Database Management Systems II

Robert Gottstein

gottstein@dvs.tu-darmstadt.de

The need for Concurrency Control and Recovery in Database Systems

Exercise 5.1
The need for Concurrency Control and Recovery in Database Systems

TECHNISCHE UNIVERSITÄT DARMSTADT

a) Explain the role of transactions in database systems.

b) Describe the ACID properties of transactions.

c) Which of the four ACID properties are enforced respectively by concurrency control and recovery techniques?

Exercise 5.1
The need for Concurrency Control and Recovery in Database Systems

a) Explain the role of transactions in database systems.

Definition Transaction (TX):

- A transaction is an **atomic process** that transforms a database from **one consistent state into another consistent state**
- **Physically** made of a sequence of operations bracketted by **BOT (Begin of Transaction)** and **EOT (End of Transaction)** operations.

Exercise 5.1
The need for Concurrency Control and Recovery in Database Systems

TECHNISCHE UNIVERSITÄT DARMSTADT

Concurrent execution of transactions is essential for good performance.

Two problems arise:

- **Concurrent access** to data may lead to data corruption.
- **System failures** interrupting running TXs may also lead to data corruption.

DBMSs automatically take care of these problems and thereby simplify application programming!
Application programs can be developed as though they were to be executed *strictly sequentially* in a *failure-free* system environment.

Exercise 5.1
The need for Concurrency Control and Recovery in Database Systems

b) Describe the ACID properties of transactions.

Atomicity	**either executes completely or has no effect at all**
Consistency	preserves the consistency of the database. **Leads from one consistent state to another**
Isolation	**does not interfere with concurrently executing transactions**
Durability	**once committed its updates to the database are made permanent,** i.e. are guaranteed to survive subsequent system failures.

Exercise 5.1
The need for Concurrency Control and Recovery in Database Systems

c) Which of the four ACID properties are enforced respectively by **concurrency control** and **recovery** techniques?

- CCtrl takes care of Isolation (and Consistency).
- Recovery enforces Atomicity and Durabililty.

Correctness Criteria in Transaction Processing

Exercise 5.2
Correctness Criteria in Transaction Processing

a) Why is a ***correctness criterion*** needed?
Give examples of incorrect concurrent executions.

b) What requirements does a correctness criterion need to satisfy?
What is understood by ***prefix-commit closeness***?
Why is this property important for recoverability?

c) What is understood by ***serializability***? Why are serializable histories considered as correct?

Exercise 5.2
Correctness Criteria in Transaction Processing

a) Why is a ***correctness criterion*** needed? Give examples of incorrect concurrent executions.

Example of interference:

We have a transaction ***Deposit***, which deposits money into the bank account of a client:

```
Transaction Deposit(acc#, amount)
{
   Start;
   temp := Read(A[acc#]);
   temp := temp + amount;
   Write(A[acc#], temp);
   Commit;
}
```

Exercise 5.2
Correctness Criteria in Transaction Processing

TECHNISCHE UNIVERSITÄT DARMSTADT

Suppose:

A[5] = 1000 and we execute ***"T1: Deposit(5, 100)"*** concurrently with ***"T2: Deposit(5, 100 000)"*** in the following manner:

`T1:Read`(A[5]) //A[5] = 1000 €
`T2:Read`(A[5]) //A[5] = 1000 €

`T2:Write`(A[5], 101000 €) // A[5] = 101000 €
`T2:Commit`

`T1:Write`(A[5], 1100 €) //A[5] = 1100 €
`T1:Commit`

The **result** is that A[5] contains **1100 € instead of 101 100 €**.
This is sometimes called a **lost update anomaly.**

Exercise 5.2
Correctness Criteria in Transaction Processing

TECHNISCHE UNIVERSITÄT DARMSTADT

Why is a correctness criterion needed?

A correctness criterion is needed to determine which histories should be allowed by the scheduler and which shouldn't.

Exercise 5.2
Correctness Criteria in Transaction Processing

b) What requirements does a correctness criterion need to satisfy?
What is understood by ***prefix-commit closeness***?
Why is this property important for recoverability?

Efficiently decidable (testable),
Scheduler should not take forever to determine whether what it has come up with is acceptible.

Allows a sufficiently large set of histories.
Larger set of allowable schedules
$\Rightarrow$ The more concurrency and thus better performance

Set of allowed histories must be <u>prefix-commit closed</u>.

