Kapitel 9: Transaktionsverwaltung

Transaktion

Unter einer *Transaktion* verstehen wir eine Folge von Datenbankzugriffen, die logisch zusammengehören. Eine Transaktion kann durch die Ausführung einer SQL-Anweisung definiert sein oder durch eine Ausführung eines Programms mit Datenbankzugriffen (Prozess).

ACID-Eigenschaften

Wünschenswerte Eigenschaften von Transaktionen:

- **A**tomicity: Alles-Oder-Nichts-Prinzip
- **C**onsistency: Integrität wird bewahrt
- **I**solation: keine unerwünschten Abhängigkeiten bei Parallelität
- **D**urability: Daten gehen nicht verloren

Kapitel 9: Transaktionsverwaltung

Transaktion

Unter einer *Transaktion* verstehen wir eine Folge von Datenbankzugriffen, die logisch zusammengehören. Eine Transaktion kann durch die Ausführung einer SQL-Anweisung definiert sein oder durch eine Ausführung eines Programms mit Datenbankzugriffen (Prozess).

ACID-Eigenschaften

Wünschenswerte Eigenschaften von Transaktionen:

- **A**tomicity: Alles-Oder-Nichts-Prinzip
- **C**onsistency: Integrität wird bewahrt
- **I**solation: keine unerwünschten Abhängigkeiten bei Parallelität
- **D**urability: Daten gehen nicht verloren

Aufgaben einer Transaktionsverwaltung

- Umfassen diejenigen Komponenten eines Datenbankmanagementsystems, deren Aufgabe die Gewährleistung der Atomizität, Isolation und Dauerhaftigkeit der Transaktionen ist.

(1) **Mehrbenutzerkontrolle**: Isolation der einzelnen Transaktionen.

(2) **Fehlerbehandlung**: Atomizität und Dauerhaftigkeit einer Transaktion.

9.1 Grundlagen

Transaktionen

- Eine Datenbank sei gegeben als eine Menge von Objekten.

  logische Größen: Relation, Tupel
  physische Größen: Blöcke, Seiten

- Eine Transaktion T greift lesend und schreibend auf die Objekte der Datenbank zu (*Lese-* und *Schreiboperationen*).
  Wir repräsentieren sie durch ihre Folge von Lese- und Schreiboperationen:
  - Bei Ausführung einer Leseoperation zu einem Datenbankobjekt A (RA), wird der Wert des Objektes A aus der Datenbank in den lokalen Arbeitsbereich der Transaktion übertragen.
  - Bei Ausführung einer Schreiboperation zu A (WA), wird der Wert des Objektes A aus dem lokalen Arbeitsbereich der Transaktion in die Datenbank übertragen.
- Lese- und Schreiboperationen betrachten wir als atomar.

9.1 Grundlagen

Transaktionen

- Eine Datenbank sei gegeben als eine Menge von Objekten.

  logische Größen: Relation, Tupel
  physische Größen: Blöcke, Seiten

- Eine Transaktion T greift lesend und schreibend auf die Objekte der Datenbank zu (*Lese*- und *Schreiboperationen*).
  Wir repräsentieren sie durch ihre Folge von Lese- und Schreiboperationen:
  - Bei Ausführung einer Leseoperation zu einem Datenbankobjekt A (RA), wird der Wert des Objektes A aus der Datenbank in den lokalen Arbeitsbereich der Transaktion übertragen.
  - Bei Ausführung einer Schreiboperation zu A (WA), wird der Wert des Objektes A aus dem lokalen Arbeitsbereich der Transaktion in die Datenbank übertragen.
- Lese- und Schreiboperationen betrachten wir als atomar.

Schedule

- Sei $\mathcal{T} = \{T_1, \ldots, T_n\}$ eine Menge von Transaktionen:

$$\mathcal{T} = \{T_1, T_2, T_3\}$$

- Die Folge der Lese- und Schreiboperationen einer Transaktion $T_i \in \mathcal{T}$ bezeichnen wir als ihre *Historie* h_i:

$$\begin{aligned} T_1 &= R_1A\ W_1A\ R_1B\ W_1B \\ T_2 &= R_2A\ W_2A\ R_2B\ W_2B \\ T_3 &= R_3A\ W_3B \end{aligned}$$

- Einen möglicherweise verzahnten Ablauf der Transaktionen aus $\mathcal{T}$ nennen wir einen *Schedule* S zu $\mathcal{T}$:

$$S = R_1A\ W_1A\ R_3A\ R_1B\ W_1B\ R_2A\ W_2A\ W_3B\ R_2B\ W_2B$$

Schedule

- Sei $\mathcal{T} = \{T_1, \ldots, T_n\}$ eine Menge von Transaktionen:

$$\mathcal{T} = \{T_1, T_2, T_3\}$$

- Die Folge der Lese- und Schreiboperationen einer Transaktion $T_i \in \mathcal{T}$ bezeichnen wir als ihre *Historie* h_i:

$$\begin{aligned} T_1 &= R_1A\ W_1A\ R_1B\ W_1B \\ T_2 &= R_2A\ W_2A\ R_2B\ W_2B \\ T_3 &= R_3A\ W_3B \end{aligned}$$

- Einen möglicherweise verzahnten Ablauf der Transaktionen aus $\mathcal{T}$ nennen wir einen *Schedule* S zu $\mathcal{T}$:

$$S = R_1A\ W_1A\ R_3A\ R_1B\ W_1B\ R_2A\ W_2A\ W_3B\ R_2B\ W_2B$$

- Ein Schedule ist eine Folge von Lese- und Schreiboperationen der einzelnen Transaktionen aus $\mathcal{T}$.
- Die relative Reihenfolge der Operationen einer Transaktion T_i in einem Schedule S entspricht der Reihenfolge der Operationen in der zugehörigen Historie h_i von T_i.
- Ein *serieller* Schedule zu $\mathcal{T}$ ergibt sich als Konkatenation der Historien der einzelnen Transaktionen aus $\mathcal{T}$.

