Database Management Systems II

Robert Gottstein

gottstein@dvs.tu-darmstadt.de

The Two-Phase-Locking Protocol (2PL)

Exercise 6.1
The Two-Phase-Locking Protocol (2PL)

TECHNISCHE UNIVERSITÄT DARMSTADT

a) Show that the ***set of histories produced by 2PL*** schedulers is ***a strict subset of CSR.***

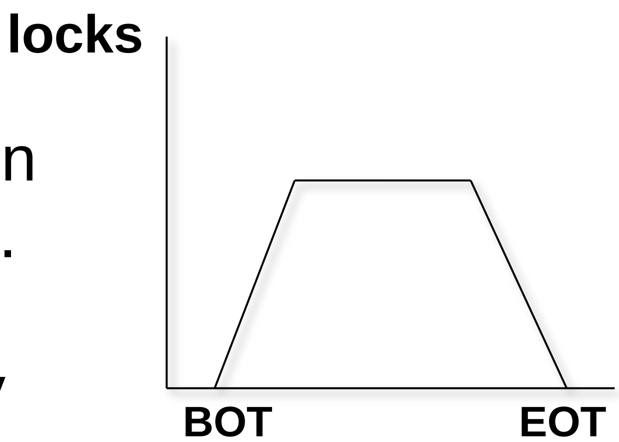
Exercise 6.1
The Two-Phase-Locking Protocol (2PL)

Basic 2PL

1. ***Conflicting operations scheduled*** in the order in which locks are obtained.

2. ***Handshake principle*** - ensures they are also processed in this order.

3. ***2 Phase Rule:* Once a lock** is released, **no more lock requests.**



Upper layer waits for confirmation that command was executed.

Exercise 6.1
The Two-Phase-Locking Protocol (2PL)

TECHNISCHE UNIVERSITÄT DARMSTADT

Definition **Conflict-Equivalence:**
Two histories are **conflict equivalent** if they have the same operations and they order conflicting operations in the same order.

Conflicting operations - operations whose effect depends on the order in which they are executed

Definition: A history H is **Conflict-Serializable (CSR)** if C(H) is conflict equivalent to a serial history.

Conflict serializability can easily be enforced using an efficient algorithm

Exercise 6.1
The Two-Phase-Locking Protocol (2PL)

$\mathcal{H}_{[2PL]} \subset \mathcal{H}_{[CSR]}$????

i. $\mathcal{H}_{[2PL]} \subseteq \mathcal{H}_{[CSR]}$

ii. $\mathcal{H}_{[2PL]} \subset \mathcal{H}_{[CSR]}$

i. $\mathcal{H}_{[2PL]} \subseteq \mathcal{H}_{[CSR]}$

Proof: see lecture

Idea:

A cycle in SG would imply a violation of 2PL's 2 phase rule.

Exercise 6.1
The Two-Phase-Locking Protocol (2PL)

ii. $\mathcal{H}_{[2PL]} \subset \mathcal{H}_{[CSR]}$

Must show that $\exists$ H : (H $\in \mathcal{H}_{[CSR]} \wedge$ H $\notin \mathcal{H}_{[2PL]}$)

Idea:
Request a lock after having released one, thereby violating the 2 phase rule.

Exercise 6.1
The Two-Phase-Locking Protocol (2PL)

1: $\mathbf{r_1[x]}$ $\mathbf{w_1[z]c_1}$

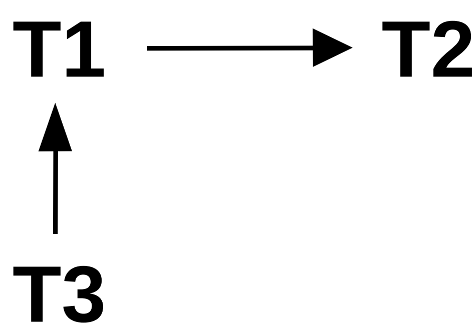
2: $w_2[x]$ c_2

3: $r_3[z]$ $w_3[z]\ c_3$

$$H = r_3[z]\ \mathbf{r_1[x]}\ w_2[x]\ w_3[z]\ c_3\ \mathbf{w_1[z]}\ \mathbf{c_1}\ c_2$$

CSR?

Yes→



Exercise 6.1
The Two-Phase-Locking Protocol (2PL)

1: **$r_1[x]$** **$w_1[z]c_1$**

1: **$rl_1[x]$ $r_1[x]$ $ru_1[x]$ $\longrightarrow$ $wl_1[z]w_1[z]$ $wu_1[z]$**

2: $w_2[x]$ c_2

2: $wl_2[x]$ $w_2[x]$ $wu_2[x]$ c_2

3: *$rl_3[z]$ $r_3[z]$ $wl_3[z]$ $w_3[z]$ $ru_3[z]$ $wu_3[z]$ c_3*

3: *$r_3[z]$ $w_3[z]$ c_3*

H = *$r_3[z]$* **$r_1[x]$** $w_2[x]$ *$w_3[z]$ c_3* **$w_1[z]$ c_1** c_2

H = *$r_3[z]$* **$r_1[x]$** $w_2[x]$ *$w_3[z]$ c_3* **$w_1[z]$ c_1** c_2

2PL? No

Exercise 6.1
The Two-Phase-Locking Protocol (2PL)

TECHNISCHE UNIVERSITÄT DARMSTADT

b) Give an example of a history that is **produced by a 2PL** scheduler, **but is *not strict***. What extension of 2PL could you suggest, that would guarantee that **only ST histories are produced**?

Exercise 6.1
The Two-Phase-Locking Protocol (2PL)

Strict (ST)

ST Rule: $W_j[x] < O_i[x] \Rightarrow (A_j < O_i[x]) \vee (C_j < O_i[x]),\ i \neq j$

Disallows: *w1[x] o2[x] , where o* $\in$ *{r,w}*

Extends ACA by preventing transactions ***not only to read from active transactions, but also to overwrite any data written*** by them. (*Overwrite only iff creator transaction finished*)

ST is also optional, but if enforced further simplifies the abort operation, by allowing **before-images** to be used

Exercise 6.1
The Two-Phase-Locking Protocol (2PL)

H = $\mathbf{r_1[x]\ w_1[x]}\ r_2[x]\ w_2[x]\ \mathbf{c_1}\ c_2$

2PL?

H = $\mathbf{rl_1[x]\ r_1[x]\ wl_1[x]\ w_1[x]\ wu_1[x]\ ru_1[x]}\ rl_2[x]\ r_2[x]\ wl_2[x]\ w_2[x]\ wu_2[x]\ ru_2[x]\ \mathbf{c_1}\ c_2$

➔Yes

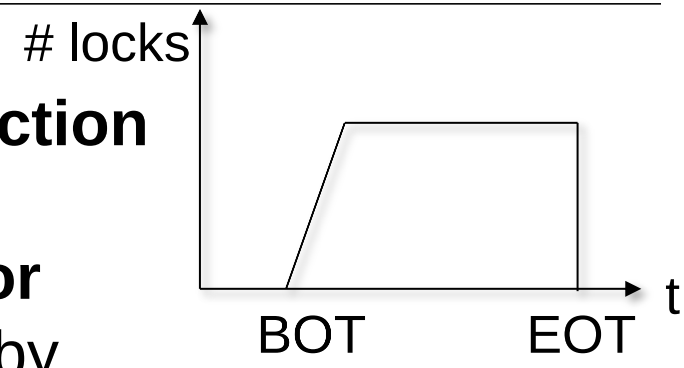
ST?

➔No

How to extend 2PL to guarantee that only ST histories are produced?

Exercise 6.1
The Two-Phase-Locking Protocol (2PL)

Strict 2PL



1. **All locks released after the transaction terminates** (commits or aborts).
   More specifically, after the **commit or abort operation is acknowledged** by the DM.
2. Since **ST is only concerned with "ww" and "wr" conflicts and not with "rw" conflicts**, we can **allow read locks to be released earlier**, i.e. in the absence of additional information the scheduler **can release read locks when a termination op. is sent** (C_i or A_i) and write locks after the op. is processed and acknowledged by the DM.

