# 21 Database Foundation: *Transactions*

Summer Term 2010

Robert Elsässer

Albert-Ludwigs-Universität Freiburg

# Basics

- A database is a set of objects.

  logical units: *relation*, *tuple*,

  physical units: *block*, *page*.
- A transaction is a process having access to a database.
- Transaction $T$ may read and write a database object: a sequence of read- and write-operations.
  - $RA$: the current value of $A$ in the database is copied into the local address space of the respective transaction.
  - $WA$: the value of $A$ in the local address space is copied in the database and becomes the new current value of $A$.
- read and write are atomic operations.

# Schedule

- Let $\mathcal{T} = \{T_1, \ldots, T_n\}$ a set of transactions.
- The sequence of read- and write-operations of a transaction $T_i \in \mathcal{T}$ is called its *history* $h_i$.
- An execution of the transactions in $\mathcal{T}$ is called a *schedule* $S$ of $\mathcal{T}$.
- A schedule is a sequence of read- and write-operations of the transactions in $\mathcal{T}$.
- The relative order of the operations in a transaction $T$ mentioned in $S$ is consistent with the history $h$ of $T$.
- A *serial* schedule of $\mathcal{T}$ is a concatenation of the histories of the transactions in $\mathcal{T}$.

# Example

- $\mathcal{T} = \{T_1, T_2, T_3\}$, where $T_1 = R_1A\ W_1A\ R_1B\ W_1B$, $T_2 = R_2A\ W_2A\ R_2B\ W_2B$ and $T_3 = R_3A\ W_3B$.
- There exist six serial schedules of $\mathcal{T}$, e.g. $S_1 = h_1h_2h_3$, $S_2 = h_2h_3h_1$.
- The following are not serial:

$$S_3 = R_1A\ W_1A\ R_3A\ R_1B\ W_1B\ R_2A\ W_2A\ W_3B\ R_2B\ W_2B,$$
$$S_4 = R_3A\ R_1A\ W_1A\ R_1B\ W_1B\ R_2A\ W_2A\ R_2B\ W_2B\ W_3B.$$

# augmented schedule

- $T_0$ is a transaction with a write for each database object and no read. $T_0$ will create the initial state of the database.
- $T_\infty$ has a read for each object and no writes. It will read the final state of the database. $T_\infty$ reads the final state of the database.
- $S$ a schedule to $\mathcal{T}$. $\widehat{S} = T_0 \ S \ T_\infty$ is the augmented schedule.

# Concurrency Control

**Problem**

$T_1$ adds 100 to $A$; $T_2$ subtracts 50 from $A$.

| $S_1$ | $S_2$ | $S_3$ | $S_4$ | $S_5$ | $S_6$ |
|---|---|---|---|---|---|
| $A = 80$ | $A = 80$ | $A = 80$ | $A = 80$ | $A = 80$ | $A = 80$ |
| $R_1A$ | $R_1A$ | $R_1A$ | $R_2A$ | $R_2A$ | $R_2A$ |
| $W_1A$ | $R_2A$ | $R_2A$ | $W_2A$ | $R_1A$ | $R_1A$ |
| $R_2A$ | $W_1A$ | $W_2A$ | $R_1A$ | $W_2A$ | $W_1A$ |
| $W_2A$ | $W_2A$ | $W_1A$ | $W_1A$ | $W_1A$ | $W_2A$ |
| $A = 130$ | $A = 30$ | $A = 180$ | $A = 130$ | $A = 180$ | $A = 30$ |

Which of the six schedules can be considered correct?

# Serializability

## Definition

A schedule is called *serializable*, if there exists an equivalent schedule with the same transactions.

## Definition

Schedule $S$ and $S'$ over the same set of transactions are *equivalent*, if for any initial state of the database and any possible semantics of the transactions the following holds.

- The transactions read in $S$ und $S'$ the same values.
- $S$ und $S'$ produce the same final state of the database.

