Aufgabe 10.1:

Lösung: Aus Konfliktserialisierbarkeit folgt allgemeine Serialisierbarkeit. Bleibt zu zeigen, dass jetzt auch aus Serialisierbarkeit Konfliktserialisierbarkeit folgt, falls die Transaktionen keine *blind writes* enthalten. Gibt es keine *blind writes*, dann sind je zwei Schreibaktionen W_iA, W_jA eines seriellen Schedules S' o.B.d.A. durch einen Pfad der Form $W_iA \longrightarrow \ldots \longrightarrow R_kA \longrightarrow W_kA \longrightarrow \ldots \longrightarrow R_jA \longrightarrow W_jA$ im Abhängigkeitsgraphen $AG(S')$ verbunden.

Sei S serialisierbar gegeben. Wir zeigen, dass $KG(S)$ zyklenfrei.

Es existiert somit ein serieller Schedule S' mit $AG(S) = AG(S')$. Weiter ist $KG(S')$ zyklenfrei. Um auch $KG(S)$ zyklenfrei zu zeigen, genügt es

$$T_i \longrightarrow T_j \in KG(S) \Rightarrow T_i \longrightarrow \ldots \longrightarrow T_j \in KG(S')$$

zu zeigen. D.h., jeder Konflikt in $KG(S)$ tritt direkt oder indirekt auch in $KG(S')$ auf. Betrachte also einen beliebigen Konflikt $T_i \longrightarrow T_j$ in $KG(S)$:

1) (WR-Konflikt) Also $W_iA \ldots R_jA$ ohne dazwischenliegende WA-Aktion in S. Es ist somit auch $W_iX \longrightarrow R_jX \in AG(S)$. Wegen $AG(S) = AG(S')$ folgt $T_i \longrightarrow T_j \in KG(S')$.
2) (WW-Konflikt) Also $W_iA \ldots W_jA$ ohne dazwischenliegende WA-Aktion in S. W_iA und W_jA sind in $AG(S')$ durch einen Pfad verbunden. Damit enthält $AG(S')$ entweder einen Pfad (i) der Form $W_iA \longrightarrow \ldots \longrightarrow R_kA \longrightarrow W_kA \longrightarrow \ldots \longrightarrow R_jA \longrightarrow W_jA$, oder (ii) der Form $W_jA \longrightarrow \ldots \longrightarrow R_kA \longrightarrow W_kA \longrightarrow \ldots \longrightarrow R_iA \longrightarrow W_iA$. Beachte $AG(S) = AG(S')$. Im Falle (i) enthält $KG(S')$ einen Pfad $T_i \longrightarrow \ldots \longrightarrow T_j$, wie gewünscht. Im Falle (ii) folgt $S = \ldots W_jA \ldots R_iA \ldots W_iA \ldots$, ein Widerspruch zum angenommenen WW-Konflikt.
3) (RW-Konflikt) Also $R_iA \ldots W_jA$ ohne dazwischenliegende WA-Aktion in S. Sei W_kA die letzte Schreibaktion vor R_iA in S und damit wegen $AG(S) = AG(S')$ auch in S'. S' hat somit entweder (i) die Form $T_k \ldots T_i \ldots T_j \ldots$ oder (ii) die Form $T_j \ldots T_k \ldots T_i \ldots$. Im Falle (i) enthält $KG(S')$ einen Pfad $T_i \longrightarrow \ldots \longrightarrow T_j$, wie gewünscht. Im Falle (ii) hat $AG(S')$ einen Pfad $W_jA \longrightarrow \ldots \longrightarrow R_kA \longrightarrow W_kA \longrightarrow R_iA$. Damit muss S die Form $\ldots W_jA \ldots W_kA \ldots R_iA \ldots$, ein Widerspruch.

Aufgabe 10.2:

(a)

Lösung: Der Konfliktgraph enthält einen Zyklus $T_1 \longrightarrow T_2 \longrightarrow T_1$.

