	Lehrveranstaltung	**Grundlagen von Datenbanken**		WS 2016/17
	Aufgabenzettel	**5 (Lösungsvorschläge)**		
	Gesamtpunktzahl	**40**		
	Ausgabe	**Mi. 14.12.2016**	Abgabe	**Fr. 13.01.2017**

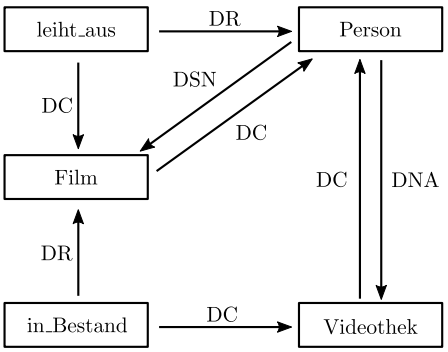
1 Referentielle Aktionen

[9 P.]

Betrachten Sie das folgende Datenbankschema:

Person(Vorname, Nachname, DOB, Wohnort, Lieblingsfilm → Film.IMDb-ID, Videothek → Videothek.VID)
Film(IMDb-ID, Titel, (ProduzentVN, ProduzentNN) → (Person.Vorname, Person.Nachname))
Videothek(VID, Adresse, Name, (LeiterVN, LeiterNN) → (Person.Vorname, Person.Nachname))
in_Bestand(Videothek → Videothek.VID, Film → Film.IMDb-ID, Anzahl)
leiht_aus((Vorname, Nachname) → (Person.Vorname, Person.Nachname), Film → Film.IMDb-ID, Datum)

Der SQL-Standard erlaubt die Definition von referentiellen Aktionen, um Verletzungen der referentiellen Integrität zu vermeiden. Der unten abgebildete Referenzgraph zeigt die im gegebenen Datenbankschema geltenden referentiellen Aktionen.



a) Welche Anforderung erfüllt ein (bzgl. der referentiellen Aktionen) sicheres Schema? [1 P.]

Lösungsvorschlag:

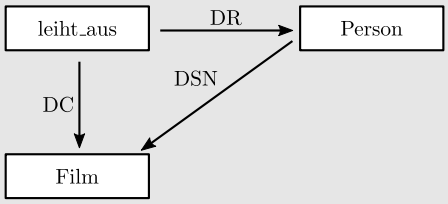
Bei einem sicheren Schema ist das Ergebnis einer Änderungsoperation unabhängig von der Reihenfolge, in der die referentiellen Aktionen ausgeführt werden; es treten also keine reihenfolgeabhängige Ergebnisse auf.

b) Handelt es sich im vorliegenden Fall um ein sicheres Schema? [8 P.]
Sollte dies nicht der Fall sein, diskutieren Sie alle Szenarien, bei denen reihenfolgeabhängige Ergebnisse auftreten können.

Lösungsvorschlag:

Reihenfolgeabhängige Ergebnisse können nur dann auftreten, wenn durch das Löschen eines Eintrages in einer Tabelle mehrere referentielle Aktionen ausgelöst werden, die zur gleichen Zieltabelle führen. Dies wiederum ist nur möglich, wenn mehrere Pfade von Fremdschlüsselreferenzen zwischen zwei Tabellen existieren. In unserem Beispiel ist dies in fünf Fällen gegeben:

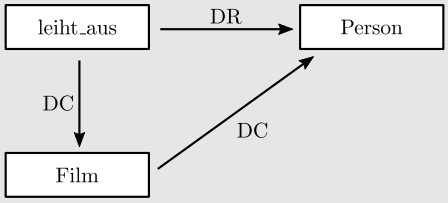
- **Fall 1 (Löschen eines Film-Eintrags F, betrachtete Zieltabelle leiht_aus): sicher**



  - Pfad 1: Film→leiht_aus
  - Pfad 2: Film→Person→leiht_aus

  Unabhängig von der Ausführungsreihenfolge werden alle leiht_aus-Einträge gelöscht, die F referenzieren (Pfad 1). Außerdem wird der Lieblingsfilm einer Person auf NULL gesetzt, wenn es sich dabei um F handelt (Pfad 2).

- **Fall 2 (Löschen eines Person-Eintrags P, betrachtete Zieltabelle leiht_aus): nicht sicher**



  - Pfad 1: Person→leiht_aus
  - Pfad 2: Person→Film→leiht_aus

  Das Ergebnis ist reihenfolgeabhängig, wenn die zu löschende Person P nur selbstproduzierte Filme ausgeliehen hat.