# Example

- $T_1 = R_1A\ W_1A\ R_1B\ W_1B, \quad T_2 = R_2A\ W_2A\ R_2B\ W_2B.$

- $$S_1 = R_1A\ W_1A\ R_2A\ W_2A\ R_2B\ W_2B\ R_1B\ W_1B$$
  $$S_2 = R_1A\ W_1A\ R_2A\ W_2A\ R_1B\ W_1B\ R_2B\ W_2B$$

What's about semantics?

| | schedule $T_1 T_2$ | | schedule $T_2 T_1$ |
|---|---|---|---|
| $R_1A$ | $A_0$ | $R_2A$ | $A_0$ |
| $W_1A$ | $f_{T_1,A}(A_0)$ | $W_2A$ | $f_{T_2,A}(A_0)$ |
| $R_1B$ | $B_0$ | $R_2B$ | $B_0$ |
| $W_1B$ | $f_{T_1,B}(A_0, B_0)$ | $W_2B$ | $f_{T_2,B}(A_0, B_0)$ |
| $R_2A$ | $f_{T_1,A}(A_0)$ | $R_1A$ | $f_{T_2,A}(A_0)$ |
| $W_2A$ | $f_{T_2,A}(f_{T_1,A}(A_0))$ | $W_1A$ | $f_{T_1,A}(f_{T_2,A}(A_0))$ |
| $R_2B$ | $f_{T_1,B}(A_0, B_0)$ | $R_1B$ | $f_{T_2,B}(A_0, B_0)$ |
| $W_2B$ | $f_{T_2,B}(f_{T_1,A}(A_0), f_{T_1,B}(A_0, B_0))$ | $W_1B$ | $f_{T_1,B}(f_{T_2,A}(A_0), f_{T_2,B}(A_0, B_0))$ |

# Dependency graph

A *dependency graph* of schedule $S$ is a directed graph $AG(S) = (V, E)$, $V$ the set of operations in $S$ $\widehat{S}$ and $E$ a set of edges $(i \neq j)$:

- $\widehat{S} = \ldots R_i B \ldots W_i A \ldots \;\Rightarrow\; R_i B \rightarrow W_i A \in E$,
- $\widehat{S} = \ldots W_i A \ldots R_j A \ldots \;\Rightarrow\; W_i A \rightarrow R_j A \in E$, if between $W_i A$ and $R_j A$ in $\widehat{S}$ there are no other writes to $A$.

## Theorem

Schedules $S$ and $S'$ over the same transactions are equivalent, if $AG(S) = AG(S')$.

# Conflict graph

*Conflict Graph* of $S$ is a directed graph $KG(S) = (V, E)$, where $V$ set of transactions in $\widehat{S}$ and $E$ the set of edges $(i \neq j)$:

- $\widehat{S} = \ldots W_i A \ldots R_j A \ldots \;\Rightarrow\; T_i \rightarrow T_j \in E$, if between $W_i A$ and $R_j A$ in $\widehat{S}$ no other writes to $A$. *(WR-conflict)*
- $\widehat{S} = \ldots W_i A \ldots W_j A \ldots \;\Rightarrow\; T_i \rightarrow T_j \in E$, if between $W_i A$ and $W_j A$ in $\widehat{S}$ no other writes to $A$. *(WW-conflict)*
- $\widehat{S} = \ldots R_i A \ldots W_j A \ldots \;\Rightarrow\; T_i \rightarrow T_j \in E$, if between $R_i A$ and $W_j A$ in $\widehat{S}$ no other writes to $A$. *(RW-conflict)*