Exercise 5.2
Correctness Criteria in Transaction Processing

TECHNISCHE UNIVERSITÄT DARMSTADT

Committed Projection
(history obtained from S by deleting all operations that do not belong to committed transactions)

Prefix commit-closeness (PCC):

Def: A property of histories $\mathcal{H}$ is **PCC** iff:

$$\forall\ H \in \mathcal{H} : \forall\ (S : \text{prefix}(H)) \Rightarrow C\ (S) \in \mathcal{H}$$

Any correctness criterion that accounts for system failures must induce a PCC set of histories.

This is because **any execution can be interrupted at any time** by a system failure and without PCC correctness might be compromised.

Exercise 5.2
Correctness Criteria in Transaction Processing

TECHNISCHE UNIVERSITÄT DARMSTADT

c) What is understood by ***serializability***? Why are serializable histories considered as correct?

Most correctness criteria are based on ***serializability*!**

Definition:

A history (execution) is **serializable** if it has the same **effect** as a **serial** history (execution) of the same transactions. By **effect** here, we mean both the **values read** by each transaction **and the final state** of the database.

Exercise 5.2
Correctness Criteria in Transaction Processing

Why are serializable histories considered as correct?

A transaction when executed on itself preserves the consistency of the DB. (Per Definition!)

$\Rightarrow$ **A serial execution preserves consistency and is corrrect.**

Serializable executions have the same effect as serial executions

$\Rightarrow$ Therefore we consider them as correct.

View Serializability (VSR) vs. Conflict Serializability (CSR)

Exercise 5.3
View Serializability(VSR) vs. Conflict Serializability(CSR)

TECHNISCHE UNIVERSITÄT DARMSTADT

a) What is the difference between **view** and **conflict equivalence** and **view** and **conflict serializability** respectively?

b) Given is the following history:
$H = r_1[a]\ r_3[b]\ r_2[a]\ w_1[a]\ w_1[c]\ c_1\ w_2[c]\ w_2[d]\ c_2\ w_3[c]\ c_3$

- Is the history conflict serializable? Why?
- Is the history view serializable? Why?

Exercise 5.3
View Serializability(VSR) vs. Conflict Serializability(CSR)

a) What is the difference between **view** and **conflict equivalence** and **view** and **conflict serializability** respectively?

Exercise 5.3 View Serializability(VSR) vs. Conflict Serializability(CSR)

TECHNISCHE UNIVERSITÄT DARMSTADT

Def: **View-Equivalence:**
Two histories are ***view-equivalent*** if they have the ***same read-from relationships*** and the ***same final writes***. View-equivalent histories obviously have the same **effect**.

Def: A history H is **View-Serializable (VSR)** if for any *prefix* H' of H, C (H') is *view equivalent* to a *serial history*.

Problem: To implement an efficient view equivalence test is **practically impossible**.

Solution: Find a stricter correctness criterion, which can be implemented efficiently.

Exercise 5.3 View Serializability(VSR) vs. Conflict Serializability(CSR)

Example:

$H=w_1[x]w_2[x]w_2[y]c_2w_1[y]c_1\mathbf{w_3[x]w_3[y]}c_3$

➔View equivalent to $T_1T_2T_3$ and $T_2T_1T_3$

➔VSR?

H'=w1[x]**w2[x]**w2[y]c2**w1[y]**c1

➔Not view equivalent to T_1T_2 or T_2T_1

Exercise 5.3
View Serializability(VSR) vs. Conflict Serializability(CSR)

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 5.3
View Serializability(VSR) vs. Conflict Serializability(CSR)

TECHNISCHE UNIVERSITÄT DARMSTADT

Since the effect of an execution depends ***only on the order*** in which conflicting operations are executed, it can be easily shown that

"conflict equivalent histories are also view equivalent"

The converse is not generally true - i.e. ***view equivalent histories are not always conflict equivalent.***

Exercise 5.3
View Serializability(VSR) vs. Conflict Serializability(CSR)

b) Given is the following history:
$H = r_1[a]\ r_3[b]\ r_2[a]\ w_1[a]\ w_1[c]\ c_1\ w_2[c]\ w_2[d]\ c_2\ w_3[c]\ c_3$

- Is the history conflict serializable? Why?
- Is the history view serializable? Why?