  Zuerst Auswertung von Pfad 1:

  - Die von P getätigten Ausleihen können *nicht* gelöscht werden.

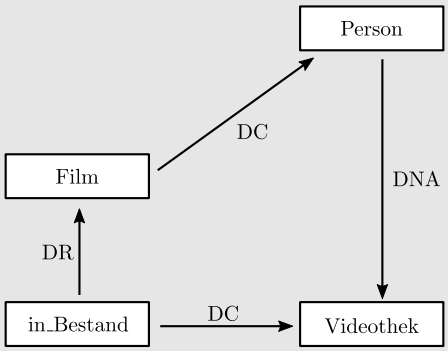
  ⇒ Der Löschvorgang wird abgebrochen.

  Zuerst Auswertung von Pfad 2:

  - Alle Filme, die von P produziert wurden, werden ebenfalls gelöscht.
  - Alle Ausleihen dieser Filme werden gelöscht.
  - P kann nun gelöscht werden, da für P keine Ausleihen mehr verzeichnet sind.

  ⇒ Der Löschvorgang ist erfolgreich.

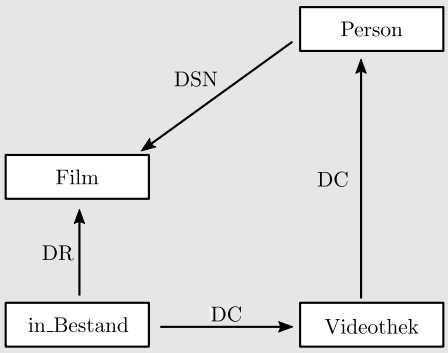
- **Fall 3 (Löschen eines Videothek-Eintrags V, betrachtete Zieltabelle in_Bestand): sicher**



  - Pfad 1: Videothek→in_Bestand
  - Pfad 2: Videothek→Person→Film→in_Bestand

  Unabhängig davon, welcher Pfad zuerst verfolgt wird, wird bei der Löschung einer Videothek V auch immer jeder in_Bestand-Eintrag gelöscht (Pfad 1). Das Löschen einer Videothek ist allerdings nur dann gestattet, wenn am Ende der Auswertung aller referentiellen Aktionen für diese Videothek keine Person vorhanden ist, die diese Videothek leitet (Pfad 2).

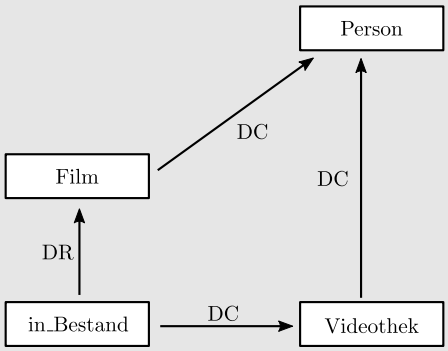
- **Fall 4 (Löschen eines Film-Eintrags F, betrachtete Zieltabelle in_Bestand): sicher**



  - Pfad 1: Film→in_Bestand
  - Pfad 2: Film→Person→Videothek→in_Bestand

  Sollte der zu löschende Film F bereits in den Bestand einer Videothek aufgenommen worden sein, wird die Löschung zurückgesetzt (Pfad 1). Ist F der Lieblingsfilm einer Person, wird der entsprechende Wert bei dieser Person auf NULL gesetzt (Pfad 2). Es liegen keine Reihenfolgeabhängigkeiten vor.

- **Fall 5 (Löschen eines Person-Eintrags P, betrachtete Zieltabelle in_Bestand): nicht sicher**



  - Pfad 1: Person→Videothek→in_Bestand
  - Pfad 2: Person→Film→in_Bestand

  Das Ergebnis ist reihenfolgeabhängig, wenn eine Person P gelöscht wird, die Leiter einer Videothek ist und nur Filme produziert hat, die ausschließlich in den eigenen Videotheken vorhanden sind.

  Zuerst Auswertung von Pfad 1:

  - Die von P produzierten Filme werden gelöscht.
  - Die entsprechenden in_Bestand-Einträge können jedoch *nicht* gelöscht werden, da diese noch in dem Bestand einer Videothek enthalten sind.

  ⇒ Der Löschvorgang wird abgebrochen.