Beispiel

- Sei $\mathcal{T} = \{T_1, T_2, T_3\}$, mit
  $T_1 = R_1A\ W_1A\ R_1B\ W_1B$
  $T_2 = R_2A\ W_2A\ R_2B\ W_2B$
  $T_3 = R_3A\ W_3B$
- Es existieren sechs unterschiedliche serielle Schedule zu $\mathcal{T}$, z.B. $S_1 = h_1h_2h_3$, oder auch $S_2 = h_2h_3h_1$.
- Wir schreiben im Folgenden auch vereinfachend:
  $S_1 = T_1T_2T_3 \qquad S_2 = T_2T_3T_1$
- Beispiele für nicht serielle Schedule sind:

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

9.2 Mehrbenutzerkontrolle

Problematik

Seien $T_1 = R_1A\ W_1A$ und $T_2 = R_2A\ W_2A$ zwei Transaktionen, die beide dasselbe Objekt A lesen und schreiben. Nehmen wir, dass A ein Lagerkonto repräsentiert und T_1 den Bestand um 100 Einheiten erhöht und T_2 den Bestand um 50 verringert.
Zu Beginn betrage der Bestand 80 Einheiten.

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
$A = 130$	$A = 30$	$A = 180$	$A = 130$	$A = 180$	$A = 30$

Welche der sechs Schedule sind korrekt?

9.2.1 Serialisierbarkeit

Definition

Ein Schedule S heißt *serialisierbar* genau dann, wenn zu ihm ein äquivalenter serieller Schedule S' bestehend aus denselben Transaktionen existiert.

Definition

Zwei Schedule S und S' über derselben Transaktionsmenge heißen *äquivalent*, wenn für jeden Startzustand der Datenbank und jede Semantik der Transaktionen die folgenden beiden Bedingungen erfüllt sind:

- Die Transaktionen lesen in S und S' jeweils dieselben Werte.
- Desweiteren erzeugen S und S' dieselben Endzustände der Datenbank.

Formalisierung der Semantik einer Transaktion: *Herbrand-Semantik*

9.2.1 Serialisierbarkeit

Definition

Ein Schedule S heißt *serialisierbar* genau dann, wenn zu ihm ein äquivalenter serieller Schedule S' bestehend aus denselben Transaktionen existiert.

Definition

Zwei Schedule S und S' über derselben Transaktionsmenge heißen *äquivalent*, wenn für jeden Startzustand der Datenbank und jede Semantik der Transaktionen die folgenden beiden Bedingungen erfüllt sind:

- Die Transaktionen lesen in S und S' jeweils dieselben Werte.
- Desweiteren erzeugen S und S' dieselben Endzustände der Datenbank.

Formalisierung der Semantik einer Transaktion: *Herbrand-Semantik*

Beispiel

- Seien $T_1 = R_1A\ W_1A\ R_1B\ W_1B$ und $T_2 = R_2A\ W_2A\ R_2B\ W_2B$.
- Seien S_1 und S_2 Schedule wie folgt:

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

	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	$f_{T_1,B}(f_{T_2,A}(A_0), f_{T_2,B}(A_0, B_0))$

Beispiel

- Seien $T_1 = R_1A\ W_1A\ R_1B\ W_1B$ und $T_2 = R_2A\ W_2A\ R_2B\ W_2B$.
- Seien S_1 und S_2 Schedule wie folgt:

$$
\begin{aligned}
S_1 &= R_1A\ W_1A\ R_2A\ W_2A\ R_2B\ W_2B\ R_1B\ W_1B \\
S_2 &= R_1A\ W_1A\ R_2A\ W_2A\ R_1B\ W_1B\ R_2B\ W_2B
\end{aligned}
$$

	Schedule T_1T_2		Schedule T_2T_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	$f_{T_1,B}(f_{T_2,A}(A_0), f_{T_2,B}(A_0,B_0))$

S_1 ist nicht serialisierbar, S_2 ist serialisierbar.

Beispiel

- Seien $T_1 = R_1A\ W_1A\ R_1B\ W_1B$ und $T_2 = R_2A\ W_2A\ R_2B\ W_2B$.
- Seien S_1 und S_2 Schedule wie folgt:

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

	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	$f_{T_1,B}(f_{T_2,A}(A_0), f_{T_2,B}(A_0, B_0))$

S_1 ist nicht serialisierbar, S_2 ist serialisierbar.

Beispiel fortgesetzt

- Seien $T_1 = R_1A\ W_1A\ R_1B\ W_1B$ und $T_2 = R_2A\ W_2A\ R_2B\ W_2B$.

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

$S_1 = R_1A\ W_1A\ R_2A\ W_2A\ R_2B\ W_2B\ R_1B\ W_1B$	
R_1A	A_0
W_1A	$f_{T_1,A}(A_0)$
R_2A	$f_{T_1,A}(A_0)$
W_2A	$f_{T_2,A}(f_{T_1,A}(A_0))$
R_2B	B_0
W_2B	$f_{T_2,B}(f_{T_1,A}(A_0), B_0)$
R_1B	$f_{T_2,B}(f_{T_1,A}(A_0), B_0)$
W_1B	$f_{T_1,B}(A_0, f_{T_2,B}(f_{T_1,A}(A_0), B_0))$

Beispiel fortgesetzt

- Seien $T_1 = R_1A\ W_1A\ R_1B\ W_1B$ und $T_2 = R_2A\ W_2A\ R_2B\ W_2B$.

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

$S_1 = R_1A\ W_1A\ R_2A\ W_2A\ R_2B\ W_2B\ R_1B\ W_1B$	
R_1A	A_0
W_1A	$f_{T_1,A}(A_0)$
R_2A	$f_{T_1,A}(A_0)$
W_2A	$f_{T_2,A}(f_{T_1,A}(A_0))$
R_2B	B_0
W_2B	$f_{T_2,B}(f_{T_1,A}(A_0), B_0)$
R_1B	$f_{T_2,B}(f_{T_1,A}(A_0), B_0)$
W_1B	$f_{T_1,B}(A_0, f_{T_2,B}(f_{T_1,A}(A_0), B_0)))$

S_1 ist nicht serialisierbar, da es keinen seriellen Schedule S_1' zu T_1 und T_2 gibt, so dass T_1 in S und S' dieselben Werte liest.

Beispiel fortgesetzt

- Seien $T_1 = R_1A\ W_1A\ R_1B\ W_1B$ und $T_2 = R_2A\ W_2A\ R_2B\ W_2B$.