→ next Page – convenient way to solve such problems

Exercise 5.3
View Serializability(VSR) vs. Conflict Serializability(CSR)

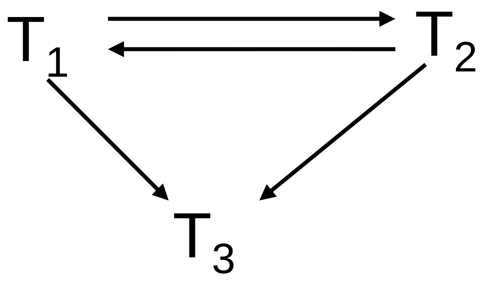
Conflict serializable?

$H = r_1[a]\ \mathbf{r_3[b]}\ \underline{r_2[a]}\ w_1[a]\ w_1[c]\ c_1\ \underline{w_2[c]}\ \underline{w_2[d]}\ \underline{c_2}\ \mathbf{w_3[c]\ c_3}$

1:	$r_1[a]$			$w_1[a]\ w_1[c]\ c_1$		
2:			$r_2[a]$		$w_2[c]\ w_2[d]\ c_2$	
3:		$r_3[b]$				$w_3[c]\ c_3$

Conflicts:

$r_2[a] \rightarrow w_1[a]$
$w_1[c] \rightarrow w_2[c]$
$w_1[c] \rightarrow w_3[c]$
$w_2[c] \rightarrow w_3[c]$



➔ Not CSR!

Exercise 5.3
View Serializability(VSR) vs. Conflict Serializability(CSR)

View-Serializable?

H = $r_1[a]$ $\mathbf{r_3[b]}$ $\underline{r_2[a]}$ $w_1[a]$ $w_1[c]$ c_1 $\underline{w_2[c]}$ $\underline{w_2[d]}$ $\underline{c_2}$ $\mathbf{w_3[c]}$ $\mathbf{c_3}$

1:	$r_1[a]$			$\mathbf{w_1[a]}$ $w_1[c]$ c_1	
2:			$r_2[a]$		$w_2[c]$ $\mathbf{w_2[d]}$ c_2
3:		$r_3[b]$			$\mathbf{w_3[c]}$ c_3

Final writes

<u>View-Serializability:</u>

For **every** C(prefix(H)) a serial history must exist that has:

1). the same final writes

2). the same read-from relationships

Exercise 5.3
View Serializability(VSR) vs. Conflict Serializability(CSR)

1: $r_1[a]$ $\mathbf{w_1[a]}$ $w_1[c]$ c_1

2: $r_2[a]$ $w_2[c]$ $\mathbf{w_2[d]}$ c_2

3: $r_3[b]$ $\mathbf{w_3[c]}$ c_3

(1). $T_1, T_2, T_3 \in C$ (prefix)

(i). final writes - $w_1[a]$, $w_2[d]$, $w_3[c]$

(ii). read-from relationships (introduce initializing trans. T_0)

- All T_1, T_2 and T_3 read from T_0

A view-equivalent serial history must satisfy:

(i.) $\leftrightarrow$ *T_3 is last*

(ii.) $\leftrightarrow$ *T_2 is before T_1*, so that T_2 reads 'a' from T_0

$$\Rightarrow H \cong T_2\, T_1\, T_3$$

Exercise 5.3 View Serializability(VSR) vs. Conflict Serializability(CSR)

1:	$r_1[a]$	$\mathbf{w_1[a]}$ $w_1[c]$ c_1	
2:	$r_2[a]$		$\mathbf{w_2[c]}$ $\mathbf{w_2[d]}$ c_2

(2). T_1, T_2 ∈ C (prefix)

(i). final writes - $w_1[a]$, $w_2[c]$, $w_2[d]$

(ii).read-from relationships

Both T_1 and T_2 read from T_0

A view-equivalent serial history must satisfy:

- *T_2 must be after T_1* to have a final write $w_2[c]$
- *T_2 must be before T_1* so that T_2 reads from T_0

➔This is impossible

➔ ***View Equivalent but Not View-Serializable***

TECHNISCHE UNIVERSITÄT DARMSTADT

Order Preservation of Transactions

Exercise 5.4 Order Preservation of Transactions

Order preservation of transactions:

Definition:

T_i is in H *entirely before* T_j, if
for all $o \in op(T_j)$ we have $c_i <_H o$.

Give an example of a CSR history H with the following properties:

- Transaction **T_1 is entirely before T_2** in H.
- In ***every serial history*** H', conflict-equivalent to H, **T_2 is entirely before T_1.**

Exercise 5.4
Order Preservation of Transactions

Idea: Construct a history where T1 and T2 don't have any direct conflicts, but conflict indirectly through a third transaction T3.

