Transactions Management

(Διαχείριση Δοσοληψιών)

Pitikakis Marios, Ph.D.
Problem statement

• We focus on those parts of a DBMS that control access to data → major issues:
  – Data must be protected in case of a system failure (system resilience and integrity of the data when the system fails in some way).
  – Data must not be corrupted simply because several queries or database modifications are being done at the same time and data are shared.
Failure types

• Erroneous data entry
  – solution: write constraints and triggers that detect erroneous data

• Disk/Media failures
  – disk becomes unreadable: use RAID, maintain a backup full/incremental, replication, redundant copies etc.

• Catastrophic failure of the external environment
  – e.g. explosions, fires, vandalism etc (distributed backup, cloud)

• System failures
  – software error (bad program with bugs)
  – transaction failure (the transaction state is lost, transaction cancelled by the user, transaction deadlock etc)
  – other system failures (e.g. operating system deadlocks, electric power failure, CPU halts/resets, system crash)
**Failure types**

Events → Desired
Undesired → Expected
Unexpected

---

**Storage Hierarchy**

Temporary Memory (RAM, cache, registers)
- a
- b
- buffer

Permanent Storage Media (Disk)
- A
- B

The dotted lines represent operations that may be delayed, ignored or never be executed.

- **Read(A,a)**
- **Write(B,b)**
Transaction concurrency

• Simultaneous execution of transactions is necessary for a DBMS with good performance
  – Due to frequent and relatively slow disk access, it is important to keep the CPU busy with many user programs/transactions
  – Multi-user DBMS

• The DBMS is responsible for the “intertwining” of transactions and must be done in a “non transparent” manner to the user. The user should believe that is the only one using the system.

• Things get more difficult when there are failures.

Intertwined model of concurrent execution (transaction schedule)
How can we prevent and/or fix violations?

• Due to failures only
• Due to data sharing only
• Due to failures and sharing
Lectures overview

• Failure recovery (Αντιμετώπιση αστοχιών συστήματος)
• Logging (Undo, Redo, Undo/Redo) (Χρήση ιστορικού/πρακτικού)
• Transaction Scheduling and Serializability (Χρονοπρογράμματα Δοσοληψιών και Σειριοποιησιμότητα)
• Concurrency Control and Lock-based Protocols (Έλεγχος Συγχρονικότητας και Τεχνικές Κλειδώματος)
Lectures material

• Recommended book:
    H. Garcia-Molina, J. D. Ullman and J. D. Widom

• Chapters: 17, 18, 19

• Assignment #3 (TBA)
Transactions
(Δοσοληψίες)
DB Consistency (Συνέπεια ΒΔ)

• We would like data to be “accurate” and “correct” («ακριβή» και «σωστά») at all times

• A DB has several integrity constraints (κανόνες ακεραιότητας)
  – primary key, value constraints etc

• Moreover, there are logical constraints that we “hide” inside the applications.

• Before and after the execution of a transaction (but not necessarily in-between), all constraints must be satisfied.
DB Consistency (Συνέπεια ΒΔ)

• Examples:
  – no employee should make more than twice the average salary (value constraint)
  – when salary is updated, new salary > old salary (transaction constraint)
  – when bank account record is deleted, balance = 0 (transaction constraint)
DB Consistency (Συνέπεια ΒΔ)

Observation: DB cannot be consistent always!