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

$S_1 = R_1A\ W_1A\ R_2A\ W_2A\ R_2B\ W_2B\ R_1B\ W_1B$	
R_1A	A_0
W_1A	$f_{T_1,A}(A_0)$
R_2A	$f_{T_1,A}(A_0)$
W_2A	$f_{T_2,A}(f_{T_1,A}(A_0))$
R_2B	B_0
W_2B	$f_{T_2,B}(f_{T_1,A}(A_0), B_0)$
R_1B	$f_{T_2,B}(f_{T_1,A}(A_0), B_0)$
W_1B	$f_{T_1,B}(A_0, f_{T_2,B}(f_{T_1,A}(A_0), B_0)))$

S_1 ist nicht serialisierbar, da es keinen seriellen Schedule S_1' zu T_1 und T_2 gibt, so dass T_1 in S und S' dieselben Werte liest.

augmentierter Schedule

- Sei T_0 eine Transaktion, die gerade zu jedem Objekt der Datenbank eine Schreiboperation enthält. T_0 erzeugt einen Startzustand der Datenbank.
- Sei T_∞ eine Transaktion, die zu jedem Objekt eine Leseoperation enthält. T_∞ liest den Endzustand der Datenbank.
- Sei S ein Schedule zu $\mathcal{T}$. Sei $\widehat{S} = T_0 \; S \; T_\infty$.

  $\widehat{S}$ nennen wir den *augmentierten* Schedule zu S.

augmentierter Schedule

- Sei T_0 eine Transaktion, die gerade zu jedem Objekt der Datenbank eine Schreiboperation enthält. T_0 erzeugt einen Startzustand der Datenbank.
- Sei T_∞ eine Transaktion, die zu jedem Objekt eine Leseoperation enthält. T_∞ liest den Endzustand der Datenbank.
- Sei S ein Schedule zu $\mathcal{T}$. Sei $\widehat{S} = T_0 \; S \; T_\infty$.

  $\widehat{S}$ nennen wir den *augmentierten* Schedule zu S.

Abhängigkeitsgraph

Der *Abhängigkeitsgraph* eines Schedule S ist ein gerichteter Graph $AG(S) = (V, E)$, wobei V die Menge aller Operationen in $\widehat{S}$ und E die Menge der Kanten gemäß den folgenden Bedingungen ($i \neq j$):

- $\widehat{S} = \ldots R_i B \ldots W_j A \ldots \;\Rightarrow\; R_i B \rightarrow W_j A \in E$
- $\widehat{S} = \ldots W_i A \ldots R_j A \ldots \;\Rightarrow\; W_i A \rightarrow R_j A \in E$,
  sofern zwischen $W_i A$ und $R_j A$ in $\widehat{S}$ keine weitere Schreiboperation zu A existiert.

Satz

Zwei Schedule S und S' einer gemeinsamen Menge von Transaktionen sind genau dann *äquivalent*, wenn $AG(\widehat{S}) = AG(\widehat{S'})$ gilt.

Abhängigkeitsgraph

Der *Abhängigkeitsgraph* eines Schedule S ist ein gerichteter Graph $AG(S) = (V, E)$, wobei V die Menge aller Operationen in $\widehat{S}$ und E die Menge der Kanten gemäß den folgenden Bedingungen $(i \neq j)$:

- $\widehat{S} = \dots R_iB \dots W_jA \dots \;\Rightarrow\; R_iB \rightarrow W_jA \in E$
- $\widehat{S} = \dots W_iA \dots R_jA \dots \;\Rightarrow\; W_iA \rightarrow R_jA \in E$,
  sofern zwischen W_iA und R_jA in $\widehat{S}$ keine weitere Schreiboperation zu A existiert.

Satz

Zwei Schedule S und S' einer gemeinsamen Menge von Transaktionen sind genau dann *äquivalent*, wenn $AG(\widehat{S}) = AG(\widehat{S'})$ gilt.

Konfliktgraph

Der *Konfliktgraph* eines Schedule S ist ein gerichteter Graph $KG(S) = (V, E)$, wobei V die Menge aller Transaktionen in $\widehat{S}$ und E die Menge der Kanten gemäß den folgenden Bedingungen ($i \neq j$):

- $\widehat{S} = \ldots W_iA \ldots R_jA \ldots \;\Rightarrow\; T_i \to T_j \in E$, sofern zwischen W_iA und R_jA in $\widehat{S}$ keine weitere Schreiboperation zu A existiert. *(WR-Konflikt)*
- $\widehat{S} = \ldots R_iA \ldots W_jA \ldots \;\Rightarrow\; T_i \to T_j \in E$, sofern zwischen R_iA und W_jA in $\widehat{S}$ keine weitere Schreiboperation zu A existiert. *(RW-Konflikt)*
- $\widehat{S} = \ldots W_iA \ldots W_jA \ldots \;\Rightarrow\; T_i \to T_j \in E$, sofern zwischen W_iA und W_jA in $\widehat{S}$ keine weitere Schreiboperation zu A existiert. *(WW-Konflikt)*

Satz und Definition

- Ein Schedule S ist serialisierbar, wenn $KG(S)$ zyklenfrei ist.
- Ein Schedule S heißt *konfliktserialisierbar* genau dann, wenn $KG(S)$ zyklenfrei ist.

Konfliktgraph

Der *Konfliktgraph* eines Schedule S ist ein gerichteter Graph $KG(S) = (V, E)$, wobei V die Menge aller Transaktionen in $\widehat{S}$ und E die Menge der Kanten gemäß den folgenden Bedingungen ($i \neq j$):

- $\widehat{S} = \ldots W_iA \ldots R_jA \ldots \;\Rightarrow\; T_i \rightarrow T_j \in E$, sofern zwischen W_iA und R_jA in $\widehat{S}$ keine weitere Schreiboperation zu A existiert. *(WR-Konflikt)*
- $\widehat{S} = \ldots R_iA \ldots W_jA \ldots \;\Rightarrow\; T_i \rightarrow T_j \in E$, sofern zwischen R_iA und W_jA in $\widehat{S}$ keine weitere Schreiboperation zu A existiert. *(RW-Konflikt)*
- $\widehat{S} = \ldots W_iA \ldots W_jA \ldots \;\Rightarrow\; T_i \rightarrow T_j \in E$, sofern zwischen W_iA und W_jA in $\widehat{S}$ keine weitere Schreiboperation zu A existiert. *(WW-Konflikt)*

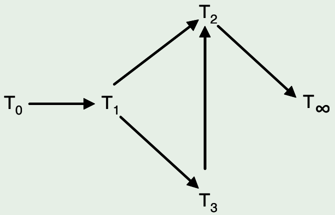
Satz und Definition

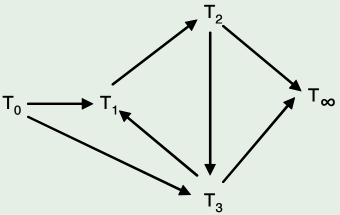
- Ein Schedule S ist serialisierbar, wenn $KG(S)$ zyklenfrei ist.
- Ein Schedule S heißt *konfliktserialisierbar* genau dann, wenn $KG(S)$ zyklenfrei ist.

