Rückblick

Hashing

Betrachten Sie folgende Einträge der Tabelle *Airport*(<u>*Code*</u>, *City*):

Code	City
AAA	Anaa
...	...
ZZV	Zanesville

Geben Sie eine Hash-Funktion $h(K) = f(K)\ mod\ M$ an, die auf dem Primärschlüssel angewendet werden soll.

(i) Wählen Sie hierzu eine entsprechende Darstellung des Schlüssels $f(K)$ und das kleinstmögliche M, welches sicherstellt, dass es auch bei vollständiger Belegung des Schlüsselraums zu *keinen* Kollisionen kommen kann.

(ii) Wenden Sie Ihre Hash-Funktion auf die obigen beiden Einträge "AAA" und "ZZV" an.

Hashing

(i)
- Sei $K = k_1k_2k_3$ und sei $char(\text{A}) = 0, \ldots, char(\text{Z}) = 25$
- Wähle $f(K) = (char(k_1) * 26^0) + (char(k_2) * 26^1) + (char(k_3) * 26^2)$ und
- $M = 26^3 = 17576$.

(ii)
- $f(\text{AAA}) = ((0 * 26^0) + (0 * 26^1) + (0 * 26^2))\ mod\ 17576 = 0$
- $f(\text{ZZV}) = ((25 * 26^0) + (25 * 26^1) + (21 * 26^2))\ mod\ 17576 = 14871$
- $f(\text{ZZZ}) = ((25 * 26^0) + (25 * 26^1) + (25 * 26^2))\ mod\ 17576 = 17575$

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.
- *Mehrbenutzerkontrolle*: Isolation der einzelnen Transaktionen.
- *Fehlerbehandlung*: Atomizität und Dauerhaftigkeit einer Transaktion.

9.1 Grundlagen

- Eine Datenbank sei gegeben als eine Menge von Objekten.
  logische Größen: Relation, Tupel einer Relation,
  physische Größen: Blöcke, Seiten einer physischen Datenbank.
- Transaktionen T greifen lesend und schreibend zu den Objekten 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 $\mathcal{T}$.

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

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

Beispiel

- Sei $\mathcal{T} = \{T_1, T_2, T_3\}$, wobei $T_1 = R_1A\ W_1A\ R_1B\ W_1B$, $T_2 = R_2A\ W_2A\ R_2B\ W_2B$ und $T_3 = R_3A\ W_3B$.
- Es existieren sechs unterschiedliche serielle Schedule zu $\mathcal{T}$, beispielsweise $S_1 = h_1h_2h_3$, oder auch $S_2 = h_2h_3h_1$.
- Beispiele für nicht serielle Schedule sind:

$$\begin{aligned} S_3 &= R_1A\ W_1A\ R_3A\ R_1B\ W_1B\ R_2A\ W_2A\ W_3B\ R_2B\ W_2B, \\ S_4 &= R_3A\ R_1A\ W_1A\ R_1B\ W_1B\ R_2A\ W_2A\ R_2B\ W_2B\ W_3B. \end{aligned}$$

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 heißt *serialisierbar* genau dann, wenn zu ihm ein äquivalenter serieller Schedule derselben 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_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 jedoch.

Beispiel

- Seien $T_1 = R_1A\ W_1A\ R_1B\ W_1B$ und $T_2 = R_2A\ W_2A\ R_2B\ W_2B$.
- $$S_1 = R_1A\ W_1A\ R_2A\ W_2A\ R_2B\ W_2B\ R_1B\ W_1B$$

Schedule $S = R_1A\ W_1A\ R_2A\ W_2A\ R_2B\ W_2B\ R_1B\ W_1B$	
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 zu T_1 und T_2 gibt, so dass T_1 in diesem Schedule und in 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.

Abhängigkeitsgraph

Der *Abhängigkeitsgraph* von 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 ($i \neq j$):

- $\widehat{S} = \ldots R_iB \ldots W_jA \ldots \;\Rightarrow\; R_iB \to W_jA \in E$,
- $\widehat{S} = \ldots W_iA \ldots R_jA \ldots \;\Rightarrow\; W_iA \to R_jA \in E$, 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* von 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 ($i \neq j$):

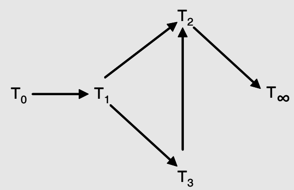
- $\widehat{S} = \ldots W_iA \ldots R_jA \ldots \;\Rightarrow\; T_i \to T_j \in E$, sofern zwischen W_iA und R_jA in $\widehat{S}$ keine weitere Schreiboperation zu A existiert. *(WR-Konflikt)*
- $\widehat{S} = \ldots W_iA \ldots W_jA \ldots \;\Rightarrow\; T_i \to T_j \in E$, sofern zwischen W_iA und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

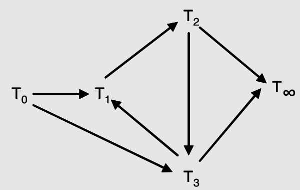
Satz und Definition

- 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 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*: angefordertes Privileg zu A: / gehaltenes 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 nun genau dann vor, wenn der Wartegraph einen Zyklus hat.

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

2-Phasen Sperren 2PL

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

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

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

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

Beispiel

$$
\begin{aligned}
T_1 &= L_1A\ R_1A\ L_1B\ U_1A\ W_1B\ U_1B,\\
T_2 &= L_2A\ R_2A\ W_2A\ U_2A,\\
T_3 &= L_3C\ R_3C\ U_3C.
\end{aligned}
$$

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

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 $\mathcal{T} = \{T_1, \ldots, T_n\}$, 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 der Form $T_1 \rightarrow T_2 \rightarrow \cdots \rightarrow T_k \rightarrow T_1$.

- Eine Kante $T \rightarrow T'$ eines solchen Zyklus setzt voraus, dass T und T' zu einem gemeinsamen Objekt A jeweils eine Operation ausführen, von denen mindestens eine schreibend ist.
- Da die Operation zu A in T und T' jeweils durch eine Sperr- und Freigabeoperation umfasst ist, kann T' seine Operation zu A erst nach der Freigabeoperation von T 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 vor l_2, ..., l_{k-1} vor l_k und l_k vor 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.

- 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:

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

Ausblick

Ausblick

Geben Sie einen Schedule S an, der

- konfliktserialisierbar,
- jedoch nicht bei Anwendung von 2PL entstehbar

ist.