Example: \( a_1 + a_2 + \ldots + a_n = \text{Total} \) (constraint)

Deposit 100€ in \( a_3 \): \( a_3 \leftarrow a_3 + 100 \)

\[
\begin{array}{c|c}
\text{a}_3 & \hline \\
30 & \\
\vdots & \\
2000 & \\
\end{array}
\quad \rightarrow \quad 
\begin{array}{c|c}
\text{a}_3 & \hline \\
130 & \\
\vdots & \\
2000 & \\
\end{array}
\quad \rightarrow \quad 
\begin{array}{c|c}
\text{a}_3 & \hline \\
130 & \\
\vdots & \\
2100 & \\
\end{array}
\]

Total \( \leftarrow \text{Total} + 100 \)

\[
\begin{array}{c|c}
\text{Total} & \hline \\
2000 & \\
\vdots & \\
\end{array}
\quad \rightarrow \quad 
\begin{array}{c|c}
\text{Total} & \hline \\
2000 & \\
\vdots & \\
\end{array}
\quad \rightarrow \quad 
\begin{array}{c|c}
\text{Total} & \hline \\
2100 & \\
\vdots & \\
\end{array}
\]

DB Consistency (Συνέπεια ΒΔ)

• Assumption (Correctness Principle):
  A transaction must leave the DB in a consistent state if the DB was consistent when the transaction started. (DB is in a consistent state: all constraints are satisfied)

• There are two major issues with transactions:
  – What happens when there is a failure or a crash during the execution of a transaction?
  – What happens when two transactions try to alter the same object at the same time?
Constraint Violations

- Constraint violations may occur due to:
  - transaction bugs
  - DBMS bugs
  - hardware failure
    - e.g., disk failure alters an attribute value
  - data sharing
    - e.g., transaction $T_1$ gives a 20% raise to all employees of category A; $T_2$ promotes employees of category A to category B

- In many cases, constraint violations can be prevented
- In other cases, constraint violations need to be fixed in order to restore consistency
The concept of a transaction

• A transaction is a set of operations that access or change (read or write) the content (objects) of a DB, and ends with a commit or an abort action.

• A transaction can
  – commit (να επικυρωθεί) when all operations are completed
  – abort (να ακυρωθεί) when some operations have been executed
Transaction states

- **BEGIN**
- **R(X)**
- **W(X)**
- **END**
- **COMMIT** (επικύρωση ή οριστικοποίηση) - success - all changes are committed and cannot be undone
- **ABORT** (ακύρωση ή ανάκληση) – failure – all changes must be undone

```plaintext
e.g.  
T1: BEGIN R(X), X=X-N, W(X), R(Y), Y=Y+N, W(Y) END  
T2: BEGIN R(X) X=X+M, W(X) END
```
Desirable properties of a transaction

- **Atomicity (Ατομικότητα)**: either all operations of the transaction succeed or all fail.
- **Consistency (Συνέπεια)**: at the end of the transaction, the DB should remain in a consistent state.
- **Isolation (Απομόνωση)**: even if many transactions are executed at the same time, every transaction should believe that is the only one executed and all intermediate results/states should not be revealed.
- **Durability (Μονιμότητα ή Διάρκεια)**: after a transaction successfully completes, changes to data persist (cannot be lost), even in the event of a system failure.

*Internationally known as ACID test*
Primitive transaction operations - Notation

- **Input**(X): Copy the disk block containing database element X → memory buffer
- **Read**(X,t): Do **Input**(X) if necessary; Assign the value of X in block → t (local variable)
- **Write**(X,t): Do **Input**(X) if necessary; copy the value of t → X in the memory buffer
- **Output**(X): Copy the block containing X from buffer → disk

- See next example: incomplete transactions create DB inconsistencies.
Transaction example

T₁:  
Read (A,t);  t ← t*2
Write (A,t);
Read (B,t);  t ← t*2
Write (B,t);
Output (A);
Output (B);

A: 5  
B: 5

A: 10
B: 10

memory

disk
Incomplete transaction

T₁: Read (A,t);  t ← t*2
    Write (A,t);
    Read (B,t);  t ← t*2
    Write (B,t);
    Output (A);
    Output (B);

We need Atomicity: either all operations are executed or none

memory disk

Inconsistent DB state

A: 5 10
B: 5 10
Failure Recovery

(Τεχνικές Ανάκαμψης από σφάλματα)
Log (Ιστορικό/Πρακτικό/Ημερολόγιο)