Beispiel

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



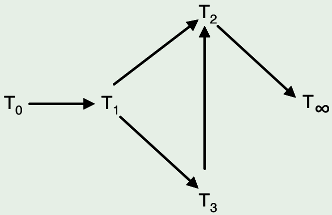


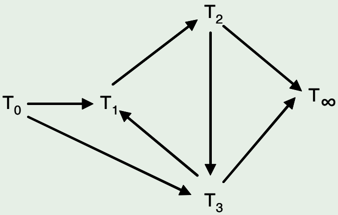
Im Folgenden betrachten wir - sofern nicht explizit unterschieden - nur noch Konflikt-Serialisierbarkeit und verwenden hierfür als Synonym den Begriff Serialisierbarkeit!

Beispiel

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





Im Folgenden betrachten wir - sofern nicht explizit unterschieden - nur noch Konflikt-Serialisierbarkeit und verwenden hierfür als Synonym den Begriff Serialisierbarkeit!

9.2.2 Sperrverfahren

- Bevor eine Transaktion lesend oder schreibend zu einem Objekt zugreifen darf, muss ihr ein entsprechendes Privileg gewährt werden.
- **Sperroperation** (*Lock*):
  - *Leseprivileg* $L^R A$
  - *Lese- und Schreibprivileg* LA
- **Freigabeoperation** (*Unlock*): UA bzw. $U^R A$
- *Sperrtabelle*
- *Kompatibilitätsmatrix*:

gehaltenes Privileg zu A:

angefordertes Privileg zu A:

	$L^R A$	LA
$L^R A$	J	N
LA	N	N

- *Livelock* und *Deadlock* können auftreten.

9.2.2 Sperrverfahren

- Bevor eine Transaktion lesend oder schreibend zu einem Objekt zugreifen darf, muss ihr ein entsprechendes Privileg gewährt werden.
- **Sperroperation** (*Lock*):
  - *Leseprivileg* $L^R A$
  - *Lese- und Schreibprivileg* LA
- **Freigabeoperation** (*Unlock*): UA bzw. $U^R A$
- *Sperrtabelle*
- *Kompatibilitätsmatrix*:

gehaltenes Privileg zu A:

angefordertes Privileg zu A:	$L^R A$	LA
$L^R A$	J	N
LA	N	N

- *Livelock* und *Deadlock* können auftreten.

Vermeidung von Livelocks und Deadlocks

- Vermeidung von Livelocks: *First-Come-First-Served* Strategie
- Vermeidung von Deadlocks:
  - Jede Transaktion bewirbt sich zu Beginn um alle benötigten Privilegien auf einmal (in einer atomaren Operation).
  - Auf den Objekten wird eine lineare Ordnung definiert. Die Transaktionen fordern ihre jeweiligen Privilegien gemäß dieser Ordnung an.
- **Wartegraph**:
  Ein Wartegraph hat eine Kante $T_i \rightarrow T_j$, wenn T_i sich um ein Privileg bewirbt, das T_j besitzt und das, aufgrund der Kompatibilitätsmatrix, nicht zugeteilt werden kann.

  Ein Deadlock liegt genau dann vor, wenn der Wartegraph einen Zyklus hat.

Vermeidung von Livelocks und Deadlocks

- Vermeidung von Livelocks: *First-Come-First-Served* Strategie
- Vermeidung von Deadlocks:
  - Jede Transaktion bewirbt sich zu Beginn um alle benötigten Privilegien auf einmal (in einer atomaren Operation).
  - Auf den Objekten wird eine lineare Ordnung definiert. Die Transaktionen fordern ihre jeweiligen Privilegien gemäß dieser Ordnung an.
- **Wartegraph**:
  Ein Wartegraph hat eine Kante $T_i \rightarrow T_j$, wenn T_i sich um ein Privileg bewirbt, das T_j besitzt und das, aufgrund der Kompatibilitätsmatrix, nicht zugeteilt werden kann.

  Ein Deadlock liegt genau dann vor, wenn der Wartegraph einen Zyklus hat.

Wie kann ein Deadlock aufgelöst werden?
Nur indem eine beteiligte Transaktion abgebrochen wird.

Vermeidung von Livelocks und Deadlocks

- Vermeidung von Livelocks: *First-Come-First-Served* Strategie
- Vermeidung von Deadlocks:
  - Jede Transaktion bewirbt sich zu Beginn um alle benötigten Privilegien auf einmal (in einer atomaren Operation).
  - Auf den Objekten wird eine lineare Ordnung definiert. Die Transaktionen fordern ihre jeweiligen Privilegien gemäß dieser Ordnung an.
- **Wartegraph**:
  Ein Wartegraph hat eine Kante $T_i \rightarrow T_j$, wenn T_i sich um ein Privileg bewirbt, das T_j besitzt und das, aufgrund der Kompatibilitätsmatrix, nicht zugeteilt werden kann.

  Ein Deadlock liegt genau dann vor, wenn der Wartegraph einen Zyklus hat.

Wie kann ein Deadlock aufgelöst werden?
Nur indem eine beteiligte Transaktion abgebrochen wird.

2-Phasen Sperrverfahren 2PL (two-phase locking)

Hat eine Transaktion eine Freigabeoperation ausgeführt, dann darf sie keine Sperroperation mehr ausführen.

Mögliche Lock- und Unlock-Operationen gemäß 2PL der Transaktion $T = RA\ WA\ RB\ WB\ RC\ WC$

$$
\begin{aligned}
S_1 &: \quad LA\ RA\ WA\ LB\ RB\ WB\ LC\ RC\ WC\ UA\ UB\ UC \\
S_2 &: \quad LA\ RA\ WA\ LB\ LC\ UA\ RB\ WB\ UB\ RC\ WC\ UC \\
S_3 &: \quad LA\ LB\ LC\ RA\ WA\ UA\ RB\ WB\ UB\ RC\ WC\ UC \\
S_4 &: \quad LA\ LB\ LC\ RA\ WA\ RB\ WB\ RC\ WC\ UA\ UB\ UC
\end{aligned}
$$

2-Phasen Sperrverfahren 2PL (two-phase locking)

Hat eine Transaktion eine Freigabeoperation ausgeführt, dann darf sie keine Sperroperation mehr ausführen.

