction REDO and UNDO, and restores the database to a consistent state after a system crash.

– In addition, there is a **buffer manager**.

- * It performs all physical reads and writes, and maintains the page buffer (page cache). The purpose of the page buffer is to reduce the number of physical reads and writes by keeping heavily used pages in core.
- * A logical (record) write does not necessarily result in an immediate physical write. The physical block affected by the write may stay in the page buffer, and the physical write may occur much later.
- * A logical read does not require a physical read if the required page is already in the page buffer.

6.2 Cocurrency Control

- Concurrently executed transactions may interfere with each other, producing an incorrect overall result, even if each transaction is correct when executed in isolation.

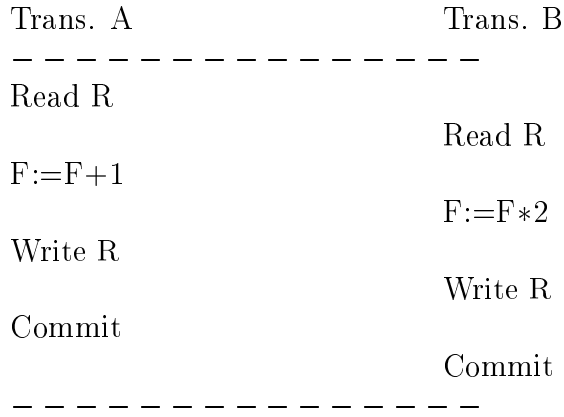
- **Example:** The lost update problem.

Transaction A: Add 1 to field F of record R

Transaction B: Double the value of field F of R

Initial value: R.F=2

Interleaved execution:



Final result: R.F = 4 (wrong)

- Correctness criterion:

An interleaved execution of a set of transactions is considered correct if it produces the same final result as some serial execution of the same set of transactions. Such an execution is said to be serializable.

- A **schedule** for a set of transactions is an ordering of the read and write operations of the transactions (retaining the relative ordering within each transaction).
- A schedule S is serializable if there exists a serial schedule S' equivalent to S .
- Testing for serializability

We assume that a transaction cannot write a data item without first reading it (read-before-write assumption).

Serializability can be tested by building a directed graph, called the **precedence graph**. There is an edge from T_i to T_j if there exists a data item R for which

- T_i writes R before T_j reads R, or

- T_i reads R before T_j writes R.

- **Theorem:** A schedule S is serializable if and only if its precedence graph contains no cycles.

- There are concurrency control schemes based on

- Locking

- Versioning

- **Exclusive locks**

- A transaction can lock a data object by issuing a request to a Lock Manager. We assume that lock request are combined with read and write requests.

- **Definition** (exclusive lock, X-lock)

If a transaction T holds an exclusive lock on some object O then no other transaction T' can acquire a lock (of any type) on O, nor access O.

- The object to be locked may be the entire database, a table, a page, a record, or an attribute value.

- Protocol PXC

1. Any transaction intending to update an object O must first acquire an X-lock on O (by means of an XREAD O). The transaction will be forced to wait until the lock can be granted.

2. All exclusive locks are retained until the end of the transaction (COMMIT or ROLLBACK).

- Rule (2) above is necessary to prevent transactions from becoming dependent on uncommitted updates (prevent dirty reads).

- **Shared locks**

- A transaction may need to lock data objects even if it is not updating.

– **Definition:** (shared lock, S-lock)

If a transaction T holds a shared lock on some object O then another transaction T' can also acquire a shared lock on O, but not an exclusive lock. If transaction T holds the only shared lock on R, then T can promote it to an X-lock.

– Lock compatibility matrix

		T'	
		X	S
T	X	N	N
	S	N	Y

– Protocol PSC

1. Any transaction reading an object O must first acquire an S-lock on O (by means of an SREAD O).
2. Any transaction updating an object O must first acquire an X-lock on O (by means of an XWRITE O).
3. All locks are retained until the transaction terminates.

• Deadlocks

– A deadlock occurs when transactions wait indefinitely for each other to release resource.

– A deadlock may involve two or more transactions.

– Deadlocks can be detected by using a **wait-for** graph.

* Nodes: Transactions

* Edges: There is an edge from T_i to T_j if T_i is waiting for an object locked by T_j .

– Theorem:

Deadlock \Leftrightarrow cycle in the wait-for graph

– A deadlock can be broken by rolling back any one of the transactions on the cycle, undoing its changes, and then restarting it again.

– Deadlocks can be prevented by granting all the lock requests at the beginning of a transaction.

- **Versioning**

- Versioning is an “optimistic” method based on the assumption that most of the time users do not want the same record.
- When a transaction modifies a record, it is given a version (copy) of the record to modify.
- When there is no conflict, the transaction’s changes are merged into the database.
- When there is a conflict, the changes by one of the transactions are committed to the database. Other transactions check out another copy of the record and repeat the previous work.

Reading: Chapters 18.1, 18.2, 18.3, 18.8.