• The aim of recovery: restore the DB to the most recent consistent state before the moment of failure.
• The DBMS logs (καταγράφει) all operations performed by a transaction and every other important event, with the help of a log manager.
• A log (πρακτικό) is a file of log records (έγγραφες/δελτία πρακτικών). Only the append (προσθήκη) operation is allowed.
• Logs are also written in memory and are transferred to disk, with a flash-log action, at appropriate times and/or conditions.
Transaction manager and other subsystems
The concept of recovery (ανάκαμψη)

- **Transaction properties we need**
  - **Atomicity** (Ατομικότητα) - either a transaction executes in its entirety or not at all
    - Transactions may be undone ("rollback").
  - **Durability** (Μονιμότητα ή Διάρκεια) - after a transaction commits (successful completion of a transaction), changes cannot be lost
    - What will happen if the DBMS crashes? Redo
The concept of recovery (ανάκαμψη)

- E.g. desirable behavior after a failure:
  - T1, T2 & T3 must be redone (Durability)
  - T4 & T5 must be undone (Atomicity)
Log records

• Forms of log record and symbolism:
  – <START T> : This indicates that transaction T has begun
  – <COMMIT T> : Transaction T has completed successfully. Any changes to the database made by T should appear on disk. However, we cannot be sure of that ...
  – <ABORT T> : Transaction T could not complete successfully. Changes made by T must not appear on disk.

• For an undo log (πρακτικά αναίρεσης) we need:
  – update records (εγγραφές ενημέρωσης)
  – a triple <T, X, v> : transaction T has changed database element X, and its former value was v (the log record is a response to a Write action into memory, not an Output action to disk)
Undo Logging

T1: Read (A,t); t ← t*2
    Write (A,t);
    Read (B,t); t ← t*2
    Write (B,t);
    Output (A);
    Output (B);

One possible “complication”:
- The log is first written in memory
- It’s not written to disk on every action
The Undo Logging rules

- Transactions must obey to the following rules in order to allow recovery from a system failure using an undo log:
  1. If transaction T modifies a database element X, then the log record \(<T,X,v>\) must be written to disk before the new value of X is written to disk (Write Ahead Log - WAL)
  2. If a transaction commits, then its \(<\text{COMMIT } T>\) log record must be written to disk only after all database elements changed by the transaction have been written to disk, but as soon as possible.

- These rules impose the following disk write order:
  a) The log records, which indicate changed database elements.
  b) The changed database elements themselves.
  c) The COMMIT log record.

- To force log records to disk, the log manager needs a FLUSH LOG command.
# Undo Logging example

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
<th>t</th>
<th>A</th>
<th>B</th>
<th>A</th>
<th>B</th>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initially</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>Read (A, t)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>t ← t*2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Write (A, t)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Read (B, t)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>t ← t*2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Write (B, t)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>FLUSH LOG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Output (A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Output (B)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>FLUSH LOG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Undo Logging example

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
<th>t</th>
<th>A</th>
<th>B</th>
<th>A</th>
<th>B</th>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initially</td>
<td></td>
<td></td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td>&lt;START T&gt;</td>
</tr>
<tr>
<td>2</td>
<td>Read (A,t)</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>t ← t*2</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Write (A,t)</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;T, A, 5&gt;</td>
</tr>
<tr>
<td>5</td>
<td>Read (B,t)</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>t ← t*2</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Write (B,t)</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;T, B, 5&gt;</td>
</tr>
<tr>
<td>8</td>
<td>FLUSH LOG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Output (A)</td>
<td></td>
<td></td>
<td></td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Output (B)</td>
<td></td>
<td></td>
<td></td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;COMMIT T&gt;</td>
</tr>
<tr>
<td>12</td>
<td>FLUSH LOG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Recovery using Undo Logging

- In case of a failure, the Recovery Manager (Διαχειριστής Αποκατάστασης) must restore the DB in a consistent state using the log.

- We consider the simplest form of recovery manager:
  - looks at the entire log, no matter how long, and makes database changes as a result of its examination.