Exercise 6.1
The Two-Phase-Locking Protocol (2PL)

Be careful: **DIFFERENT DEFINTIONS EXIST**

Weikum / Vossen: Transactional Information Systems:

Under **strict 2PL** all (exclusive) write locks that a transaction has acquired are held until the transaction terminates.

Under **strong 2PL**, all locks (i.e. , both (exclusive) write and (shared) read locks), that a transaction has acquired are held until the transaction terminates.

Exercise 6.2

Deadlock Management

Exercise 6.2
Deadlock Management

TECHNISCHE UNIVERSITÄT DARMSTADT

a) What is **understood by a deadlock**?
Show how deadlocks could occur under the 2PL protocol.

Deadlock:

$\mathbf{rl_1[x]}$ $rl_2[y]$ $\mathbf{wl_1[y]}$ **<blocked T1>** $wl_2[x]$ <blocked T2>

Wait-for Graph (WFG):

$T1 : wl_1[y] \rightarrow ru_2[y] \ : T2$

$T1 : ru_1[x] \leftarrow wl_2[x] \ : T2$

Deadlocks are also caused by
lock conversion / upgrade.

flashyDB DVS

Exercise 6.2
Deadlock Management

b) How can we **detect / eliminate deadlocks** in **locking-based schedulers**?

Detection:

- **WFG** - Tested for cycles; impl. overhead
- **Timeouts** - phantom deadlocks possible, timeout interval important, more like a prevention scheme

Elimination:

- Deadlocks eliminated **by aborting a transaction**
- Victim Selection, Fairness and Starvation?

Exercise 6.2
Deadlock Management

- Criteria for selecting a victim for roll-back
  - **minimization** of **lost work (define *work*)**
  - minimization of **roll-back costs**
  - **priority** for transactions that must finish
  - maximization of broken-up cycles
  - ***Example:*** Sybase chooses transaction that has consumed the least CPU-time
- Avoiding starvation
  - transactions may participate repeatedly in cycles
  - selection algorithm must be fair

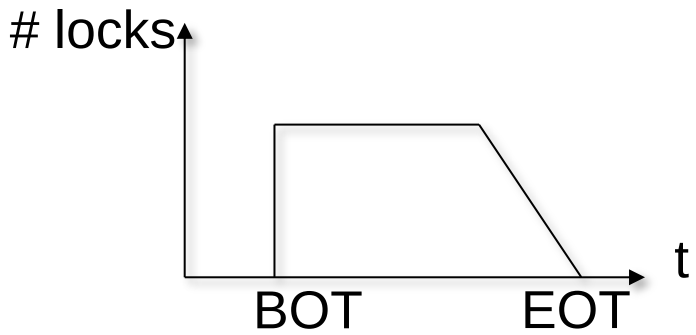
Exercise 6.2
Deadlock Management

c) Discuss the **different ways in which deadlocks can be avoided.**

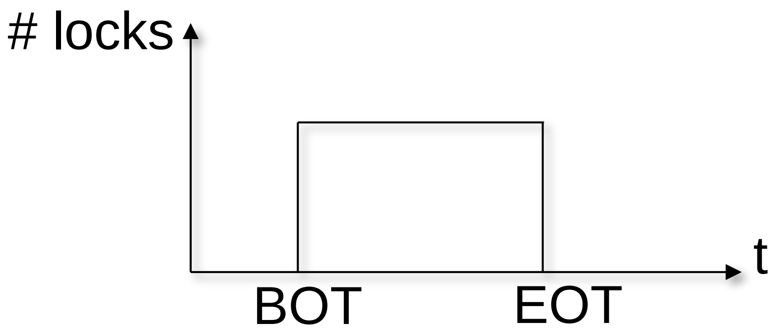
- Preclaiming (**Conservative 2PL**)
- Priority-based techniques

Exercise 6.2
Deadlock Management

Conservative 2PL – Preclaiming



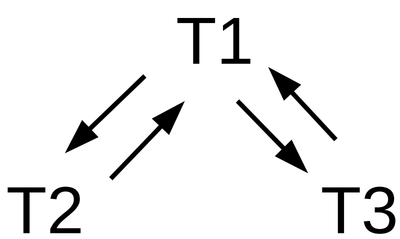
Conservative Strict 2PL



Exercise 6.2
Deadlock Management

d) Is it possible that **a transaction participates in more than one deadlock?**

WFG:



T1 waits for both T2 and T3

T1: $rl_1[x]$ $r_1[x]$ $rl_1[y]$ $r_1[y]$ $wl_1[u]$ <blocked>
T2: $rl_2[u]$ $r_2[u]$ $wl_2[x]$ <blocked>
T3: $rl_3[u]$ $r_3[u]$ $wl_3[y]$ <blocked>

Lock queues:

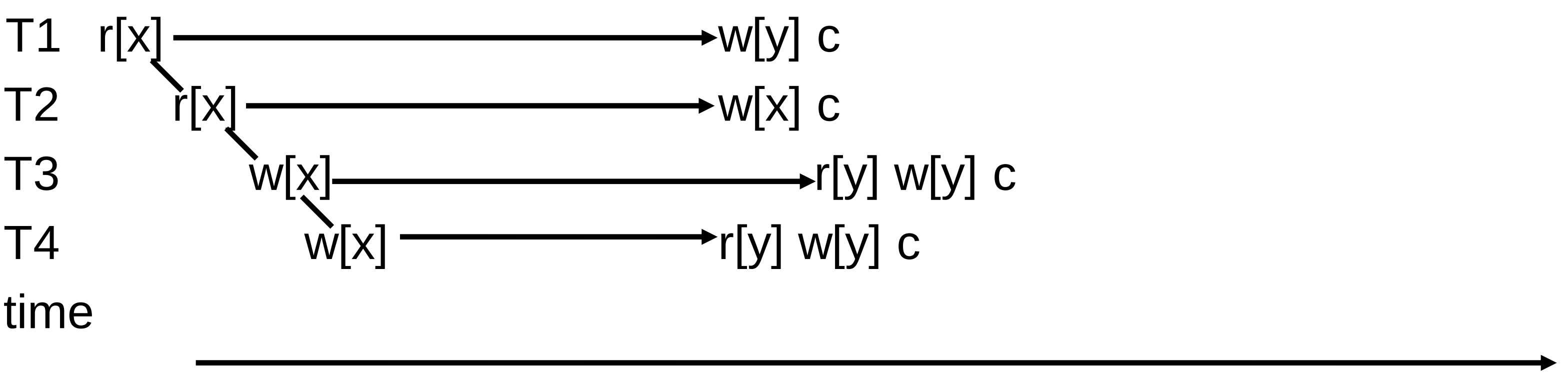
$x : rl_1 \mid wl_2$
$y : rl_1 \mid wl_3$
$u : rl_2, rl_3 \mid wl_1$

Exercise 6.3

2PL Scheduler

Exercise 6.3
2PL Scheduler

a) Given is the following partially ordered history over the transactions T1, T2, T3 and T4:



Assuming that **Strict 2PL** is used for concurrency control, describe the corresponding locking operations and managed state information (locks, waiting queues). Operations of individual transactions can be executed at the times shown in the diagram the earliest. Delays caused by blocking are potentially possible.

Exercise 6.3
2PL Scheduler

a) Strict 2PL Scenario

Op.	Lock x	Lock y	Queues	Comments
rl1[x]	**r1**			
r1[x]	**r1**			
rl2[x]	**r1** *r2*			
r2[x]	"			
wl3[x]	"		wl3[x]	T3 blocked
wl4[x]	"		wl3[x], wl4[x]	T4 blocked
wl1[y]	"	**w1**	"	
wl2[x]	"	"	wl3[x], wl4[x], *wl2[x]*	T2 is blocked
w1[y]	"	"	"	
C1	"	"	"	
……				

Exercise 6.3
2PL Scheduler

b) Given is the history:

$H = r_1[x]\ r_2[z]\ r_3[y]\ r_3[z]\ w_2[z]\ c_3\ r_1[z]\ w_1[y]\ r_2[x]\ c_1\ c_2$

Can this be a history produced by a 2PL Scheduler? Is this a strict history?