Mögliche Lock- und Unlock-Operationen gemäß 2PL der Transaktion $T = RA\ WA\ RB\ WB\ RC\ WC$

$$\begin{array}{ll} S_1: & LA\ RA\ WA\ LB\ RB\ WB\ LC\ RC\ WC\ UA\ UB\ UC \\ S_2: & LA\ RA\ WA\ LB\ LC\ UA\ RB\ WB\ UB\ RC\ WC\ UC \\ S_3: & LA\ LB\ LC\ RA\ WA\ UA\ RB\ WB\ UB\ RC\ WC\ UC \\ S_4: & LA\ LB\ LC\ RA\ WA\ RB\ WB\ RC\ WC\ UA\ UB\ UC \end{array}$$

Striktes 2PL

2PL ist *strikt*, wenn alle Freigabeoperationen einer Transaktion am Transaktionsende ausgeführt werden.

2-Phasen Sperrverfahren 2PL (two-phase locking)

Hat eine Transaktion eine Freigabeoperation ausgeführt, dann darf sie keine Sperroperation mehr ausführen.

Mögliche Lock- und Unlock-Operationen gemäß 2PL der Transaktion $T = RA\ WA\ RB\ WB\ RC\ WC$

$$
\begin{array}{ll}
S_1: & LA\ RA\ WA\ LB\ RB\ WB\ LC\ RC\ WC\ UA\ UB\ UC \\
S_2: & LA\ RA\ WA\ LB\ LC\ UA\ RB\ WB\ UB\ RC\ WC\ UC \\
S_3: & LA\ LB\ LC\ RA\ WA\ UA\ RB\ WB\ UB\ RC\ WC\ UC \\
S_4: & LA\ LB\ LC\ RA\ WA\ RB\ WB\ RC\ WC\ UA\ UB\ UC
\end{array}
$$

Striktes 2PL

2PL ist *strikt*, wenn alle Freigabeoperationen einer Transaktion am Transaktionsende ausgeführt werden.

Sperrpunkt

$$T_1 = L_1A\ R_1A\ L_1B\ U_1A\ W_1B\ U_1B,$$
$$T_2 = L_2A\ R_2A\ W_2A\ U_2A,$$
$$T_3 = L_3C\ R_3C\ U_3C.$$

$$S = L_1A\ R_1A\ L_1B\ \color{red}{U_1A}\ L_2A\ R_2A\ L_3C\ R_3C\ \color{red}{U_3C}\ W_1B\ U_1B\ W_2A\ \color{red}{U_2A}$$

Die Position der ersten Unlock-Operation einer Transaktion T_i in einem Schedule S ist der *Sperrpunkt* von T_i in S.

Das 2-Phasen Sperrprotokoll garantiert serialisierbare Schedule

Beweis:
Sei S ein Schedule einer Menge $\mathcal{T} = \{T_1, \ldots, T_n\}$, wobei jede Transaktion die Bedingung des 2PL-Protokolls erfüllt. Vereinfachend nehmen wir an, dass alle Transaktionen Schreibsperren erwerben.

Sperrpunkt

$$T_1 = L_1A\ R_1A\ L_1B\ U_1A\ W_1B\ U_1B,$$
$$T_2 = L_2A\ R_2A\ W_2A\ U_2A,$$
$$T_3 = L_3C\ R_3C\ U_3C.$$

$$S = L_1A\ R_1A\ L_1B\ U_1A\ L_2A\ R_2A\ L_3C\ R_3C\ U_3C\ W_1B\ U_1B\ W_2A\ U_2A$$

Die Position der ersten Unlock-Operation einer Transaktion T_i in einem Schedule S ist der *Sperrpunkt* von T_i in S.

Das 2-Phasen Sperrprotokoll garantiert serialisierbare Schedule

Beweis:
Sei S ein Schedule einer Menge $\mathcal{T} = \{T_1, \dots, T_n\}$, wobei jede Transaktion die Bedingung des 2PL-Protokolls erfüllt. Vereinfachend nehmen wir an, dass alle Transaktionen Schreibsperren erwerben.

Das 2-Phasen Sperrprotokoll garantiert serialisierbare Schedule (fortgesetzt).

Angenommen, S ist nicht serialisierbar ist, d.h. der Konfliktgraph $KG(S)$ enthält einen Zyklus, ohne Beschränkung der Allgemeinheit in der Form $T_1 \rightarrow T_2 \rightarrow \cdots \rightarrow T_k \rightarrow T_1$.

- Eine Kante $T_i \rightarrow T_j$ eines solchen Zyklus setzt voraus, dass T_i und T_j zu einem gemeinsamen Objekt A jeweils eine Operation ausführen, von denen mindestens eine schreibend ist.

Das 2-Phasen Sperrprotokoll garantiert serialisierbare Schedule (fortgesetzt).

Angenommen, S ist nicht serialisierbar ist, d.h. der Konfliktgraph $KG(S)$ enthält einen Zyklus, ohne Beschränkung der Allgemeinheit in der Form $T_1 \rightarrow T_2 \rightarrow \cdots \rightarrow T_k \rightarrow T_1$.

- Eine Kante $T_i \rightarrow T_j$ eines solchen Zyklus setzt voraus, dass T_i und T_j zu einem gemeinsamen Objekt A jeweils eine Operation ausführen, von denen mindestens eine schreibend ist.
- Da die Operation zu A in T_i und T_j jeweils durch eine Sperr- und Freigabeoperation umfasst ist, kann T_j seine Operation zu A erst nach der Freigabeoperation von T_i zu A ausführen. Betrachten wir alle Kanten des Zyklus, dann müssen Objekte $A_1, \ldots, A_k$ existieren, so dass für die Struktur von S gilt:

$$
\begin{aligned}
S &= \ldots U_1 A_1 \ldots L_2 A_1 \ldots, \\
&\vdots \\
S &= \ldots U_{k-1} A_{k-1} \ldots L_k A_{k-1} \ldots, \\
S &= \ldots U_k A_k \ldots L_1 A_k \ldots.
\end{aligned}
$$

Das 2-Phasen Sperrprotokoll garantiert serialisierbare Schedule (fortgesetzt).