1. $\mathbf{r_1[x]\ w_1[x]\ c_1}$
2. $\underline{r_2[y]\ w_2[y]\ c_2}$
3. $r_3[x]$ $w_3[y]\ c_3$

H = $r_3[x]$ $\mathbf{r_1[x]\ w_1[x]\ c_1}$ $\underline{r_2[y]\ w_2[y]\ c_2}$ $w_3[y]\ c_3$

$T_2 \rightarrow T_3 \rightarrow T_1$

Note: we must not have any conflicts between T_1 and T_2!

Exercise 5.4
Order Preservation of Transactions

Conclusion:

The order in which transactions are executed is not guaranteed by CSR.

The order is not significant as far as correctness is concerned.

→ Very important conclusion!

Recoverability

Exercise 5.5
Recoverability

a) To what extent are the classes (resp. their characteristics):
   RC (recoverable),
   ACA (Avoids Cascading Aborts) and
   ST (Strict)
   of practical significance?
   Explain the problematic by use of examples.

b) Show that **CSR** and **ST** are incomparable sets.

Exercise 5.5 Recoverability

Definition: T_i reads 'x' from T_k in H if:

(i). $w_k[x] < r_i[x]$

(ii). $a_k < r_i[x]$

(iii). $\forall\, w_m[x] : (w_k[x] < w_m[x] < r_i[x]) \mid a_m < r_i[x]$

- A history is ***recoverable (RC)*** if
  T_i reads from T_k $(i \neq k)$ in H and c_i is in H ➔ $c_k < c_i$
- A history is RC if every transaction in H commits after the transactions from which it read

Exercise 5.5
Recoverability

Recoverable (RC):

***RC Rule**:* T_i reads from T_k ➔ $C_k < C_i$

***Disallows**:* $r_1[x]$ $w_1[x]$ $\mathbf{r_2[x]}$ $\mathbf{w_2[x]}$ $\mathbf{c_2}$ /...Failure.../ a_1

- Requires that: ***A transaction commits only after all transactions from which it has read commit***.
- Enables the recovery manager *to undo the effects of aborted transactions* after a system failure.
- **Any scheduler should enforce this property** so that recovery is always possible in case of a failure.

Exercise 5.5 Recoverability

Avoids Cascading Aborts (ACA):

ACA Rule: T_i reads from $T_k \Rightarrow c_k < r_i$

Disallows: $r_1[x]\ w_1[x]\ r_2[x]\ w_2[y]\ a_1\ (\Rightarrow a_2)$

- Transaction allowed to **read only from committed transactions**. In other words **dirty reads are disallowed** and no uncommitted data is read.

- This means that before a transaction is committed it cannot affect other transactions. Hence in case of abort, transaction's effects can easily be undone simply by undoing its writes.

Exercise 5.5
Recoverability

ACA is optional, but recommended for the following reasons:

- *avoids the significant bookkeeping* needed otherwise

- avoids the potentially many aborts after a failure and simplifies the abort operation.

Exercise 5.5 Recoverability

Strict (ST)

ST Rule: $W_j[x] < O_i[x] \rightarrow (A_j < O_i[x]) \vee (C_j < O_i[x])$

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.

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

Exercise 5.5
Recoverability

Note:
RC, ACA, ST are ***increasingly restrictive properties***
($\mathcal{H}_{\textbf{[RC]}} \supset \mathcal{H}_{\textbf{[ACA]}} \supset \mathcal{H}_{\textbf{[ST]}}$).

!!!***None of which is comparable to SR***!!!

It can be easily shown that
RC, ACA, ST and SR are prefix-commit closed properties.

Appendix:
(Recovery 5.5b)
Serializability and Recoverability

Exercise 5.6
Summary Slide

CSR – Conflict Serializable – "*C(H) conflict equivalent to serial History: Two histories are conflict equivalent if they have the same operations and they order conflicting operations in the same order*"

PCC – Prefix Commit Closedness

"H is PCC with respect to a property when all possible prefix histories of normally terminated transactions have the same property (e.g. SR) as the full history"

VSR – View Serializable

"History is VSR if for any prefix H' of H C(H') is view equivalent to some serial history"

C(H) – The Committed Projection of History H

Exercise 5.6
Summary Slide

ACA – Avoid Cascading Aborts

"A History is ACA if transactions only read from committed transactions"

RC – Recoverable

"A History is RC if every transaction in H commits after the transactions from which it read"

ST – Strict

"A transaction only may overwrite data when the transaction that wrote them last finished (c or a)"

Exercise 5.5 Recoverability

b) Show that **CSR** and **ST** are incomparable sets with respect to '$\subseteq$'.

must show:

(i). $(ST \cap CSR) \neq \varnothing$
(ii). $CSR \nsubseteq ST$
(iii). $ST \nsubseteq CSR$

Exercise 5.5
Recoverability

(i). (ST ∩ CSR) ≠ ∅

➔ ∃ H: (H   ST and H∈CSR)?