Exercise 6.3
2PL Scheduler

$H = r_1[x]\ r_2[z]\ r_3[y]\ r_3[z]\ w_2[z]\ c_3\ r_1[z]\ w_1[y]\ r_2[x]\ c_1\ c_2$

1: $\mathbf{r_1[x]}$ $\mathbf{r_1[z] w_1[y]}$ $\mathbf{c_1}$

2: $\underline{r_2[z]}$ $\underline{w_2[z]}$ $\underline{r_2[x]}$ $\underline{c_2}$

3: $r_3[y]\ r_3[z]$ c_3

i. Is r3[z] → w2[z] a problem?

Depends on the ***implementation of the LM*** and the ***availability of the information*** that T3 doesn't have any more operations after $r_3[z]$.

ii. w2[z] → r1[z] →r2[x]

T2 must release its lock on 'z' and then later request a read lock on 'x'. **This contradicts 2PL!**

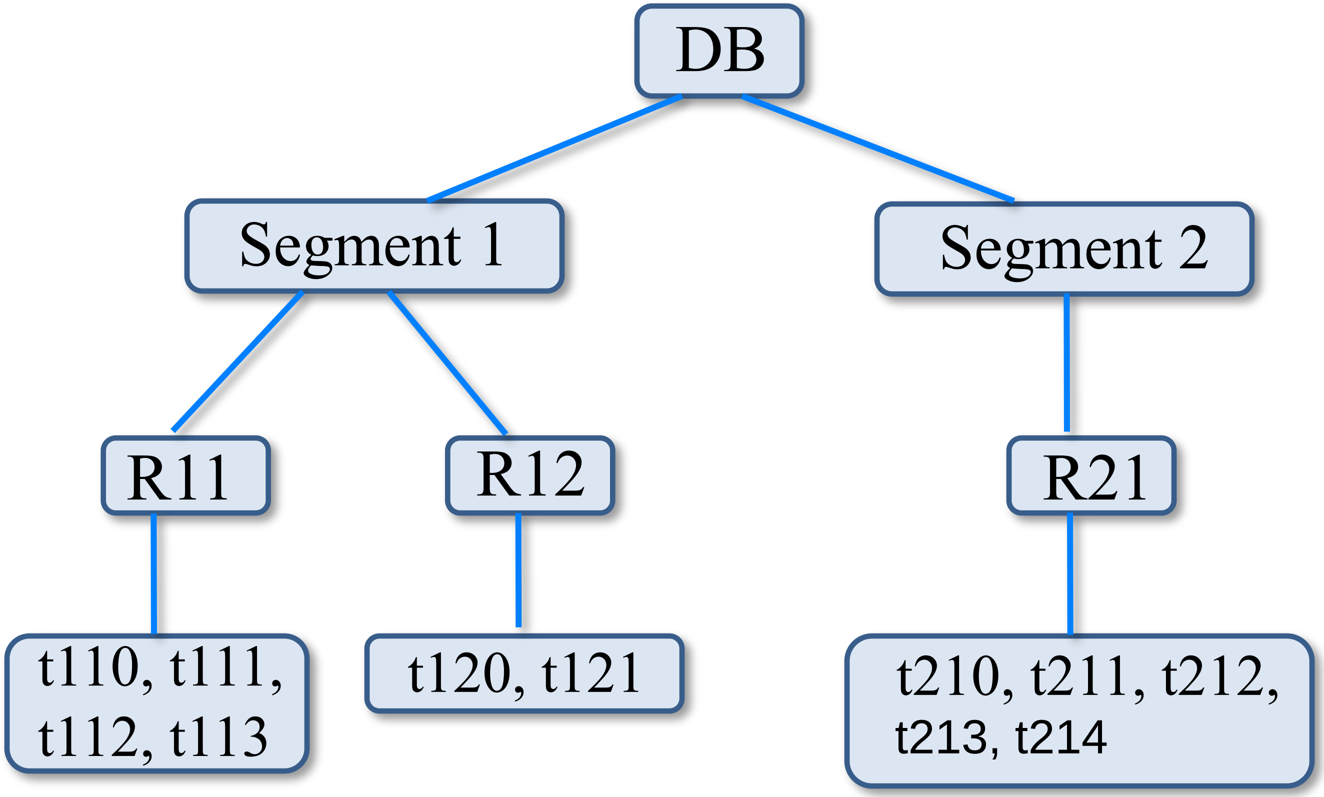
Exercise 6.5

TECHNISCHE UNIVERSITÄT DARMSTADT

Multi-Granularity Locking (MGL)

Exercise 6.5
Multi-Granularity Locking (MGL)

The diagram below shows the data graph of a RDBMS:



Exercise 6.5
Multi-Granularity Locking (MGL)

a) Explain how the **MGL** Method works.
- Discuss the **lock types and the compatibility matrix for MGL**.
- In what **order must locks be set / released in MGL**?

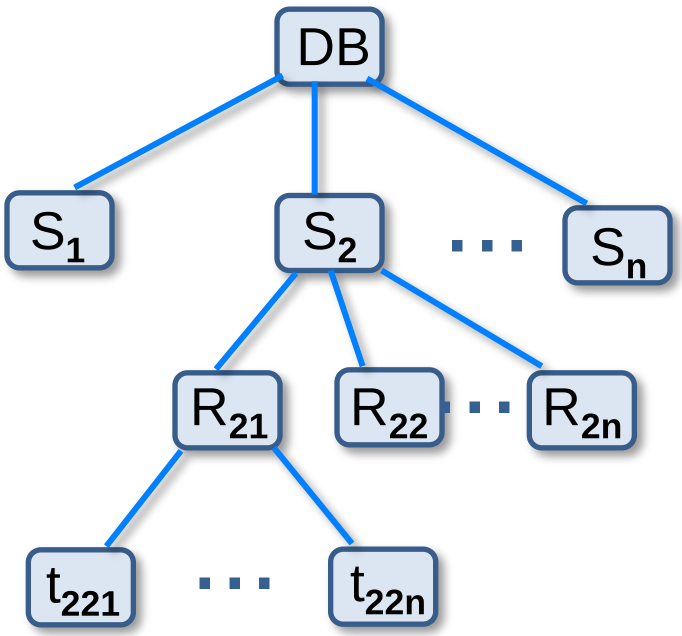
b) What locks need to be set if the following transactions are executed:
- T1 ***reads tuples*** t110, t111, t112, t113
- T2 ***reads tuples*** t210 and ***writes*** t121
- T3 ***reads tuples*** t120, t121 ***updates*** either t120 or t121
- T4 ***performs a nested-loop-join*** of R11 and R12.

c) To what extent does MGL provide ***a solution for the Phantom problem***?

Exercise 6.5
Multi-Granularity Locking (MGL)

a) Explain how the **MGL** Method works.

- Discuss the **lock types and the compatibility matrix for MGL**.
- In what **order must locks be set / released in MGL**?



Exercise 6.5
Multi-Granularity Locking (MGL)

Intention locks result in 5 lock modes instead of 2

Lock types

ir	IS	intention to read lower objects that may be locked IS or S
iw	IX	intention to write lower objects that may be locked in any mode
r	S	read lock on node and all its successors
w	X	write lock, allows exclusive access to node and successors
riw	SIX	allows read access to node and declares intention to modify successors. These may be locked in X, SIX or IX modes

Exercise 6.5
Multi-Granularity Locking (MGL)

TECHNISCHE UNIVERSITÄT DARMSTADT

Required locks for parent

Lock type	Required locks for parent
s	is, ix
is	is, ix
ix	ix, six
x	ix, six
six	ix, six

Exercise 6.5
Multi-Granularity Locking (MGL)

TECHNISCHE UNIVERSITÄT DARMSTADT

Locks should be ***released in leaf-to-root order***, in contrast to the ***root-to-leaf*** order in which they were ***obtained***.