(b)

Lösung: Sei S' der angegebene serielle Schedule. S und S' haben die gleichen Abghängigkeitsgraphen.

(c)

Lösung: Streichen wir in S die Transaktion T_3, dann ist der resultierende Schedule nicht serialisierbar.

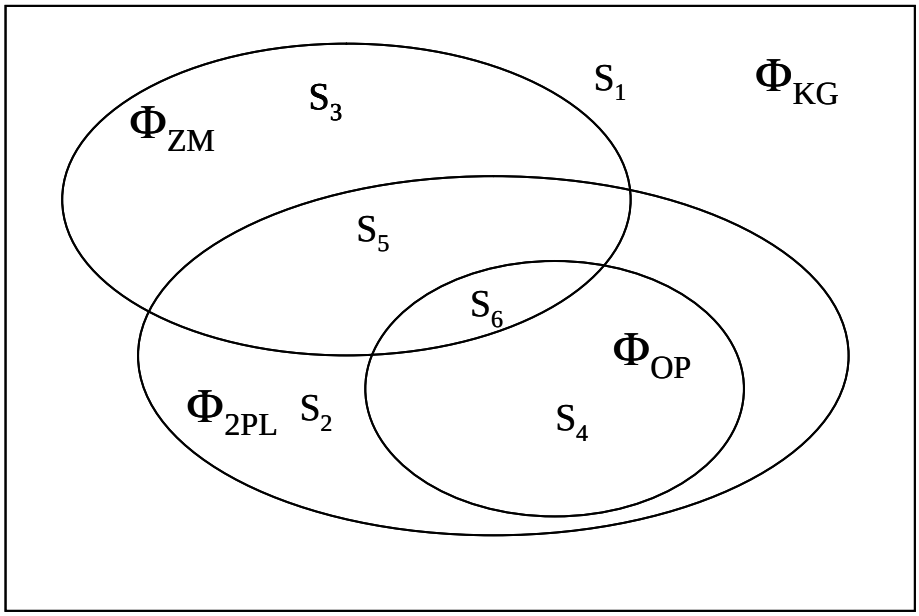
Aufgabe 10.3:

Lösung:

(a) Wir betrachten jeweils einen Schedule $S \in \{\Phi(S_I) \mid S_I \text{ ein Schedule}\}$ und nehmen an, so dass $CG(S)$ einen Zyklus der Form $T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow \ldots \rightarrow T_n \rightarrow T_1$ besitzt und führen dies dann zum Widerspruch mit der Definition der betreffenden Variante einer Mehrbenutzerkontrolle.

1) 2-Phasen-Sperren: siehe Beweis zu Satz (**??**).
2) Überwachen des Konfliktgraphen: unsere Annahme führt direkt zum Widerspruch.
3) Zeitmarken: Wenn $T_i \rightarrow T_j$ in CG existiert, dann existiert ein Konflikt zwischen T_i und T_j, so daß die Anordnung $\ldots T_i \ldots T_j \ldots$ im Schedule S existiert. Dies bedeutet, dass T_i wurde vor T_j gestartet, da der Schedule sonst nicht von dem zeitmarkenbasierten Scheduler erzeugt worden wäre.
   Also $Z(T_i) < Z(T_j)$. Aus dem angenommenen Zyklus von CG folgt $Z(T_1) < Z(T_2) < \ldots < Z(T_n) < Z(T_1)$, was ein Widerspruch zur Ordnung der Startzeitpunkte ist.
4) Eine Kante $T \rightarrow T'$ in unserem Zyklus setzt voraus, dass T und T' zu einem gemeinsamen Objekt A jeweils eine Operation ausführen, von denen mindestens eine schreibend ist und zwar in der Reihenfolge T vor T'. Der Schedule S kann nur entstehen, wenn zum Zeitpunkt des Commit von T' die Transaktion T nicht mehr aktiv ist. D.h., T führt sein Commit vor dem Commit von T' aus. Da zu einem Zeitpunkt nur höchstens eine Transaktion das Commit durchführen kann, folgt aus dem Zyklus ein Widerspruch zu der Ordnung der Commit-Zeitpunkte.