  Zuerst Auswertung von Pfad 2:

  - Die von P geleitete Videothek wird gelöscht.
  - Alle in_Bestand-Einträge der betreffenden Videothek werden gelöscht.
  - Die von P produzierten Filme werden gelöscht. Dies ist möglich, da diese im Bestand keiner Videothek enthalten sind.

  ⇒ Der Löschvorgang ist erfolgreich.

2 Änderbarkeit von Sichten

[8 P.]

a) Gegeben seien die folgenden Basisrelationen:

Rennwagen(RNr, Typ, Rennserie, Rennstall, Jahr)
Mechaniker(Vorname, Nachname, Gehalt, Geburtsjahr, wartet → Rennwagen.RNr)

Geben Sie die SQL-Anweisungen an, die notwendig sind, um die folgenden Sichten zu erstellen. Geben Sie zu jeder dieser Sichten an, ob sie Änderungsoperationen auf den in ihr enthaltenen Tupeln erlaubt. Begründen Sie Ihre Antwort, falls dies nicht der Fall ist. [4 P.]

i) *FerrariMechaniker*: Vorname und Nachname aller Mechaniker, die Ferrari-Rennwagen warten.

Lösungsvorschlag:

```
CREATE VIEW FerrariMechaniker
  AS SELECT Vorname, Nachname
    FROM Mechaniker, Rennwagen
    WHERE wartet = RNr
      AND Rennstall = 'Ferrari';
```

Die Sicht ist aufgrund der Verbundoperationen nicht änderbar.

ii) *reicheMechaniker*: Vorname und Nachname aller Mechaniker, die ein Gehalt von mehr als 2 Millionen Geldeinheiten haben.

Lösungsvorschlag:

```
CREATE VIEW reicheMechaniker
  AS SELECT Vorname, Nachname
    FROM Mechaniker
    WHERE Gehalt > 2000000;
```

Die Sicht ist änderbar.

iii) *alteRennserien*: Die Namen aller Rennserien, die es schon vor 1950 gab.

Lösungsvorschlag:

```
CREATE VIEW alteRennserien
  AS SELECT Rennserie
    FROM Rennwagen
    WHERE Jahr < 1950;
```

Die Sicht ist nicht änderbar, da sie nicht den Primärschlüssel der `Rennwagen`-Relation enthält.

iv) *FerrariWagen*: RNr, Typ, Rennserie und Jahr aller Rennwagen aus dem Stall `'Ferrari'`.

Lösungsvorschlag:

```
CREATE VIEW FerrariWagen
  AS SELECT RNr, Typ, Rennserie, Jahr
    FROM Rennwagen
    WHERE Rennstall = 'Ferrari';
```

Die Sicht ist änderbar.

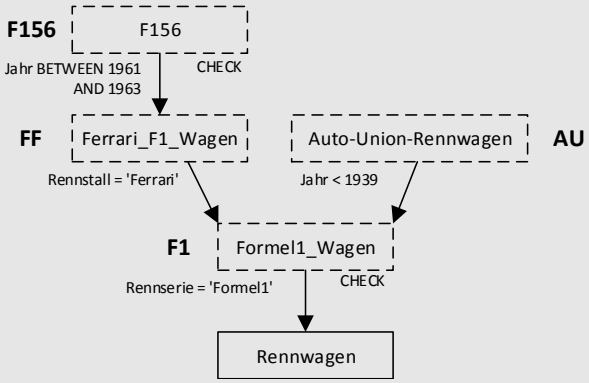
b) Auf der Rennwagen-Relation seien folgende Sichten definiert: [4 P.]

```
CREATE VIEW Formel1_Wagen
  AS SELECT * FROM Rennwagen
    WHERE Rennserie = 'Formel1'
    WITH CASCADED CHECK OPTION;

CREATE VIEW Ferrari_F1_Wagen
  AS SELECT * FROM Formel1_Wagen
    WHERE Rennstall = 'Ferrari';

CREATE VIEW Auto-Union-Rennwagen
  AS SELECT * FROM Formel1_Wagen
    WHERE Jahr < 1939;

CREATE VIEW F156
  AS SELECT * FROM Ferrari_F1_Wagen
    WHERE Jahr BETWEEN 1961 AND 1963
    WITH CASCADED CHECK OPTION;
```

Es handelt sich bei allen obigen Sichtdefinitionen um änderbare Sichten. Bestimmen Sie, ob die folgenden SQL-Anweisungen auf diesen Sichtdefinitionen durchgeführt werden können. Für die Fälle, in denen die Änderung bzw. das Einfügen zulässig ist, geben Sie außerdem an, in welchen Sichten **auf jeden Fall alle** geänderten/eingefügten Tupel nach Abschluss der Operation sichtbar werden.
Hinweis: Im Falle von `UPDATE`-Operationen ist davon auszugehen, dass die zu ändernden Tupel vor der Änderung die die Sicht definierenden Prädikate erfüllen.