Otherwise the following could occur:

1. T1 sets is(DB), is(Seg1), is(R11), s(t110) and reads tuple t110
2. T1 releases lock is(R11) before s(t110)
3. T2 can then set ix(DB) ix(Seg1), x(R11) and write t110.
4. Since T1 still owns the lock s(t110), it can read t110.

➔This violates 2PL!

Exercise 6.5
Multi-Granularity Locking (MGL)

TECHNISCHE UNIVERSITÄT DARMSTADT

MGL Lock Compatibility Matrix:

	IS	IX	S	SIX	X
IS	+	+	+	+	-
IX	+	+	-	-	-
S	+	-	+	-	-
SIX	+	-	-	-	-
X	-	-	-	-	-

Intended operations might concern non-overlapping sets of descendant nodes; x-locks are already set; s-locks are not yet set and they won't be allowed if the same data items are involved.

All descendants are implicitly x-locked disallowing any s-locks to be set on them.

All descendants are implicitly s-locked disallowing any x-locks to be set on them.

flashyDB DVS

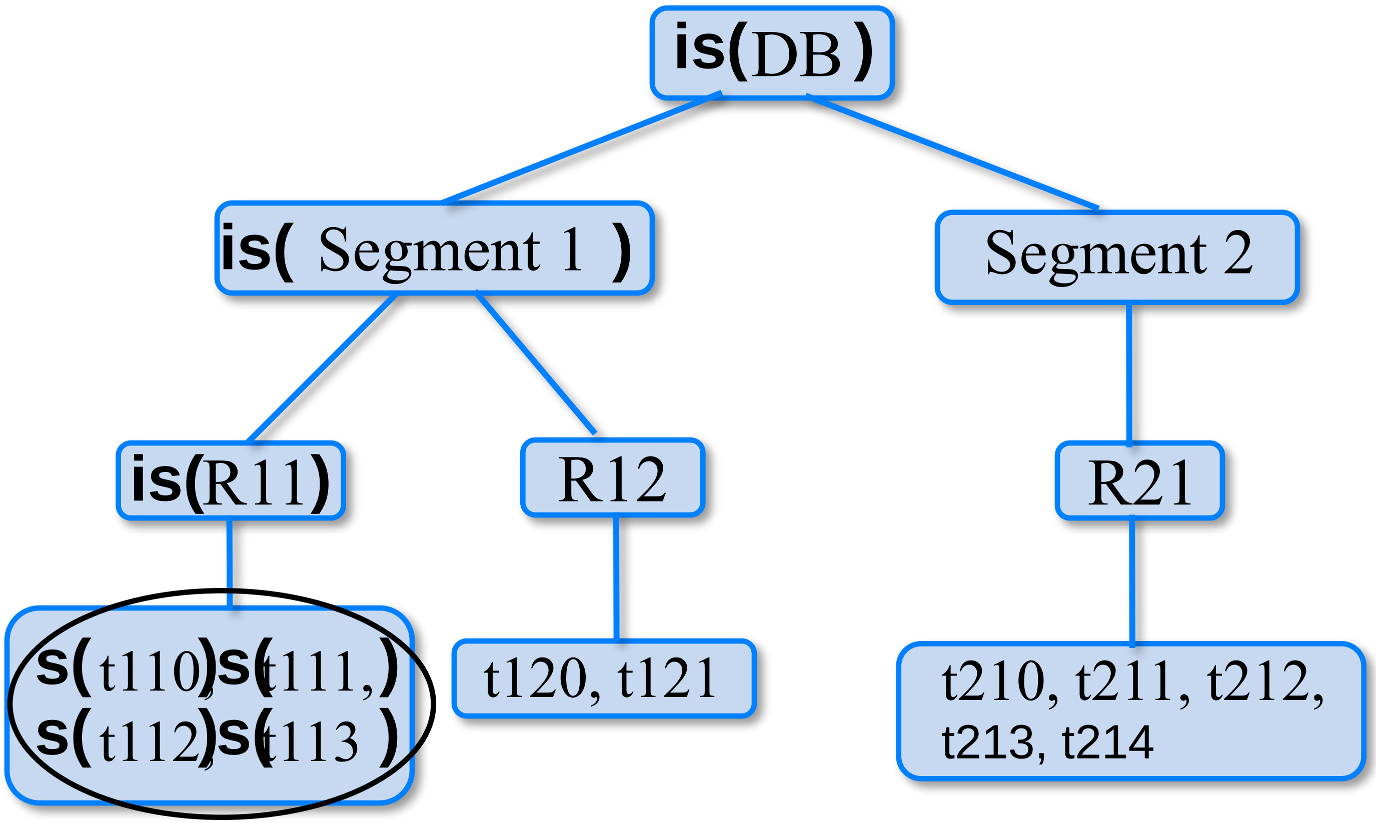
Exercise 6.5
Multi-Granularity Locking (MGL)

b) What locks need to be set if the following transactions are executed:

- T1 ***reads tuples*** t110, t111, t112, t113
- T2 ***reads tuples*** t210 and ***writes*** t121
- T3 ***reads tuples*** t120, t121 ***updates*** either t120 or t121
- T4 ***performs a nested-loop-join*** of R11 and R12.

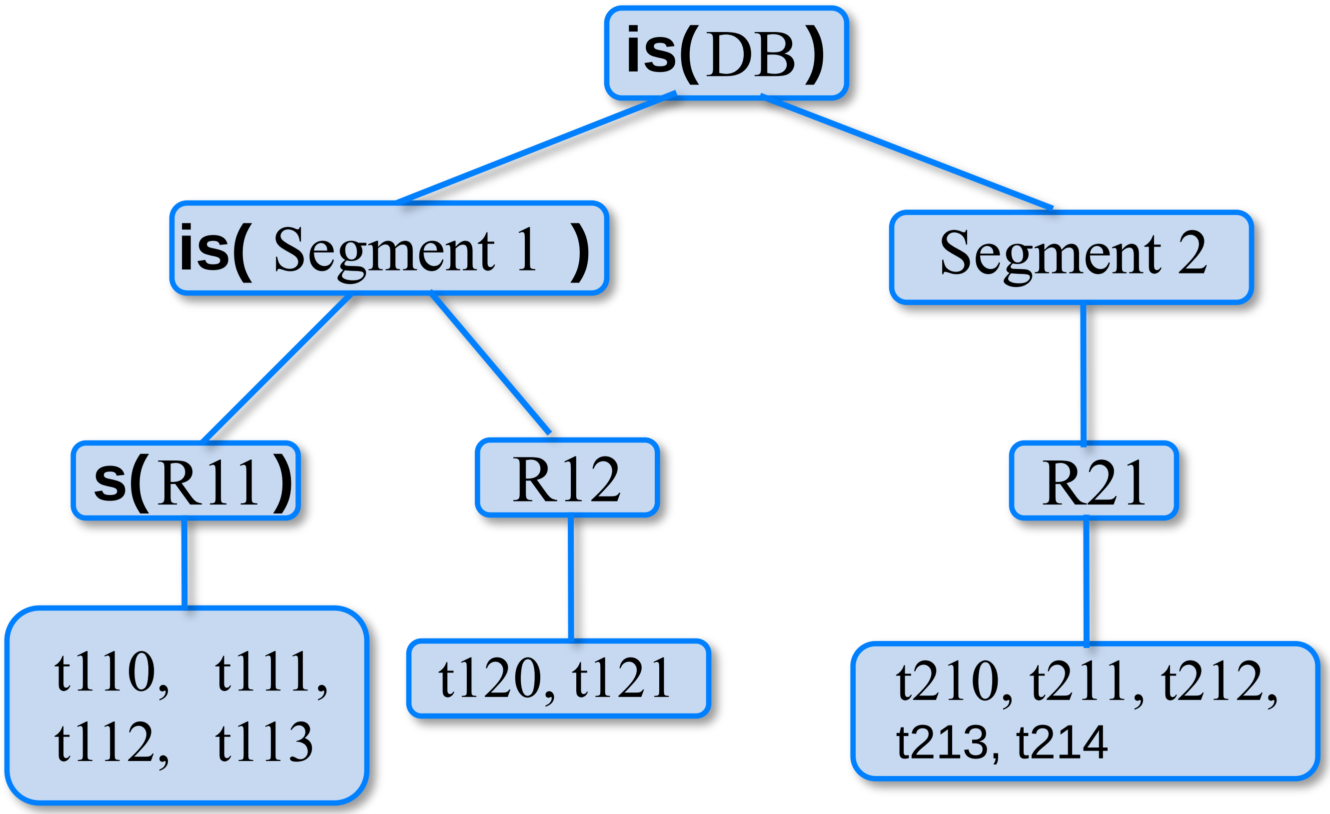
Exercise 6.5
Multi-Granularity Locking (MGL)