Angenommen, S ist nicht serialisierbar ist, d.h. der Konfliktgraph $KG(S)$ enthält einen Zyklus, ohne Beschränkung der Allgemeinheit in der Form $T_1 \to T_2 \to \cdots \to T_k \to T_1$.

- Eine Kante $T_i \to T_j$ eines solchen Zyklus setzt voraus, dass T_i und T_j zu einem gemeinsamen Objekt A jeweils eine Operation ausführen, von denen mindestens eine schreibend ist.
- Da die Operation zu A in T_i und T_j jeweils durch eine Sperr- und Freigabeoperation umfasst ist, kann T_j seine Operation zu A erst nach der Freigabeoperation von T_i zu A ausführen. Betrachten wir alle Kanten des Zyklus, dann müssen Objekte $A_1, \ldots, A_k$ existieren, so dass für die Struktur von S gilt:

$$\begin{aligned} S &= \ldots U_1 A_1 \ldots L_2 A_1 \ldots, \\ &\vdots \\ S &= \ldots U_{k-1} A_{k-1} \ldots L_k A_{k-1} \ldots, \\ S &= \ldots U_k A_k \ldots L_1 A_k \ldots. \end{aligned}$$

- Sei l_i der Sperrpunkt von T_i, $1 \leq i \leq k$. Dann impliziert S, dass $l_1 < l_2 < \cdots < l_{k-1} < l_k$ und $l_k < l_1$.
- Aufgrund der Definition eines Sperrpunktes ist dies jedoch ein Widerspruch zu der Struktur von S. Damit ist gezeigt, dass 2PL serialisierbare Schedule garantiert.

Das 2-Phasen Sperrprotokoll garantiert serialisierbare Schedule (fortgesetzt).

Angenommen, S ist nicht serialisierbar ist, d.h. der Konfliktgraph $KG(S)$ enthält einen Zyklus, ohne Beschränkung der Allgemeinheit in der Form $T_1 \rightarrow T_2 \rightarrow \cdots \rightarrow T_k \rightarrow T_1$.

- Eine Kante $T_i \rightarrow T_j$ eines solchen Zyklus setzt voraus, dass T_i und T_j zu einem gemeinsamen Objekt A jeweils eine Operation ausführen, von denen mindestens eine schreibend ist.
- Da die Operation zu A in T_i und T_j jeweils durch eine Sperr- und Freigabeoperation umfasst ist, kann T_j seine Operation zu A erst nach der Freigabeoperation von T_i zu A ausführen. Betrachten wir alle Kanten des Zyklus, dann müssen Objekte $A_1, \ldots, A_k$ existieren, so dass für die Struktur von S gilt:

$$
\begin{aligned}
S &= \ldots U_1 A_1 \ldots L_2 A_1 \ldots, \\
&\vdots \\
S &= \ldots U_{k-1} A_{k-1} \ldots L_k A_{k-1} \ldots, \\
S &= \ldots U_k A_k \ldots L_1 A_k \ldots.
\end{aligned}
$$

- Sei l_i der Sperrpunkt von T_i, $1 \leq i \leq k$. Dann impliziert S, dass $l_1 < l_2 < \cdots < l_{k-1} < l_k$ und $l_k < l_1$.
- Aufgrund der Definition eines Sperrpunktes ist dies jedoch ein Widerspruch zu der Struktur von S. Damit ist gezeigt, dass 2PL serialisierbare Schedule garantiert.

Optimalität und Mächtigkeit von 2PL

- 2PL ist ein optimales Sperrverfahren in dem Sinn, dass zu jeder nicht 2-phasigen Transaktion T_1 eine 2-phasige Transaktion T_2 konstruiert werden kann, so dass zu T_1 und T_2 ein nicht serialisierbarer Schedule existiert.
- Es existieren konfliktserialisierbare Schedule, die bei Einhaltung von 2PL nicht entstehen können.

Optimalität und Mächtigkeit von 2PL

- 2PL ist ein optimales Sperrverfahren in dem Sinn, dass zu jeder nicht 2-phasigen Transaktion T_1 eine 2-phasige Transaktion T_2 konstruiert werden kann, so dass zu T_1 und T_2 ein nicht serialisierbarer Schedule existiert.
- Es existieren konfliktserialisierbare Schedule, die bei Einhaltung von 2PL nicht entstehen können.

Beweis

- Sei $L_1A\ U_1A\ L_1B\ U_1B$ die nicht 2-phasige Folge von Sperr- und Freigabeoperationen einer Transaktion T_1 und $L_2A\ L_2B\ U_2A\ U_2B$ eine 2-phasige Folge von Sperr-und Freigabeoperationen von T_2.

  Dann ist der folgende, durch seine Sperr- und Freigabeoperationen definierte, nicht serialisierbare Schedule möglich:

$$S = L_1A\ U_1A\ L_2A\ L_2B\ U_2A\ U_2B\ L_1B\ U_1B$$

Optimalität und Mächtigkeit von 2PL

- 2PL ist ein optimales Sperrverfahren in dem Sinn, dass zu jeder nicht 2-phasigen Transaktion T_1 eine 2-phasige Transaktion T_2 konstruiert werden kann, so dass zu T_1 und T_2 ein nicht serialisierbarer Schedule existiert.
- Es existieren konfliktserialisierbare Schedule, die bei Einhaltung von 2PL nicht entstehen können.

Beweis

- Sei $L_1A\ U_1A\ L_1B\ U_1B$ die nicht 2-phasige Folge von Sperr- und Freigabeoperationen einer Transaktion T_1 und $L_2A\ L_2B\ U_2A\ U_2B$ eine 2-phasige Folge von Sperr-und Freigabeoperationen von T_2.

  Dann ist der folgende, durch seine Sperr- und Freigabeoperationen definierte, nicht serialisierbare Schedule möglich:

$$S = L_1A\ U_1A\ L_2A\ L_2B\ U_2A\ U_2B\ L_1B\ U_1B$$

- $S = R_1A\ R_2A\ W_2A\ R_3B\ W_3B\ W_1B$
  ist konfliktserialisierbar, jedoch nicht bei Anwendung von 2PL entstehbar.

Optimalität und Mächtigkeit von 2PL

- 2PL ist ein optimales Sperrverfahren in dem Sinn, dass zu jeder nicht 2-phasigen Transaktion T_1 eine 2-phasige Transaktion T_2 konstruiert werden kann, so dass zu T_1 und T_2 ein nicht serialisierbarer Schedule existiert.
- Es existieren konfliktserialisierbare Schedule, die bei Einhaltung von 2PL nicht entstehen können.