(b) Seien $\Phi_{2PL}, \Phi_{2KG}, \Phi_{ZM}$, und Φ_{OP} jeweils die betreffenden Fixpunkte. Das folgende Diagramm zeigt das Verhältnisse der einzelnen Fixpunkte zueinander.



S_1 :

T_1 :	*RA*	*WA*					*RD*
T_2 :			*RA*	*WB*			
T_3 :					*RC*	*WD*	

S_2 :

T_1 :		*RA*	*WA*			*RD*	
T_2 :				*RA*	*WB*		
T_3 :	*RC*						*WD*

S_3 :

T_1 :		*RA*	*WA*				*RD*
T_2 :				*RA*	*WB*		
T_3 :	*RC*					*WD*	

S_4 :

T_1 :		*RA*	*WA*	*RD*			
T_2 :					*RA*	*WB*	
T_3 :	*RC*						*WD*

S_5 :

T_1 :	*RA*		*WA*			*RD*	
T_2 :				*RA*	*WB*		
T_3 :		*RC*					*WD*

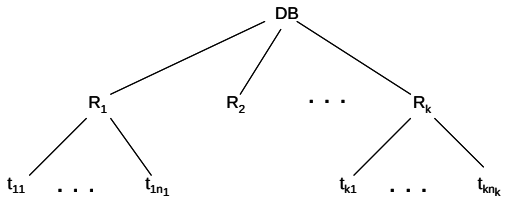
$$
\begin{array}{lllllllll}
 & T_1: & RA & & WA & RD & & & \\
S_6: & T_2: & & & & & RA & WB & \\
 & T_3: & & RC & & & & & WD
\end{array}
$$

Es bleibt zu zeigen, dass $\Phi_{OP} \subseteq \Phi_{2PL}$. Das optimistische Verfahren testet in der definierten Weise den *Readset* und *Writeset* einer Transaktion T, die ihr *commit* ausführt, mit den *Readset*s und *Writeset*s aller anderen aktiven Transaktionen T' auf Disjunktheit. Sei S ein Schdule, zu dem die gewünschten Tests erfüllt sind. Dann gilt für jede zu dem betrachtetetn Zeitpunkt aktive Transaktion T', dass sie bis zu diesem Zeitpunkt keine zu einer Aktion von T in Konflikt stehende Aktion ausgeführt hat. T und T' könnten somit in S auch gemäß dem 2PL-Protokoll abgelaufen sein, ohne dass ein Sperrkonflikt aufgetreten wäre. Damit kann der Schedule S auch bei Anwendung eines 2PL-Protokolls entstehen.

Aufgabe 10.4:

Lösung: Ein Vorteil des geforderten Verfahrens ist einmal, dass eine Transaktion, die viele Tupel einer Relation ändern möchte, mit einer Sperroperation exklusiven Zugriff zu der gesamten Relation erhalten kann. Zum anderen genügt es, dass eine Transaktion die wenige Tupel ändern möchte, nur gerade zu diesen einen exklusiven Zugriff erhält.

Wir beschränken uns auf Schreibprivilegien, d.h. die Verwendung von Sperren der Form LA. Die zu betrachtenden Objekt können in einer *Objekt-Hierarchie* angeordnet werden, die eine Enthaltensein-Beziehung ausdrückt:



In einem solchen Szenario muss eine Mehrbenutzerkontrolle in der Lage sein, beispielsweise einen Konflikt zwischen einer Sperre der Form LR_1 und einer Sperre der Form Lt_{11} zu erkennen. Das folgende *Warnsperren*-Protokoll leistet das Gewünschte unter Verwendung von Warnsperren, hier der Form $L^!R_1$. Hierbei sind Warn- und Schreibsperren zu einander nicht kompatibel, während Warnsperren untereinander kompatibel sind.