T1 ***reads tuples*** t110, t111, t112, t113



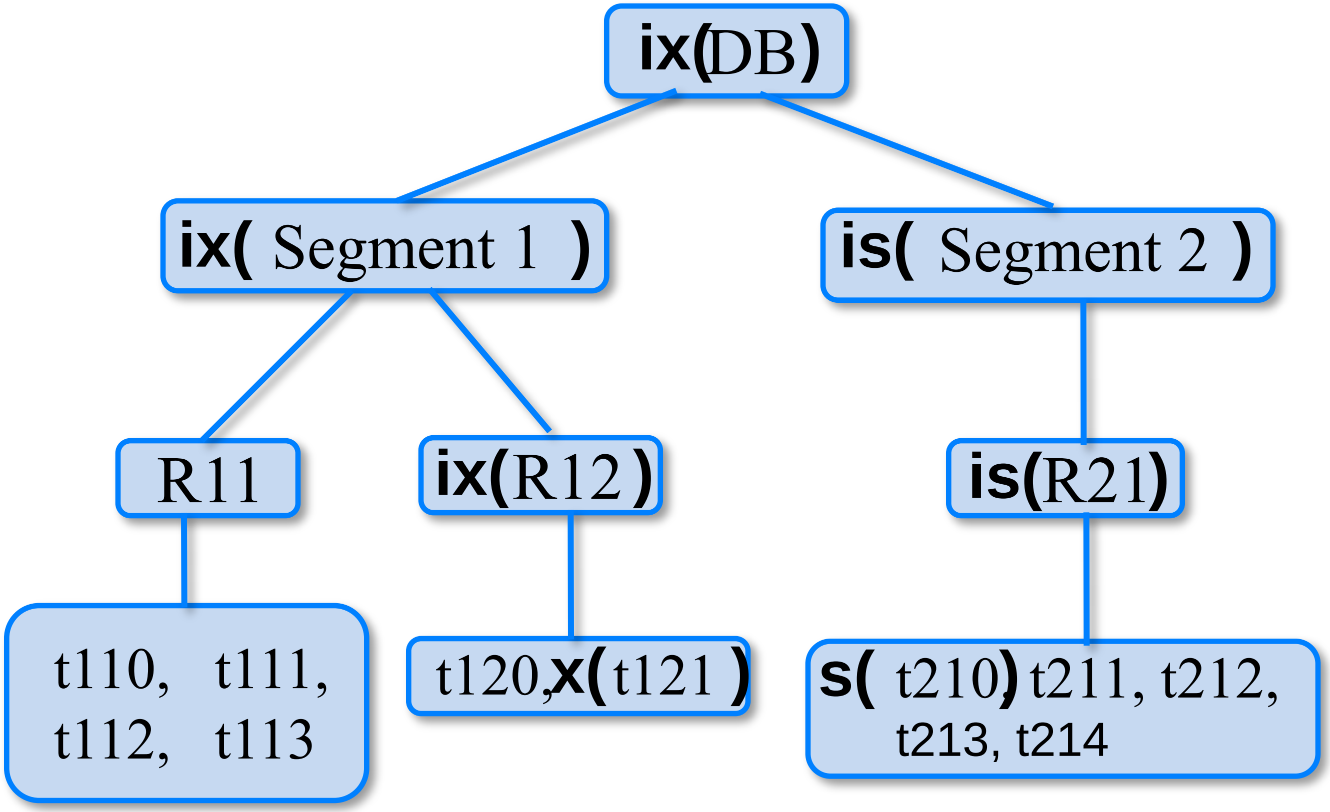
Exercise 6.5
Multi-Granularity Locking (MGL)

Or better:



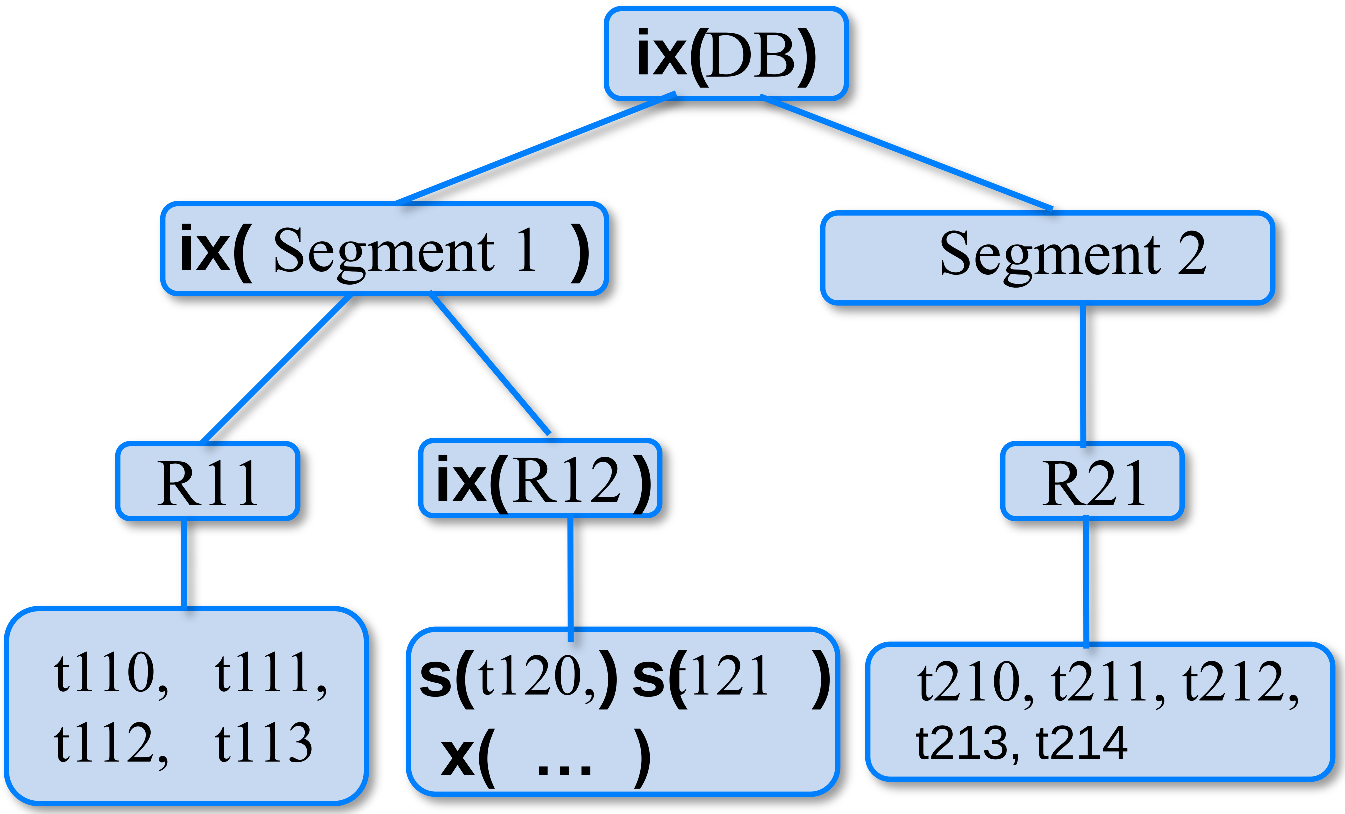
Exercise 6.5
Multi-Granularity Locking (MGL)