Beweis

- Sei $L_1A\ U_1A\ L_1B\ U_1B$ die nicht 2-phasige Folge von Sperr- und Freigabeoperationen einer Transaktion T_1 und $L_2A\ L_2B\ U_2A\ U_2B$ eine 2-phasige Folge von Sperr-und Freigabeoperationen von T_2.

  Dann ist der folgende, durch seine Sperr- und Freigabeoperationen definierte, nicht serialisierbare Schedule möglich:

  $$S = L_1A\ U_1A\ L_2A\ L_2B\ U_2A\ U_2B\ L_1B\ U_1B$$

- $S = R_1A\ R_2A\ W_2A\ R_3B\ W_3B\ W_1B$
  ist konfliktserialisierbar, jedoch nicht bei Anwendung von 2PL entstehbar.

9.2.3 Verfahren ohne Sperren

- Sperrverfahren sind nicht die einzige Technik zur Gewährleistung serialisierbarer Schedule.
- Eine Mehrbenutzerkontrolle wird formal durch eine Abbildung Φ beschrieben, die eine von den Transaktionen angeforderte (Eingabe-) Folge von Operationen S_I in eine serialisierbare auszuführende (Ausgabe-) Folge von Operationen S_O der Transaktionen transformiert.
- Es gilt $\Phi(S_I) = S_O$, wobei S_I ein angeforderter Präfix eines Schedules und S_O der entsprechende ausgeführte Schedule.
- S_I ist ein Präfix, da ein Abbruch angefangener Transaktionen erforderlich werden kann.

9.2.3 Verfahren ohne Sperren

- Sperrverfahren sind nicht die einzige Technik zur Gewährleistung serialisierbarer Schedule.
- Eine Mehrbenutzerkontrolle wird formal durch eine Abbildung Φ beschrieben, die eine von den Transaktionen angeforderte (Eingabe-) Folge von Operationen S_I in eine serialisierbare auszuführende (Ausgabe-) Folge von Operationen S_O der Transaktionen transformiert.
- Es gilt $\Phi(S_I) = S_O$, wobei S_I ein angeforderter Präfix eines Schedules und S_O der entsprechende ausgeführte Schedule.
- S_I ist ein Präfix, da ein Abbruch angefangener Transaktionen erforderlich werden kann.

Scheduler ohne Sperren

- Angeforderte Aktionen kommen zur Ausführung, sofern der Scheduler sicher ist, dass kein nicht serialisierbarer Schedule in der Entstehung ist.
- Anderenfalls wird eine *aktive* Transaktion abgebrochen.
- Dies kann den Abbruch anderer, *abhängiger* Transaktionen erfordern.

Scheduler ohne Sperren

- Angeforderte Aktionen kommen zur Ausführung, sofern der Scheduler sicher ist, dass kein nicht serialisierbarer Schedule in der Entstehung ist.
- Anderenfalls wird eine *aktive* Transaktion abgebrochen.
- Dies kann den Abbruch anderer, *abhängiger* Transaktionen erfordern.

Aktive und abhängige Transaktionen

- Eine Transaktion T ist von einer Transaktion T' *abhängig*, wenn ein gerichteter Weg $T' \rightarrow \ldots \rightarrow T$ im Konfliktgraphen des zugehörigen Schedule existiert, der durch eine Folge von WR-Konflikten begründet ist.

Scheduler ohne Sperren

- Angeforderte Aktionen kommen zur Ausführung, sofern der Scheduler sicher ist, dass kein nicht serialisierbarer Schedule in der Entstehung ist.
- Anderenfalls wird eine *aktive* Transaktion abgebrochen.
- Dies kann den Abbruch anderer, *abhängiger* Transaktionen erfordern.

Aktive und abhängige Transaktionen

- Eine Transaktion T ist von einer Transaktion T' *abhängig*, wenn ein gerichteter Weg $T' \rightarrow \ldots \rightarrow T$ im Konfliktgraphen des zugehörigen Schedule existiert, der durch eine Folge von WR-Konflikten begründet ist.
- Eine Transaktion T heißt *aktiv* in einem Schedule S, wenn eine Operation von T in S enthalten ist und T noch nicht ihr Ende erreicht hat. Eine Transaktion signalisiert ihr Ende, indem sie als letztes die Operation *Commit* ausführt.

Scheduler ohne Sperren

- Angeforderte Aktionen kommen zur Ausführung, sofern der Scheduler sicher ist, dass kein nicht serialisierbarer Schedule in der Entstehung ist.
- Anderenfalls wird eine *aktive* Transaktion abgebrochen.
- Dies kann den Abbruch anderer, *abhängiger* Transaktionen erfordern.

Aktive und abhängige Transaktionen

- Eine Transaktion T ist von einer Transaktion T' *abhängig*, wenn ein gerichteter Weg $T' \rightarrow \ldots \rightarrow T$ im Konfliktgraphen des zugehörigen Schedule existiert, der durch eine Folge von WR-Konflikten begründet ist.
- Eine Transaktion T heißt *aktiv* in einem Schedule S, wenn eine Operation von T in S enthalten ist und T noch nicht ihr Ende erreicht hat. Eine Transaktion signalisiert ihr Ende, indem sie als letztes die Operation *Commit* ausführt.

Überwachen des Konfliktgraphen Φ_{KG}

Sei S die aktuelle Folge der ausgeführten Operationen und sei *op* die nächste angeforderte Operation einer Transaktion T.

Falls $KG(S \circ op)$ zyklenfrei, dann führe *op* aus. Anderenfalls breche T und alle von T abhängigen Transaktionen ab und streiche die entsprechenden Operationen dieser Transaktionen in S.

Vergabe von Zeitmarken Φ_{ZM}

Jeder Transaktion T wird bei ihrem Beginn eine eindeutige Zeitmarke $Z(T)$ zugewiesen.

Sei S die aktuelle Folge der ausgeführten Operationen und sei *op* die nächste angeforderte Operation einer Transaktion T.

Falls für alle Transaktionen T', die bereits eine zu *op* in Konflikt stehende Operation in S ausgeführt haben, gerade $Z(T') \leq Z(T)$, dann führe *op* aus. Anderenfalls breche T und alle von T abhängigen Transaktionen ab und streiche die entsprechenden Operationen dieser Transaktionen in S.