Lösungsvorschlag:

In der folgenden Abbildung ist die Vererbungshierarchie im vorliegenden Beispiel abgebildet:



i)
```
UPDATE Formel1_Wagen
   SET Jahr = 1937
 WHERE Jahr = 1936
   AND Rennstall = 'McLaren';
```

Lösungsvorschlag:

Die Operation ist zulässig. Geänderte Tupel sind in den Relationen Formel1_Wagen und Auto-Union-Rennwagen sichtbar vor.

ii)
```
UPDATE F156
   SET Jahr = 2014
 WHERE Rennstall = 'Ferrari';
```

Lösungsvorschlag:

Die Operation wird zurückgewiesen, da die geänderten Tupel die Bedingung `Jahr BETWEEN 1961 AND 1963` nicht mehr erfüllen würden.

iii)
```
UPDATE Ferrari_F1_Wagen
   SET Rennstall = 'Lotus'
 WHERE Rennserie = 'Formel1'
   AND Jahr = 1960;
```

Lösungsvorschlag:

Die Operation ist zulässig. Geänderte Tupel sind in der Formel1_Wagen-Relation sichtbar.

iv)
```
INSERT INTO Auto-Union-Rennwagen
VALUES (12, 'Typ_C', 'AVUS', 'Auto-Union' , 1938 );
```

Lösungsvorschlag:

Die Operation wird zurückgewiesen, da die einzufügenden Tupel die Bedingung `Rennserie = 'Formel1'` nicht erfüllen würden.

3 Serialisierbarkeit, Anomalien

[15 P.]

Gegeben sind die folgenden Transaktionen $T_1 = r_1(A)\ w_1(A)\ w_1(B)$ und $T_2 = r_2(A)\ r_2(B)\ w_2(A)$. T_1 liest den Wert von A, erhöht diesen um 100 und schreibt den neuen Wert nach A zurück. Außerdem schreibt T_1 den alten Wert von A nach B (ohne B vorher gelesen zu haben). T_2 erhöht den gelesenen Wert von A um 200 und addiert den Wert von B dazu. Der Anfangswert von A sei 10 und der von B sei 5. Allgemein bezeichnet $w_i(x)$ den Schreibzugriff der Transaktion i auf das Objekt x und $r_i(x)$ den Lesezugriff der Transaktion i auf x. Gegeben sind die folgenden Schedules:

$$
\begin{aligned}
S_1 &= r_1(A)\ w_1(A)\ w_1(B)\ r_2(A)\ r_2(B)\ w_2(A)\\
S_2 &= r_2(A)\ r_2(B)\ w_2(A)\ r_1(A)\ w_1(A)\ w_1(B)\\
S_3 &= r_1(A)\ w_1(A)\ r_2(A)\ r_2(B)\ w_1(B)\ w_2(A)\\
S_4 &= r_1(A)\ r_2(A)\ r_2(B)\ w_1(A)\ w_1(B)\ w_2(A)\\
S_5 &= r_1(A)\ r_2(A)\ r_2(B)\ w_2(A)\ w_1(A)\ w_1(B)\\
S_6 &= r_1(A)\ w_1(A)\ r_2(A)\ w_1(B)\ r_2(B)\ w_2(A)
\end{aligned}
$$

Beantworten Sie für jeden der Schedules die folgenden drei Fragestellungen:

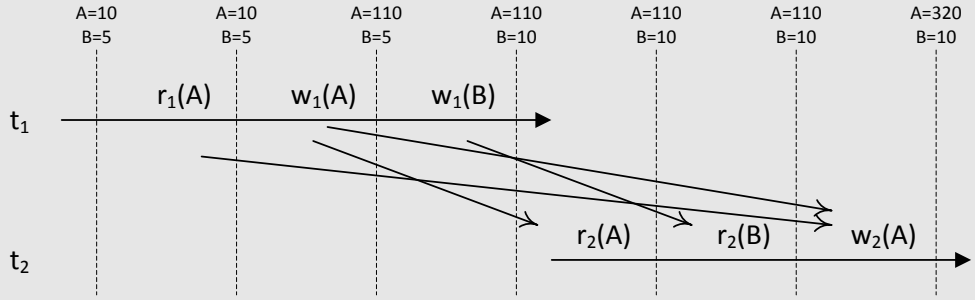
a) Wie lautet nach Ausführung des Schedules die Belegung für die Variablen A und B? [3 P.]

b) Welche Abhängigkeiten existieren zwischen den Operationen der beiden Transaktionen innerhalb des Schedules? [6 P.]

c) Ist der Schedule seriell, serialisierbar oder nicht serialisierbar? Erläutern sie zusätzlich bei einem nicht-serialisierbaren Schedule die auftretenden Datenanomalien. Begründen Sie die Antworten mit Hilfe der Abhängigkeiten! [6 P.]

Lösungsvorschlag:

- Schedule S_1:



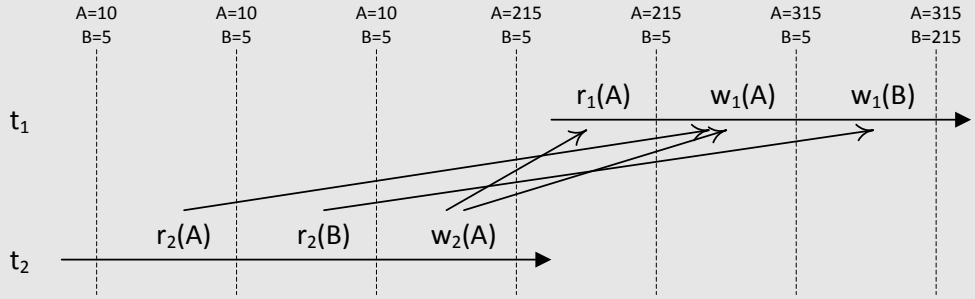
  a) Endwert für A ist 320, Endwert für B ist 10.

  b) Abhängigkeiten

  - $r_1(A) \rightarrow w_2(A)$
  - $w_1(A) \rightarrow r_2(A)$
  - $w_1(A) \rightarrow w_2(A)$
  - $w_1(B) \rightarrow r_2(B)$

  c) Der Schedule ist seriell.

- Schedule S_2:



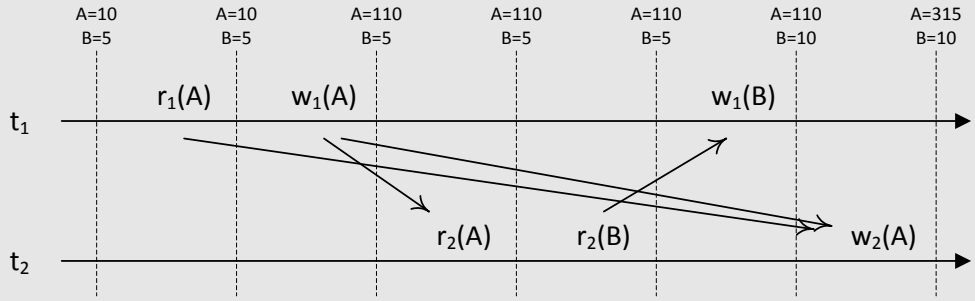
  a) Endwert für A ist 315, Endwert für B ist 215.

  b) Abhängigkeiten

  - $r_2(A) \rightarrow w_1(A)$
  - $r_2(B) \rightarrow w_1(B)$
  - $w_2(A) \rightarrow r_1(A)$
  - $w_2(A) \rightarrow w_1(A)$

c) Der Schedule ist seriell.

- Schedule S_3:



a) Endwert für A ist 315, Endwert für B ist 10.