T2 ***reads tuples*** t210 and ***writes*** t121



Exercise 6.5
Multi-Granularity Locking (MGL)

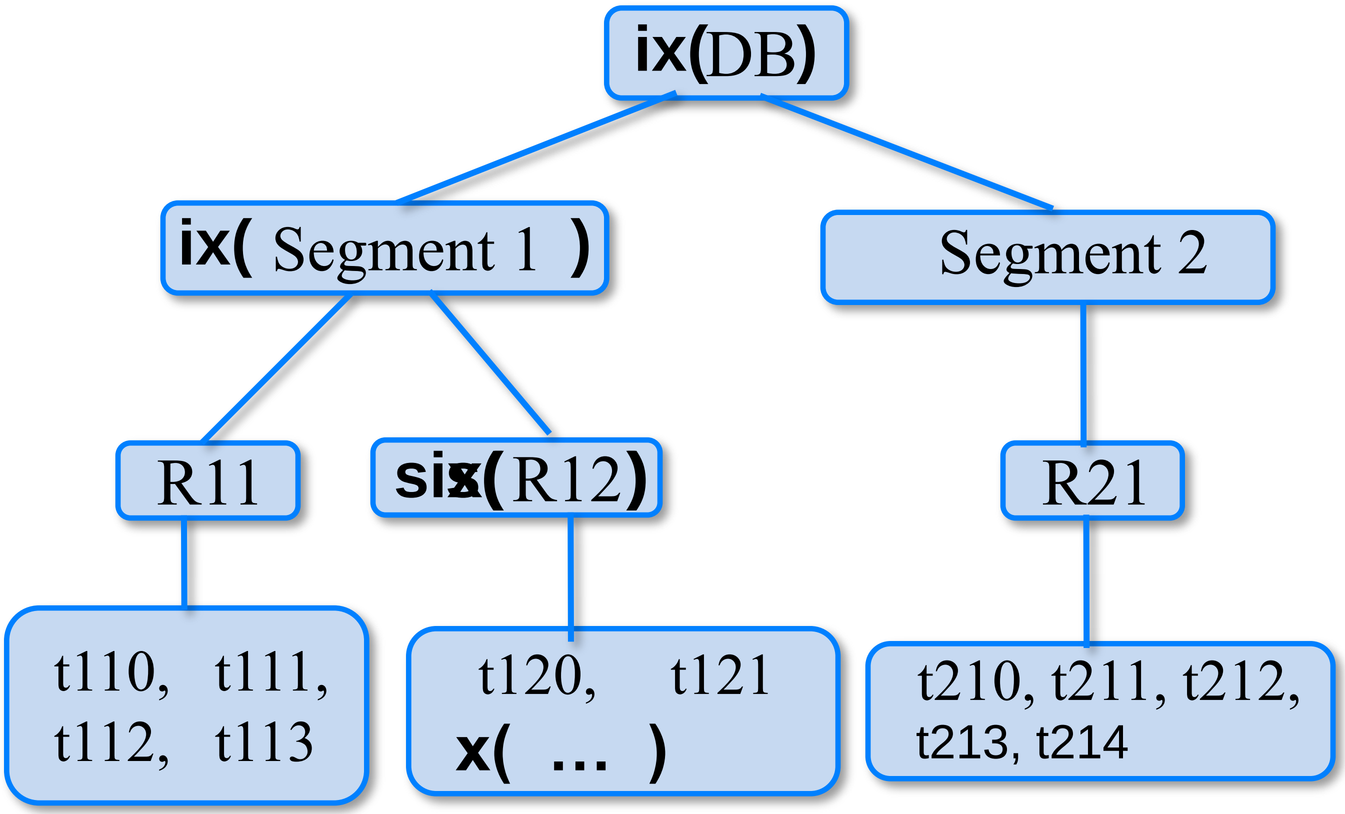
T3 ***reads tuples*** t120, t121 ***updates*** either t120 or t121



Exercise 6.5
Multi-Granularity Locking (MGL)

TECHNISCHE UNIVERSITÄT DARMSTADT

OR

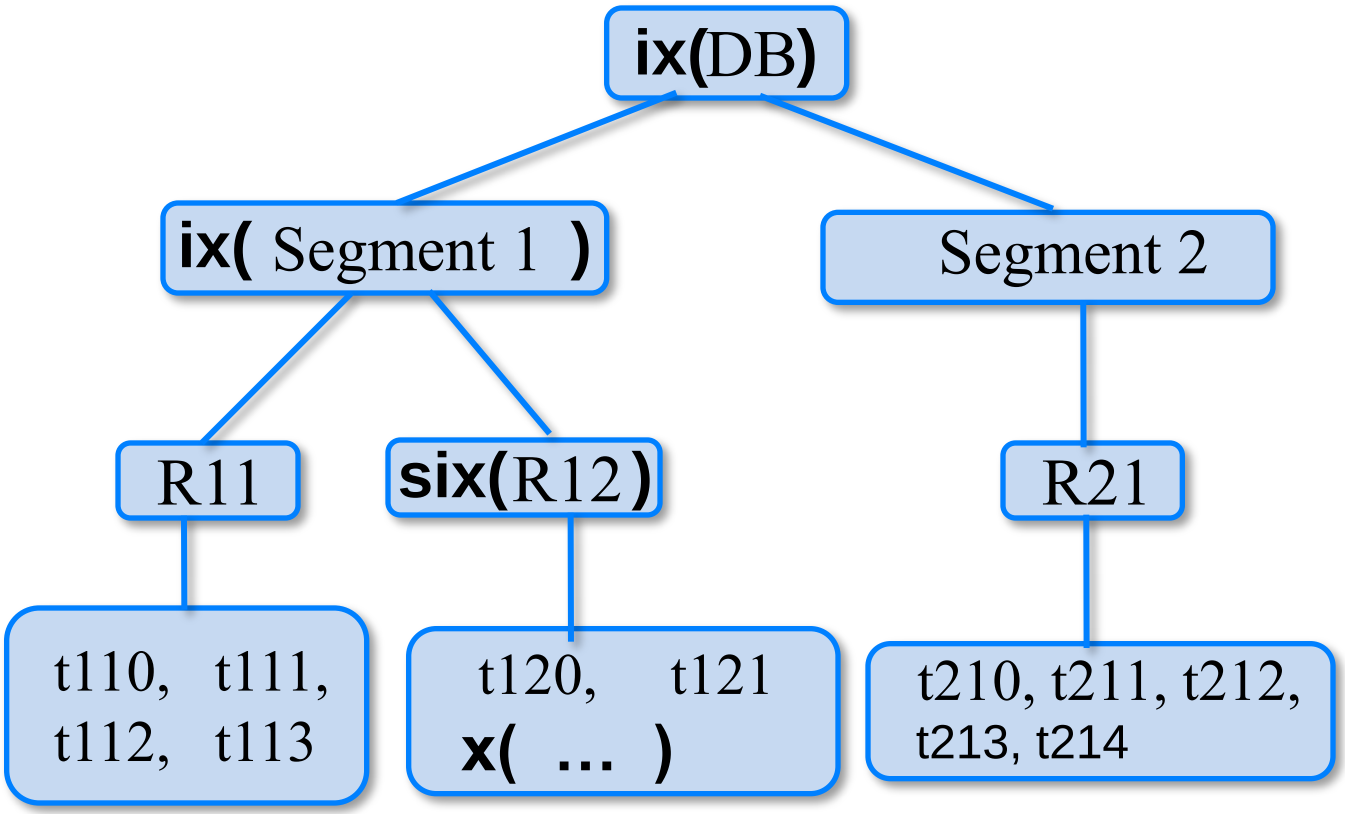


flashyDB DVS

Exercise 6.5
Multi-Granularity Locking (MGL)

TECHNISCHE UNIVERSITÄT DARMSTADT

OR



Exercise 6.5
Multi-Granularity Locking (MGL)

Note:

Different scenarios are possible depending on whether information about the transaction's ***operations*** is available in advance.