- The recovery manager divides the transactions into committed and uncommitted
  - If there is a log record <COMMIT T>, then, based on rule #2, all changes made by transaction T were already written to disk. Therefore T could not have left the database in an inconsistent state.
  - If we find a <START T> record but no <COMMIT T>, then T is considered incomplete (ανολοκλήρωτη) and must be undone (αναιρεθεί)
Recovery using Undo Logging

• **Undo** (Αναίρεση) means that whatever changes T made must be reset to their previous value.
• Rule #1 assures that if T changed X on disk before the crash, then there will be a <T, X, v> record on the log, and that record will have been copied to disk before the crash.
• During the recovery, we must write the value v to element X.
• There may be several uncommitted transactions in the log that modified X. Therefore, we have to be systematic and respect the order in which we restore values.
• The log must be scanned in reverse order: from the most recently written record to the earliest written
• A transactions list must be kept for which it has seen a <COMMIT T> record or an <ABORT T> record.
Recovery using Undo Logging

• If it sees a \(<T, X, v>\) record:
  – If a \(<\text{COMMIT } T>\) record has been seen, then do nothing. \(T\) is committed and must not be undone.
  – If \(T\) is an incomplete or an aborted transaction, then the recovery manager must change the value of \(X\) to \(v\).

• After making these changes, the recovery manager must write a \(<\text{ABORT } T>\) log record for each incomplete transaction \(T\), and then flush the log.

• This approach is simple but not very efficient:
  – the entire execution history log must be examined every time a failure occurs
  – an improvement of this relies on **checkpointing** the log (σημεία ελέγχου) in order to limit the extent to which the log must be examined (more on checkpoints later).
Undo Logging: Recovery rules

• For every Ti with <START Ti> in log:
  – If <COMMIT Ti> or <ABORT Ti> do nothing in log
  – Else

```
For all <Ti, X, v> in log:
  Write (X, v)
  Output (X)
Write <ABORT Ti> to log
```

Is this correct?
Undo Logging: Recovery rules

1) Let \( S = \) set of transactions with \(<\text{START } Ti>\) in the log, but no \(<\text{COMMIT } Ti>\) or \(<\text{ABORT } Ti>\) record in the log

2) For each \(<Ti, X, v>\) in the log, in reverse order (latest \(\rightarrow\) earliest) do:

   if \( Ti \in S \) then
   \[
   \begin{cases}
   \text{Write } (X, v) \\
   \text{Output } (X)
   \end{cases}
   \]

3) For each \( Ti \in S \) do:

   Write \(<\text{ABORT } Ti>\) to log
## Recovery example

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
<th>t</th>
<th>A</th>
<th>B</th>
<th>A</th>
<th>B</th>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initially</td>
<td></td>
<td></td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Read (A,t)</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td>&lt;START T&gt;</td>
</tr>
<tr>
<td>2</td>
<td>t ← t*2</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Write (A,t)</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;T, A, 5&gt;</td>
</tr>
<tr>
<td>4</td>
<td>Read (B,t)</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>t ← t*2</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Write (B,t)</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;T, B, 5&gt;</td>
</tr>
<tr>
<td>7</td>
<td>FLUSH LOG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Output (A)</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Output (B)</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;COMMIT T&gt;</td>
</tr>
<tr>
<td>10</td>
<td>FLUSH LOG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Undo Logging complications

• Logging actions do not occur in isolation. Since many transactions execute simultaneously, log records for one transaction may be interleaved with similar operations for other transactions.

• Flushing the log may imply that log records for a transaction appear on disk earlier than intended.

• No harm as long as the <COMMIT T> record is written only after the output actions of T are completed.

• If elements A and B share a block, writing one of them, implies writing the other as well (rule #1 may be violated).

• Need to impose additional constraints or impose a concurrency protocol (Έλεγχο Συνδρομικότητας)
  — more on concurrency control later.