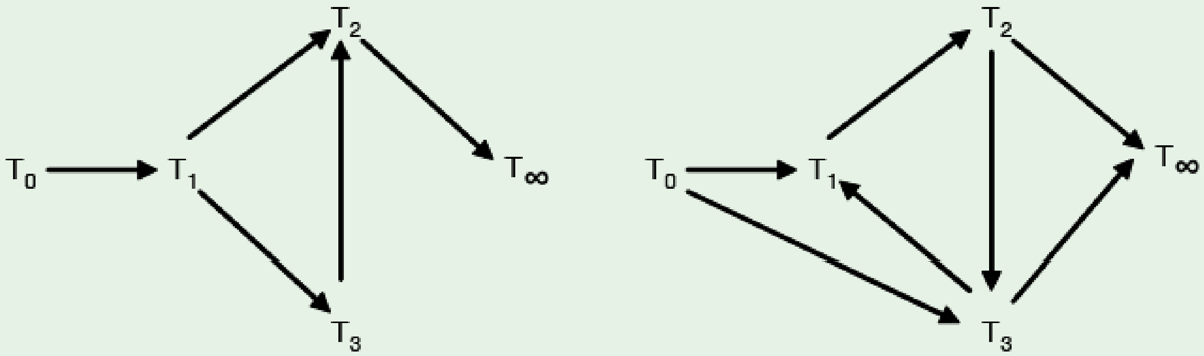
## Theorem and definition

- Schedule $S$ is serializable, if $KG(S)$ has no cycle.
- Schedule $S$ is called *conflict-serializable*, if $KG(S)$ has no cycle.

# Example

Schedule $S_1$: $R_1A\ W_1A\ R_3A\ R_1B\ W_1B\ R_2A\ W_2A\ W_3B\ R_2B\ W_2B$

Schedule $S_2$: $R_3A\ R_1A\ W_1A\ R_1B\ W_1B\ R_2A\ W_2A\ R_2B\ W_2B\ W_3B$



# Locking

- Before reading and writing a lock has to be obtained.
- (*Lock*):
  - *Read-lock* $L^R A$
  - *Read- and Write-lock* $LA$
- (*Unlock*): $UA$, respectively $U^R A$.
- *Locktable*
- *Compatibilitymatrix*:

lock hold to $A$:

| lock acquired $A$: | $L^R A$ | $LA$ |
|---|---|---|
| $L^R A$ | $J$ | $N$ |
| $LA$ | $N$ | $N$ |

- *Livelock* and *Deadlock*.

# Livelocks and Deadlocks

- avoid Livelocks: *first-come-first-served*-strategy
- avoid Deadlocks:
  - When being started, each transaction acquires for all locks in one atomic operation.
  - A linear order is defined on all objects; locks are acquired consistently to this order.
- *Wait-for-graph*: There is an edge $T_i \rightarrow T_j$, if $T_i$ acquires a lock, which $T_j$ obtains and the acquired and the obtained locks are not compatible.

  There is a deadlock, iff there is a cycle in the wait-for-graph.

How to break a deadlock?

# 2-Phase Locks 2PL

After the first unlock, it is not allowed to lock again.

Lock- and unlock-operations of a 2PL transaction *RA WA RB WB RC WC*

*LA RA WA LB RB WB LC RC WC UA UB UC*,
*LA RA WA LB LC UA RB WB UB RC WC UC*,
*LA LB LC RA WA UA RB WB UB RC WC UC*,
*LA LB LC RA WA RB WB RC WC UA UB UC*.

2PL is called *strict*, if all unlock are postponed to the end of a transaction.

**Satz**

2Pl guarantees serializability.

Proof!

# Power and optimality of 2PL

- 2PL is optimal in the sense, that for any non-2PL transaction $T$ there exists a transaction $T'$, such that there exists a not serializable schedule to $\{T, T'\}$.
- There exist serializable schedules, which cannot occur under 2PL.

# Methods without locks

- A concurrency control can be formalized as a mapping $\Phi$, which transforms an acquired sequence of operations $S_I$ (input-schedule) into a serializable sequence of operations $S_O$(output-schedule) which then is being executed.
- $\Phi(S_I) = S_O$, where $S_I$ is a prefix of a schedule and $S_O$ a schedule.