The more you know about the transaction the better you can choose.

→ Statistics and Analysis necessary

Exercise 6.5
Multi-Granularity Locking (MGL)

T4 ***performs a nested-loop-join*** of R11 and R12

A join of R11 and R12 **requires predicate locks *to avoid Phantom Anomalies.***

is(DB), is(Seg1), s(R11), s(R12)

Insert/delete would require an ix-lock on the relation(s).

Exercise 6.5
Multi-Granularity Locking (MGL)

c) To what extent does MGL provide ***a solution for the Phantom problem***?

Exercise 6.5
Multi-Granularity Locking (MGL)

How can **MGL** locks be used to **prevent Phantom Anomalies**?

Insert/Deletes are considered as write operations.

Two variants:

1. For reading all $\rightarrow$ set s(R)

   For insert $\rightarrow$ set ix(R)

   Allows concurrent Update/Inserts.

2. For reading all $\rightarrow$ set is(R) + s(txxx)

   For insert $\rightarrow$ set x(R)

Problem with 2.:

Not applicable to all predicate types. **Unless all nodes are s-locked, an update with ix(R), x(t) is still possible.** This update can produce a phantom with respect to the predicate. Apart from this, **no concurrent inserts/deletes are allowed.**

Exercise 6.4

Transaction Isolation Levels

Exercise 6.4
Transaction Isolation Levels

TECHNISCHE UNIVERSITÄT DARMSTADT

In the SQL-92 standard the 4 isolation levels were defined, which are often implemented in the following way using locking-based techniques:

Isolation Level	Row / Predicate Exclusive	Row Shared	Predicate Shared
1. Read Uncommitted	Long term (or not allowed)	None (latch)	None (latch)
2. Read Committed	Long term	Short term	Short term
3. Repeatable Read	Long term	Long term	Short term
4. Serializable	Long term	Long term	Long term

Isolation Level 1 is usually only used for read-only transactions. "Row-exclusive" and "row shared" correspond to write and read locks respectively. The table shows when locks are set and how long they are held:

"long term" means that locks are held until Commit, **"short term"** means that locks can be released earlier (usually after the operation completes). A **latch** is a short-duration lock set only for the duration of a physical I/O operation to ensure atomicity.

Exercise 6.4
Transaction Isolation Levels

TECHNISCHE UNIVERSITÄT DARMSTADT

a) Which isolation levels preclude (do not preclude) the **"Dirty-Read"**, **"Dirty-Write"** and **"Non-Repeatable Read"** phenomena respectively?

Latches do not conflict with the other types of locks, including "row exclusive" locks! This is the reason why ISO level 1 allows Dirty Reads.

2PL, if enforced completely is often too restrictive.

- Commercial DB systems offer isolation levels, so that users can achieve the best trade-off between concurrency and correctness.

- The isolation level of a transaction controls the extent to which the transaction is exposed to actions of concurrent transactions (ACID? → Isolation?)

Exercise 6.4
Transaction Isolation Levels

Dirty-Writes (WW-conflicts, overwrites):

P0: $w_1[x]...w_2[x]...((c_1$ or $a_1)$ and $(c_2$ or $a_2)$ in any order)

Since ***long term write locks are used in all ISO levels***, **Dirty Writes are precluded**. This is needed for the recovery system to enable the use of before-images for UNDO.

Exercise 6.4
Transaction Isolation Levels

TECHNISCHE UNIVERSITÄT DARMSTADT

Dirty-Reads (WR-conflicts):

P1: $w_1[x] \ldots r_2[x] \ldots ((c_1 \text{ or } a_1) \text{ and } (c_2 \text{ or } a_2) \text{ in any order})$

All levels except Read-Uncommitted prevent Dirty-Reads.
The benefit is that some unserializable schedules are eliminated and produced executions **are guaranteed to Avoid Cascading Aborts (ACA).**

Exercise 6.4
Transaction Isolation Levels

b) Which isolation levels preclude (do not preclude) the **"Inconsistent Analysis" anomaly**? Describe the problems caused by it using practical examples.

Inconsistent Analysis Anomaly:

An Inconsistent Analysis Anomaly occurs when a transaction is allowed to read an inconsistent database state.

This can happen if a transaction A reads some data before it is updated by another transaction B and some other data after it is updated by transaction B. Therefore, a part of the read data belongs to one consistent DB state, a part to another.

Exercise 6.4
Transaction Isolation Levels

Example:

Consider a history H involving a 50 € transfer from bank account A to B:

Variant 1:

$\mathbf{r_1[A]}$	$\mathbf{w_1[A]}$	$r_2[A]$	$r_2[B]$	c_2	$\mathbf{r_1[B]}$	$\mathbf{w_1[B]}$	$\mathbf{c_1}$
100	50	50	100		100	150	

T2 reads an **inconsistent state where the total balance is 150 instead of 200.** By allowing T2 to read uncommitted data, we allow it to read the partial results of other transactions (in this case T2). **This wouldn't occur if Dirty Reads were disallowed (ISO level >= 2).**

Exercise 6.4
Transaction Isolation Levels

TECHNISCHE UNIVERSITÄT DARMSTADT

Variant 2:

$r_2[A]$	**$r_1[A]$**	**$w_1[A]$**	**$r_1[B]$**	**$w_1[B]$**	**c_1**	$r_2[B]$	c_2
100	100	50	100	150		150	

T2 reads an inconsistent state where the total balance is 250 instead of 200. Again T2 reads the partial results of T1. **T2 reads only committed data**, but parts of it belong to one consistent DB state, parts of it to another. The state read by T2 *did not exist at one point in time!* This anomaly is sometimes called **Read Skew** and it **would not occur if Non-Repeatable Reads were disallowed** (ISO level >= 3).

Exercise 6.4
Transaction Isolation Levels

$\Rightarrow$ The Inconsistent Analysis Anomalies described before are avoided at the ***Repeatable Read*** ISO Level (ISO level >= 3).

Exercise 6.4
Transaction Isolation Levels

c) Which isolation levels preclude (do not preclude) the "**Lost Update**" and "**Write Skew**" anomalies? Describe the problems caused by them using practical examples

Exercise 6.4
Transaction Isolation Levels

Lost Update:

r_1[A] r_2[A] w_2[A] c_2 **w_1[A]** **c_1**

100 **100** **120** **130**

T2's update is lost!
This wouldn't occur if Non-Repeatable Reads were disallowed (ISO level >= 3).

Exercise 6.4
Transaction Isolation Levels

Write Skew:

$\mathbf{r_1[A]}$ $\mathbf{r_1[B]}$ $r_2[A]$ $r_2[B]$ $\mathbf{w_1[B]}$ $w_2[A]$ $\mathbf{c_1}$ c_2

(or more generally: $\mathbf{r_1[A]}$**...**$r_2[B]$**...**$\mathbf{w_1[B]}$**...**$w_2[A]$... {$\mathbf{c_1}$**,** c_2})

If there is a constraint involving A and B it might be violated leaving the DB in an inconsistent state. E.g. Acc. Balances are allowed to go negative as long as the sum of commonly held balances remains non-negative.

Write Skews are precluded if Non-Repeatable Reads are disallowed (ISO level >= 3).

Exercise 6.4
Transaction Isolation Levels

TECHNISCHE UNIVERSITÄT DARMSTADT

d) Which isolation levels preclude (do not preclude) the "**Phantom**" phenomenon? Describe the anomalies caused by it using practical examples.