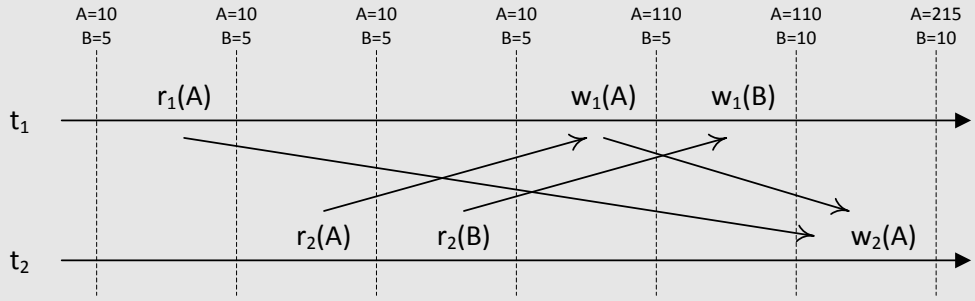
b) Abhängigkeiten

- $r_1(A) \rightarrow w_2(A)$
- $w_1(A) \rightarrow r_2(A)$
- $w_1(A) \rightarrow w_2(A)$
- $r_2(B) \rightarrow w_1(B)$

c)
- Wegen der Abhängigkeiten $r_1(A) \rightarrow w_2(A)$, $w_1(A) \rightarrow r_2(A)$ und $w_1(A) \rightarrow w_2(A)$ gilt: t_1 vor t_2.
- Wegen des Abhängigkeit $r_2(B) \rightarrow w_1(B)$ gilt: t_2 vor t_1. ↯

$\Rightarrow S_3$ ist nicht serialisierbar. Es gibt keine serielle Abfolge der beiden Transaktionen, die ein identisches Resultat für die Variablen A und B erzielt. Die Anomalie im Datenbestand entsteht, weil T_2 bereits den neuen Wert von A liest (Abhängigkeit: $w_1(A) \rightarrow r_2(A)$), aber noch den alten Wert von B benutzt (Abhängigkeit: $r_2(B) \rightarrow w_1(B)$).

- Schedule S_4:



a) Endwert für A ist 215, Endwert für B ist 10.

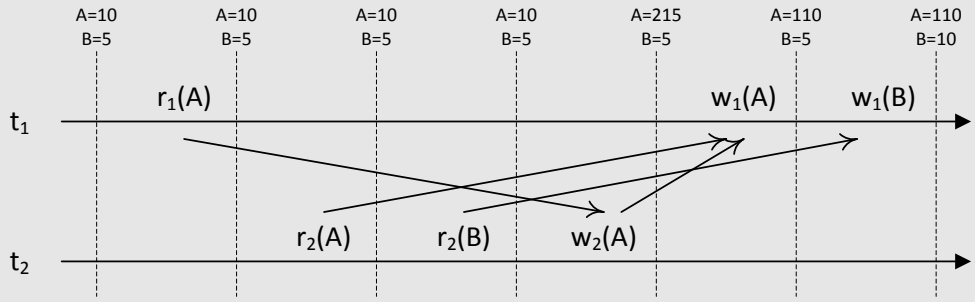
b) Abhängigkeiten

- $r_2(A) \rightarrow w_1(A)$
- $r_2(B) \rightarrow w_1(B)$
- $r_1(A) \rightarrow w_2(A)$
- $w_1(A) \rightarrow w_2(A)$

c)
- Wegen der Abhängigkeiten $r_2(A) \rightarrow w_1(A)$ und $r_2(B) \rightarrow w_1(B)$ gilt: t_2 vor t_1.
- Wegen der Abhängigkeiten $r_1(A) \rightarrow w_2(A)$ und $w_1(A) \rightarrow w_2(A)$ gilt: t_1 vor t_2. ↯

$\Rightarrow S_4$ ist nicht serialisierbar. Es gibt keine serielle Abfolge der beiden Transaktionen, die ein identisches Resultat für die Variablen A und B erzielt. Dabei überschreibt Transaktion t_2 alle Änderungen der Variable A, die von t_1 getätigt wurden (Lost-Update).

- Schedule S_5:



a) Endwert für A ist 110, Endwert für B ist 10.

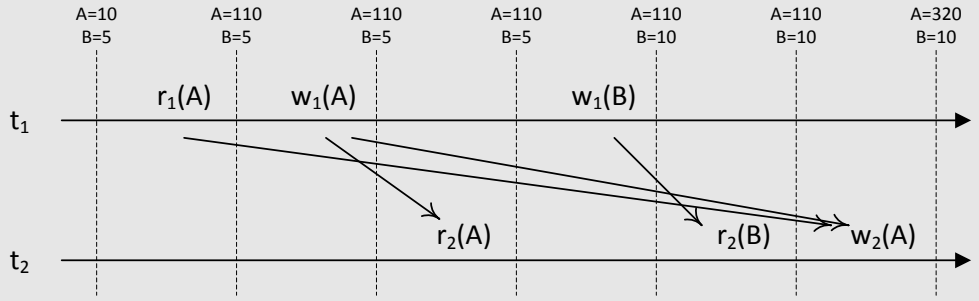
b) Abhängigkeiten

- $r_2(A) \rightarrow w_1(A)$
- $r_2(B) \rightarrow w_1(B)$
- $w_2(A) \rightarrow w_1(A)$
- $r_1(A) \rightarrow w_2(A)$