1) Die Transaktionen halten bzgl. Warn- und Schreibsperren das 2-Phasen-Sperrverfahren ein.
2) Eine Transaktion kann ein Objekt nur dann mit einer Warnsperre sperren, wenn alle übergeordneten Objekte in der Objekt-Hierarchie mit einer Warnsperre erfolgreich gesperrt wurden.
3) Eine Transaktion kann ein Objekt nur dann mit einer Schreibsperre sperren, wenn alle übergeordneten Objekte in der Objekt-Hierarchie mit einer Warnsperre erfolgreich gesperrt wurden.
4) Eine Transaktion darf ein gesperrtes Objekt erst dann freigeben, wenn alle zuvor gesperrten untergeordneten Objekte freigegeben sind.

Aufgabe 10.5:

Lösung:

(a) B-Baum

2PL angewendet auf einen B-Baum hat offensichtlich den Nachteil, dass bei Einfügen oder Löschen eines Schlüssels die Wurzel aufgrund einer möglicherweise erforderlich werdenden Restrukturierung des Baumes für die gesmte Dauer der Operation gesperrt werden muss (hot spot). Damit serialisieren sich die einzelnen Operationen vor der Wurzel, sofern nicht alle lediglich ein Suchen durchführen wollen. Ziel ist die Entwicklung eines Sperrprotokolls mit mehr potentieller Parallelität als 2PL, da die Semantik der Operationen *search*, *insert*, *delete* eines B-Baums bekannt ist.

Ein Knoten eines B-Baumes heißt **sicher** bzgl. Einfügen (Löschen), wenn er nicht ganz voll (bzw. mehr als halbvoll) ist. Sichere Knoten schränken den erforderlichen Restrukturierungsbereich ein.

Ein einfaches Sperrprotokoll für B-Bäume:

	Einfügen/Löschen	Suchen
(1)	Sperre die Wurzel	Sperre die Wurzel
(2)	Lese die Wurzel und mache sie aktuell	Lese die Wurzel und mache sie aktuell
(3)	Solange der aktuelle Knoten kein Blatt ist,	Solange der aktuelle Knoten Knoten kein Blatt ist,
(3.1)	Sperre betreff. Nachfolger	Sperre betreff. Nachfolger
(3.2)	Lese ihn und mache ihn aktuell	Gib Sperre des Vorgängers frei
(3.3)	Falls er sicher ist, gib alle Sperren der Vorgänger frei	Lese den betreff. Nachfolger und mache ihn aktuell
(4)	Führe Operation aus (inkl. Restrukturierung)	Führe Operation aus
(5)	Gib alle Sperren frei	Gib Sperre frei

(b) Inkrement- und Dekrement-Operationen

Inkrement und Dekrement eines Zählers sind kommutative Operationen; wir unterstellen, dass beide Operationen von Seiten des Datenbanksystems atomar ausgeführt werden, d.h. das bei Ausführung implizierte Lesen und Schreiben der eigentlichen Datenbank-Objekte ist nicht verzahnt.

Neben den bisher betrachteten L^R- und L-Sperren führen wir eine Inkrement-Sperre L^I und eine Dekrement-Sperre L^D ein und erweitern die bisherige Kompatibilitätsmatrix zu:

	L^R	L	L^I	L^D
L^R	J	N	N	N
L	N	N	N	N
L^I	N	N	J	J
L^D	N	N	J	J

Aufgabe 10.6:

Lösung: Betrachte den Schedule

$$S: \quad \begin{array}{l} T_1: \; LA \;\; RA \;\; WA \qquad\qquad\qquad\qquad\quad undo \\ T_2: \qquad\qquad\qquad LB \;\; RB \;\; WB \qquad\qquad LA \;\; RA \;\; WA \;\; UA \;\; UB \end{array}$$