Exercise 6.4
Transaction Isolation Levels

P3: $\mathbf{r_1[P]}$...w_2[y in P]...

P refers to a set of data items satisfying a given search condition (predicate).

Example:
Transaction T1 reads account balances of accounts belonging to a particular category, computes the sum and then compares it to the total balance stored in Y.

Exercise 6.4
Transaction Isolation Levels

Consistency constraint:

Insert new Record with CustID=***Z*****,** Amount=10

r1[A]	r1[B]	**new2[C]**	**r2[Y]**	**w2[Y]**	**c2**	r1[Y]	c1
10	20	10	30	40		40	

Write new Total

40 ≠ 30 ???
R1 Computes the wrong sum

Read Sum(Amount)
where CustID=***Z***

Exercise 6.4
Transaction Isolation Levels

TECHNISCHE UNIVERSITÄT DARMSTADT

One way to avoid phantoms:
When inserting/deleting wl(EOF). When reading rl(EOF). Poor performance.

Solution:
Use **Index locking** (or the more general predicate locking). Phantoms are avoided only in ISO level 4 - SERIALIZABLE.

Isolation Level 3 guarantees serializability for non-dynamic databases (fixed set of data items). For dynamic databases (inserts and deletes possible) a further restriction on the allowable executions is needed in order to enforce SR.

Exercise 6.4
Transaction Isolation Levels

TECHNISCHE UNIVERSITÄT DARMSTADT

Isolation Types characterized by Possible Anomalies Allowed:

ISO Level	Read Uncommitted	Read Committed	Repeatable Read	Serializable
Dirty Write	NO	NO	NO	NO
Dirty Read	YES	NO	NO	NO
Fuzzy Read	YES	YES	NO	NO
Phantom	YES	YES	YES	NO
Lost Update	YES	YES	NO	NO
Read Skew	YES	YES	NO	NO
Write Skew	YES	YES	NO	NO

Exercise 6.4
Transaction Isolation Levels

TECHNISCHE UNIVERSITÄT DARMSTADT

Note:

The original ANSI isolation levels were defined in terms of the phenomena (Dirty Reads, Fuzzy Reads, Phantoms) which transactions were allowed to experience. This was intended to allow non-lock based implementations of the SQL standard. Original definitions though, proved to be ambiguous and some refinements were needed in order to ensure correctness.

See the "**A Critique of ANSI SQL Isolation Levels**" paper.

Exercise 6.4
Transaction Isolation Levels

TECHNISCHE UNIVERSITÄT DARMSTADT

Note:

Some commercial DBMS products introduce additional levels of isolation such as: **Snapshot Isolation** (Microsoft Exchange), Cursor Stability, Read Consistency (Oracle) and others. These levels usually fall between Read Committed and Serializable in strength.

Transaction Chopping

Transaction Chopping

Definition:

Let T be a transaction program. A **chopping** of T is a decomposition of T into pieces $T_1, T_2, \ldots, T_k$ ($k \geq 1$) such that each database operation of T is performed in exactly one piece of T and the order of invocation is preserved.

Definition:

A chopping is rollback-safe, if all rollback-commands are located in the first piece (T_1).

Transaction Chopping

TECHNISCHE UNIVERSITÄT DARMSTADT

Rules:

1. Order is preserved.
2. If a piece resulting from chopping is aborted due to deadlock ➔ repeat until it commits.
3. After a rollback of the first piece no other piece may execute.

(Rollback-safe criteria)

Transaction Chopping

Given a set of transactions and a chopping construct a graph C(T) such that

- The nodes of C(T) are the transaction pieces ocurring in the chopping
- Let p and q be 2 pieces from 2 different transactions. If p and q contain operations that are in conflict, C(T) contains an undirected edge between p and q labeled “C” (conflict)
- If p and p’ are pieces from the same transaction C(T) contains an edge labeled “S” (sibling)

Chopping graph may have cycles involving s or c edges.

An sc cycle is a cycle that contains at least one c and at least one s edge

Transaction Chopping

TECHNISCHE UNIVERSITÄT DARMSTADT

Theorem:

A chopping is correct if the associated chopping graph does not contain an sc cycle

Transaction Chopping

Given:

$T1 = r_1(A1)w_1(A1)r_1(B1)w_1(B1)$

$T2 = r_2(A3)w_2(A3)r_2(B1)w_2(B1)$

$T3 = r_3(A4)w_3(A4)r_3(B2)w_3(B2)$

$T4 = r_4(A2)$

$T5 = r_5(A4)$

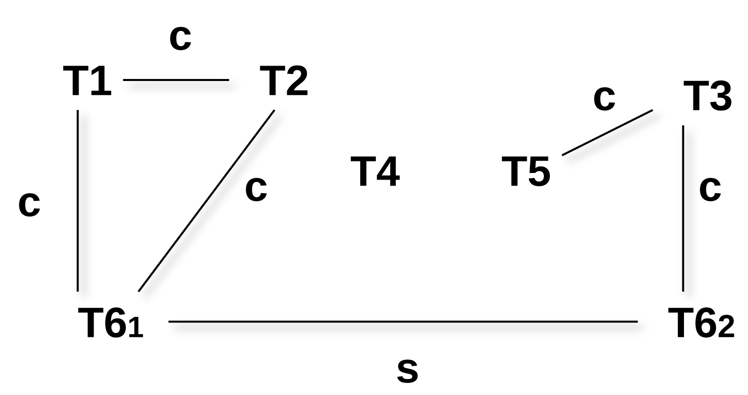
$T6 = r_6(A1)r_6(A2)r_6(A3)r_6(B1)r_6(A4)r_6(A5)r_6(B2)$

Chop T6 into

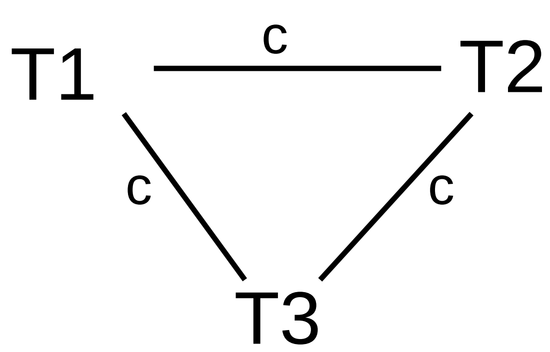
$T6_1 = r_{61}(A1)r_{61}(A2)r_{61}(A3)r_{61}(B1)$

$T6_2 = r_{62}(A4)r_{62}(A5)r_{62}(B2)$

No sc cycle, chopping is correct



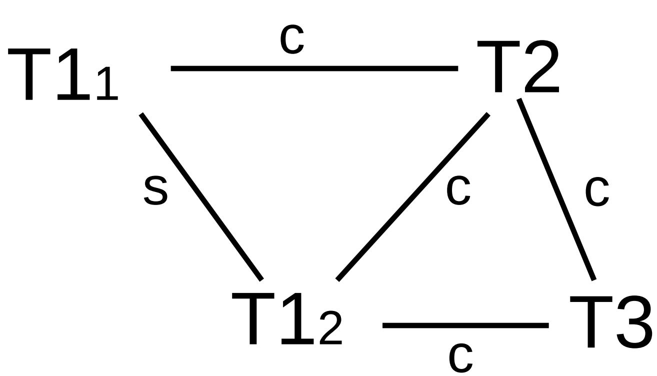
Transaction Chopping



➔No sc cycle➔correct

Conflict equivalent history, e.g.: T1 T2 T3

Transaction Chopping



$T1 = r_1(x)r_1(y)$

$T2 = w_2(x)w_2(y)$

➔ $T1_1 = r_1(x)$
$T1_2 = r_1(y)$

➔SC cycle➔incorrect

2PL: Perhaphs 2PL does not detect old conflict cycle.

➔Possible serial history: $T1_1$ T2 $T1_2$