Optimistisches Verfahren Φ_{OP}

Sei S die aktuelle Folge der ausgeführten Operationen und sei *op* die nächste angeforderte Operation einer Transaktion T.

Sei des Weiteren $\mathit{Readset}(T)$ und $\mathit{Writeset}(T)$ die Menge der von einer Transaktion T bereits gelesenen bzw. geschriebenen Objekte.

Ist *op* verschieden von *Commit*, dann führe *op* aus. Ist *op* die *Commit*-Operation C, d.h. die letzte Operation von T, dann breche T und alle von T abhängigen Transaktionen ab, sofern bzgl. einer anderen aktiven Transaktion T' eine der folgenden Bedingungen gilt:

- $\mathit{Readset}(T) \cap \mathit{Writeset}(T') \neq \emptyset$
- $\mathit{Writeset}(T) \cap \mathit{Readset}(T') \neq \emptyset$
- $\mathit{Writeset}(T) \cap \mathit{Writeset}(T') \neq \emptyset$

Streiche des Weiteren alle Operationen der abgebrochenen Transaktionen in S.

Beispiel

$T_1 = R_1A\ W_1B\ C_1$
$T_2 = R_2A\ W_2A\ C_2$
$T_3 = R_3B\ W_3B\ C_3$

$S_I = R_1A\ R_2A\ W_2A\ C_2\ R_3B\ W_3B\ C_3\ W_1B\ C_1$

S_I	S_O
Φ_{KG}	S_I
Φ_{ZM}	$R_2A\ W_2A\ C_2\ R_3B\ W_3B\ C_3$
Φ_{OP}	$R_1A\ R_3B\ W_3B\ C_3\ W_1B\ C_1$
Φ_{2PL_strict}	$R_1A\ R_3B\ W_3B\ C_3\ W_1B\ C_1\ R_2A\ W_2A\ C_2$

9.2.4 Phantomproblem

bisher implizite Annahme

Die Menge der Objekte in der Datenbank ist konstant über die Zeit.
Verletzung dieser Annahme kann zu *Phantomen* führen.

Schedule mit Phantom

Sei eine Transaktion T_1 eine Ausführung eines Programmes P, das zunächst alle Objekte A liest, die eine gewisse Bedingung p erfüllen und anschließend ein weiteres Objekt B. T hat damit eine Historie der Form $R_1A_1 \ \ldots R_1A_k \ R_1B$.

Laufe zeitlich überlappend zu T_1 eine Transaktion T_2 ab, die ein Objekt C liest, ein neues Objekt A_{k+1} in die Datenbank einfügt, das ebenfalls die Bedingung p erfüllt und anschließend B ändert.

9.2.4 Phantomproblem

bisher implizite Annahme

Die Menge der Objekte in der Datenbank ist konstant über die Zeit.
Verletzung dieser Annahme kann zu *Phantomen* führen.

Schedule mit Phantom

Sei eine Transaktion T_1 eine Ausführung eines Programmes P, das zunächst alle Objekte A liest, die eine gewisse Bedingung p erfüllen und anschließend ein weiteres Objekt B. T hat damit eine Historie der Form $R_1 A_1 \ \ldots R_1 A_k \ R_1 B$.

Laufe zeitlich überlappend zu T_1 eine Transaktion T_2 ab, die ein Objekt C liest, ein neues Objekt A_{k+1} in die Datenbank einfügt, das ebenfalls die Bedingung p erfüllt und anschließend B ändert.

Unter diesen Annahmen ist der folgende Schedule möglich:

$S = R_2 C \ R_1 A_1 \ \ldots R_1 A_k \ W_2 A_{k+1} \ R_2 B \ W_2 B \ R_1 B$

S ist formal äquivalent zu $S' = T_2 \ T_1$; es wird jedoch ignoriert, dass A_{k+1} auch die Bedingung p erfüllt und somit T_1 eine Leseoperation $R_1 A_{k+1}$ enthalten müsste.

9.2.4 Phantomproblem

bisher implizite Annahme

Die Menge der Objekte in der Datenbank ist konstant über die Zeit.
Verletzung dieser Annahme kann zu *Phantomen* führen.

Schedule mit Phantom

Sei eine Transaktion T_1 eine Ausführung eines Programmes P, das zunächst alle Objekte A liest, die eine gewisse Bedingung p erfüllen und anschließend ein weiteres Objekt B. T hat damit eine Historie der Form $R_1 A_1 \ \ldots R_1 A_k \ R_1 B$.

Laufe zeitlich überlappend zu T_1 eine Transaktion T_2 ab, die ein Objekt C liest, ein neues Objekt A_{k+1} in die Datenbank einfügt, das ebenfalls die Bedingung p erfüllt und anschließend B ändert.

Unter diesen Annahmen ist der folgende Schedule möglich:

$S = R_2 C \ R_1 A_1 \ \ldots R_1 A_k \ W_2 A_{k+1} \ R_2 B \ W_2 B \ R_1 B$

S ist formal äquivalent zu $S' = T_2 \ T_1$; es wird jedoch ignoriert, dass A_{k+1} auch die Bedingung p erfüllt und somit T_1 eine Leseoperation $R_1 A_{k+1}$ enthalten müsste.

Lösung des Phantomproblems

- Vergrößerung der Granularität der betrachteten Objekte:

  Anstatt einer Folge von Leseoperationen $R_1 A_1 \dots R_1 A_k$ betrachten wir eine einzige Leseoperation, z.B. in der Form $R_1\{A \mid p(A)\}$.

  Da A_{k+1} die Eigenschaft p erfüllt, kann der Konflikt mit dem Phantom A_{k+1} erkannt werden.

- Sperrverfahren können den Test auf p implementieren, indem ganze Relationen, Schlüsselbereiche oder auch Indexbereiche gesperrt werden.

Lösung des Phantomproblems

- Vergrößerung der Granularität der betrachteten Objekte:

  Anstatt einer Folge von Leseoperationen $R_1A_1 \dots R_1A_k$ betrachten wir eine einzige Leseoperation, z.B. in der Form $R_1\{A \mid p(A)\}$.

  Da A_{k+1} die Eigenschaft p erfüllt, kann der Konflikt mit dem Phantom A_{k+1} erkannt werden.

- Sperrverfahren können den Test auf p implementieren, indem ganze Relationen, Schlüsselbereiche oder auch Indexbereiche gesperrt werden.