Angenommen, A und B sind Tupel, die beide auf einer gemeinsamen physischen Seite liegen. A und B haben zu Beginn des Schedules den Wert 0 und werden durch die Transaktionen jeweils um 1 erhöht. T_1 wird mittels *undo* zurückgesetzt, während T_2 ihr normales Ende erreicht. Allerdings macht das *undo* von T_1 auch die Änderung von T_2 rückgängig; T_2 hat zwar eine Sperre zu B, der Konflikt zum *undo* von T_1 kann jedoch nicht erkannt werden. Es ergibt sich somit die Situation, dass im Endzustand $A = 1$ und $B = 0$ und T_2 ausgeführt. Es existiert kein serieller Schedule von T_1 (inkl. *undo*) und T_2, der einen solchen Endzustand erzeugt.

Aufgabe 10.7:

Lösung: Zunächst ein paar Eigenschaften des Verfahrens mit expliziten Tickets.

(1.) lokale Schedules und LTM
  a) lokale schedules sind serialisierbar (garantiert der LTM)
  b) alle Subtransaktionen globaler Transaktionen (Projektion einer solchen Transaktion auf eine Site *i*) beinhalten *take-a-ticket*-Operationen bezüglich des Zählers I_i
  c) diese *take-a-ticket*-Operationen implizieren eine Ordnung für die Subtransaktionen von globalen Transaktionen:

$$R_i(I)\ W_i(I+1)...R_j(I)\ W_j(I+1)$$
erzeugt Kante $T_i \rightarrow T_j$ *in* $CG(S)$

  d) diese Ordnung ist konsistent mit der globalen Serialisierung

(2.) globaler Schedule und GTM
  a) Ordnungen der S_i ergeben zusammen eine lineare Ordnung auf globalen Transaktionen (garantiert der GTM)

Zeige: S entstand aus dem explizite-Tickets-Verfahren $\Rightarrow$ S global c-serialisierbar. Beweis durch Widerspruch: Sei S nicht global c-serialisierbar. Dann folgt mit obigem Lemma, das $CG(S)$ zyklisch ist. Unterscheide 3 Fälle bezüglich der Art des Zyklus in $CG(S)$:

(1) Zyklus nur mit globalen Transaktionen: das widerspräche dem GTM (GTM garantiert eine lineare Ordnung auf den globalen Transaktionen)
(2) Zyklus nur mit lokalen Transaktionen: das widerspräche dem LTM (LTM garantiert c-serialisierbare Schedules)
(3) Zyklus mit lokalen und globalen Transaktionen: unterscheide weiter nach der Zahl der Partitionen mit globalen Transaktionen im Zyklus:
  (i) Es gibt nur eine Partition lediglich bestehend aus globalen Transaktionen (GTP_j):

$$LT_i \rightarrow GTP_j \rightarrow LT_k \rightarrow ... \rightarrow LT_i$$

  innerhalb der Partition sind alle globalen Transaktionen im CG verbunden, insbesondere die Aktionen globaler Transaktionen die auch auf dieser lokalen Site ablaufen, d.h. es gibt bereits einen Zyklus auf dieser Site. Diese würde der LTM aber unterbinden.
  (ii) Es gibt im Zyklus mehrere Partitionen, welche lediglich aus globalen Transaktionen (z.B. GTP_i, GTP_k) bestehen.

$GTP_i \rightarrow LTP_j \rightarrow GTP_j \rightarrow ... \rightarrow GTP_i$

Teile von GTP_i und GTP_k müssen sich auf der selben Site befinden. Transformiere das Problem durch Entfernen der lokalen Partition (o.b.d.A. auf Site l) LTP_j (unkritisch, da $GTP_i \rightarrow GTP_k \in CG(S_l)$ aufgrund der *take-a-ticket*-Operationen) und rekursiver Überprüfung des verbliebenen Zyklus (bis evtl. keine lokale Partition mehr vorhanden ist - dann widerspricht sich der Zyklus mit dem GTM. Der GTM garantiert eine lineare Ordnung auf den globalen Transaktionen).