Consider $\Phi_{2PL}$. $T_1 = L_1A\ R_1A\ L_1B\ U_1A\ W_1B\ U_1B$, $T_2 = L_2A\ R_2A\ W_2A\ U_2A$, and $T_3 = L_3^RC\ R_3C\ U_3^RC$.

| acquired sequence | locktable | executed sequence |
|---|---|---|
| $L_1A$ | $L_1A$ | |
| $L_1A\ R_1A$ | $L_1A$ | $R_1A$ |
| $L_1A\ R_1A\ L_2A$ | $L_1A$ | $R_1A$ |
| $L_1A\ R_1A\ L_2A\ L_3^RC$ | $L_1A, L_3^RC$ | $R_1A$ |
| $L_1A\ R_1A\ L_2A\ L_3^RC\ R_3C$ | $L_1A, L_3^RC$ | $R_1A\ R_3C$ |
| $L_1A\ R_1A\ L_2A\ L_3^RC\ R_3C\ L_1B$ | $L_1A, L_3^RC, L_1B$ | $R_1A\ R_3C$ |
| $L_1A\ R_1A\ L_2A\ L_3^RC\ R_3C\ L_1B\ U_1A$ | $L_3^RC, L_1B$ | $R_1A\ R_3C$ |
| $L_1A\ R_1A\ L_2A\ L_3^RC\ R_3C\ L_1B\ U_1A\ R_2A$ | $L_3^RC, L_1B, L_2A$ | $R_1A\ R_3C\ R_2A$ |
| $L_1A\ R_1A\ L_2A\ L_3^RC\ R_3C\ L_1B\ U_1A\ R_2A\ W_2A$ | $L_3^RC, L_1B, L_2A$ | $R_1A\ R_3C\ R_2A\ W_2A$ |
| $L_1A\ R_1A\ L_2A\ L_3^RC\ R_3C\ L_1B\ U_1A\ R_2A\ W_2A\ U_2A$ | $L_3^RC, L_1B$ | $R_1A\ R_3C\ R_2A\ W_2A$ |
| ... | ... | ... |

# Conict-graph analysis $\Phi_{KG}$

Let $S$ be the current sequence of operations being executed and let $op$ be the next operation being acquired for execution of a transaction $T$.

If $KG(S\ op)$ acyclic, then execute $op$. Otherwise cancel $T$ and all transactions depending on $T$ and delete all their operations from $S$.

# Timestamps $\Phi_{ZM}$

Each transaction $T$ is assigned an unique timestamp $Z(T)$ at its start.

Let $S$ be the current sequence of operations being executed and let *op* be the next operation being acquired for execution of a transaction $T$.

If for all transactionen $T'$, which have already executed an operation which is in conflict with *op* there hods $Z(T') \leq Z(T)$, then execute *op*. Otherwise cancel $T$ and all transactions depending on $T$ and delete all their operations from $S$.

# Example

$S_I = R_1A\ R_2A\ W_2A\ R_3B\ W_3B\ W_1B.$
$T_1 = R_1A\ W_1B,\ T_2 = R_2A\ W_2A,\ T_3 = R_3B\ W_3B.$

| $S_I$ | $S_O$ |
|---|---|
| $\Phi_{KG}$ | |
| $\Phi_{ZM}$ | |
| $\Phi_{2PL}$ | |

# Phantoms

**implicit assumption**

The set of objects does not change.

If not guaranteed: *Phantoms*.

# Schedule with phantom

Consider $T_1$ with history $R_1A_1\ \ldots R_1A_k\ R_1B$.

Consider transaction $T_2$ with history $R_2C\ W_2A_{k+1}\ W_2B$.

All $A_i$ fulfill a predicate $p$. Assume, $T_1$ wants to read all objects which fulfill $p$.

$$R_2C\ R_1A_1\ \ldots R_1A_k\ W_2A_{k+1}\ R_2B\ W_2B\ R_1B$$

This schedule formally is equivalent to $T_2\ T_1$; however, this is a wrong conclusion.

# Solution to phantoms

- Enlarge granularity of objects.
- Consider read of the form $R_1\{A \mid p(A)\}$.
- Index-locking