c)
- Wegen der Abhängigkeiten $r_2(A) \rightarrow w_1(A)$, $r_2(B) \rightarrow w_1(B)$ und $w_2(A) \rightarrow w_1(A)$ gilt: t_2 vor t_1.
- Wegen des Abhängigkeit $r_1(A) \rightarrow w_2(A)$ gilt: t_1 vor t_2. ↯

$\Rightarrow S_5$ ist nicht serialisierbar. Es gibt keine serielle Abfolge der beiden Transaktionen, die ein identisches Resultat für die Variablen A und B erzielt. Dabei überschreibt Transaktion t_1 alle Änderungen von t_2 (Lost-Update).

- Schedule S_6:



a) Endwert für A ist 320, Endwert für B ist 10.

b) Abhängigkeiten
   - $w_1(A) \rightarrow r_2(A)$
   - $w_1(B) \rightarrow r_2(B)$
   - $r_1(A) \rightarrow w_2(A)$
   - $w_1(A) \rightarrow w_2(A)$

c) – Wegen der Abhängigkeiten $r_1(A) \rightarrow w_2(A)$, $w_1(A) \rightarrow r_2(A)$ und $w_1(B) \rightarrow r_2(B)$ gilt: t_1 vor t_2.

$\Rightarrow S_6$ ist serialisierbar (Schedule S_1 erzielt das identische Ergebnis für die Variablen A und B)

4 2PL-Synchronisation mit R/X-Sperren

[8 P.]

Gegeben sind die drei Objekte x, y und z, welche von den Transaktionen T_1, T_2 und T_3 gelesen bzw. geschrieben werden. Dabei bezeichnet $w_i(x)$ den Schreibzugriff der Transaktion T_i auf das Objekt x und $r_i(x)$ den Lesezugriff der Transaktion T_i auf x.

Der Schedule S_1 zeigt an, in welcher Reihenfolge die Operationen der drei Transaktionen T_1, T_2, T_3 beim Scheduler eines Datenbanksystems eintreffen. Die Operation c_i soll das Commit der Transaktion T_i darstellen.

$$S_1 = w_1(x)\ r_2(z)\ r_1(x)\ r_3(y)\ w_2(y)\ w_1(y)\ c_1\ r_2(z)\ c_2\ w_3(y)\ c_3$$

Bei der Ausführung von S_1 soll das RX-Sperrverfahren mit 2PL zum Einsatz kommen. Vervollständigen Sie die auf der nächsten Seite angegebene Tabelle, indem Sie die Sperranforderungen (lock) und -freigaben (unlock) der Transaktionen, deren Lese- und Schreibzugriffe (read bzw. write) und Commits (commit) eintragen. Beachten Sie, dass eine Transaktion innerhalb eines Zeitschritts nur jeweils eine Operation durchführen kann. Nutzen Sie die Spalte „Bemerkungen“ für etwaige Wartebeziehungen und Benachrichtigungen an wartende Transaktionen.

Lösungsvorschlag:

$$S_1 = w_1(x)\ r_2(z)\ r_1(x)\ r_3(y)\ w_2(y)\ w_1(y)\ c_1\ r_2(z)\ c_2\ w_3(y)\ c_3$$

Zeitschritt	T_1	T_2	T_3	Bemerkung
0				
1	lock(x,X)			
2	write(x)	lock(z,R)		
3	read(x)	read(z)	lock(y,R)	
4		lock(y,X)	read(y)	T_2 wartet auf Freigabe von y
5	lock(y,X)			T_1 wartet auf Freigabe von y
6			lock(y,X)	
7			write(y)	
8			unlock(y)	T_2 wird benachrichtigt
9		write(y)	commit	
10		read(z)		
11		unlock(z)		
12		unlock(y)		T_1 wird benachrichtigt
13	write(y)	commit		
14	unlock(x)			
15	unlock(y)			
16	commit			