$r_1[x]\ w_1[x]\ c_1\ r_2[x]\ w_2[x]\ c_2$ (any serial history)

(ii). CSR ⊈ ST

➔ ∃ H: (H∈ CSR, but H ∉ ST)?

e.g. $H = w_1[x]\ w_2[x]\ c_1 c_2$

Exercise 5.5
Recoverability

(iii). ST $\nsubseteq$ CSR

➔$\exists$ H: (H$\in$ ST, but H $\notin$ CSR)?

Idea:

ST takes care of WR (Dirty Read) and WW (Overwrite) conflicts, but not of RW (Unrepeatable reads) conflicts

➔we can have a cycle caused only by RW conflicts:

$$H = r_1[x]\ \underbrace{w_2[x]}_{T_1 \rightarrow T_2}\ r_2[y]\ \underbrace{w_1[y]}_{T_2 \rightarrow T_1}\ c_1\ c_2$$

Exercise 5.6
Serializabiliy and Recoverability

Give examples of histories with the following properties:

a)

$$H1 = ? \quad : \quad H1 \in \left(CSR \cap RC\right) / ACA$$

b)

$$H2 = ? \quad : \quad H2 \in \left(VSR \cap ACA\right) / \left(CSR \cup ST\right)$$

c)

$$H3 = ? \quad : \quad H3 \in ST / VSR$$

Exercise 5.6
Serializabiliy and Recoverability

not ACA: ..... $w_1[y]\ r_2[y]$ .....

a) $H1 = ?\quad:\quad H1 \in (CSR \cap RC) / ACA$

RC: ....//............ $c_1\ c_2$ (c_1 must be first)

CSR: ....//............ $c_1\ c_2$

➔ $H1 = w_1[y]\ r_2[y]\ c_1\ c_2$

Exercise 5.6
Serializabiliy and Recoverability

b) $H2 = ? \quad : \quad H2 \in \left(VSR \cap ACA\right) / \left(CSR \cup ST\right)$

use no read operations ➔ACA

not ST: ..... $w_1[x]$ $w_2[x]$ .....

not CSR: ..... " ..... $w_2[y]$ $w_1[y]$

VSR: final writes + PCC (no read-from relationships)

H2 = $\mathbf{w_1[x]}$ $w_2[x]$ $w_2[y]$ $\mathbf{w_1[y]}$ c_2 $\underline{w_3[y] w_3[x] c_3}$ $\mathbf{w_1[z]}$ $\mathbf{c_1}$

Exercise 5.6
Serializabiliy and Recoverability

H2 = $\mathbf{w_1[x]}$ $w_2[x]$ $w_2[y]$ $\mathbf{w_1[y]}$ c_2 $\underline{w_3[y]w_3[x]c_3}$ $\mathbf{w_1[z]}$ $\mathbf{c_1}$

View equiv. to:

$T_1T_2T_3 \cong$ $\mathbf{w_1[x]w_1[y]}$$\boldsymbol{w_1[z]}$ $\mathbf{c_1}$ $w_2[x]w_2[y]c_2$ $\underline{w_3[y]w_3[x]}\underline{c_3}$

Final writes: *$w_1[z]$ $w_3[x]$ $w_3[y]$*

PCC ?

i). only T2, T3 in C(...)

C(...) = w2[x] w2[y] c2 <u>w3[y] w3[x] c3</u>

final writes: w3[y] w3[x] ➔ ≅T2 T3

ii). only T2 in C(...) ➔ trivial

Note: w1[z] in the end (to ensure PCC)!

Exercise 5.6
Serializabiliy and Recoverability

c) $H3 = ? \quad : \quad H3 \in ST / VSR$

Idea:
make final writes not correspond to read from relationships

H3=**r1[a]** r2[a] **w1[a] w1[c] c1** w2[c] w2[d] c2

not VSR: T2 must be after T1 to have final write w2[c],
but then T2 would read from T1 and not T0

ST: obviously