Aufgabe 15

Ich definiere, dass M die Anzahl Seiten der Relation R darstellt, während N das gleiche für S ist:

$$M = 2000$$
$$N = 5000$$

Ebenso soll die Anzahl Tupel pro Seite in Variablen verpackt sein:

$$p_R = 20$$
$$p_S = 10$$

Ein einzelner Zugriff auf den Datenträger dauert im Mittel 10 ms:

$$IO = 10\ ms$$

a) Es ergeben sich zwei Möglichkeiten: entweder jedes Tupel aus R wird mit der Relation S verglichen oder jedes Tupel aus S mit der Relation S. Beide unterscheiden sich in ihrer Ausführungszeit um den Faktor 2:

$$\begin{aligned} t_{join(R,S)} &= (M + p_R \cdot M \cdot N) \cdot IO \\ &= (2000 + 20 \cdot 2000 \cdot 5000) \cdot 10\,ms \\ &\approx 555{,}6\,h \\ t_{join(S,R)} &= (N + p_S \cdot N \cdot M) \cdot IO \\ &= (5000 + 10 \cdot 5000 \cdot 2000) \cdot 10\,ms \\ &\approx 277{,}8\,h \end{aligned}$$

b) Aus der Aufgabenstellung wird nicht klar, welche Indexstruktur die Datenbank aufweist. Aus diesem Grunde gebe ich zwei Zeiten für jede mögliche join-Reihenfolge an, die erste für einen Hashindex (1,2 I/O Suchkosten pro Tupel), die zweite für einen B+ Baum (im Skript werden 2 bis 4 I/O erwähnt, ich nutze den Mittelwert 3 I/O). Für einen unclustered Index sind die Werte jeweils um einiges höher, die genaue Zeit ist stark abhängig von der Zusammensetzung der Datenbank, bei Gleichverteilung muss man den Faktor 5000/2000=2,5 verwenden:

$$\begin{aligned} t_{join(R,S)hashed} &= (M + 1{,}2 \cdot M \cdot p_R) \cdot IO \\ &= (2000 + 1{,}2 \cdot 2000 \cdot 20) \cdot 10\,ms \\ &= 8\,min\,20\,s \\ t_{join(R,S)B+} &= (M + 3 \cdot M \cdot p_R) \cdot IO \\ &= (2000 + 3 \cdot 2000 \cdot 20) \cdot 10\,ms \\ &= 20\,min\,20\,s \\ t_{join(S,R)hashed} &= (N + 1{,}2 \cdot N \cdot p_s) \cdot IO \\ &= (5000 + 1{,}2 \cdot 5000 \cdot 10) \cdot 10\,ms \\ &= 10\,min\,50\,s \\ t_{join(S,R)B+} &= (N + 3 \cdot N \cdot p_s) \cdot IO \\ &= (5000 + 3 \cdot 5000 \cdot 10) \cdot 10\,ms \\ &= 25\,min\,50\,s \end{aligned}$$

c) Die Dauer für einen *Equi-Join* ist unabhängig von der Reihenfolge der beiden Relationen:

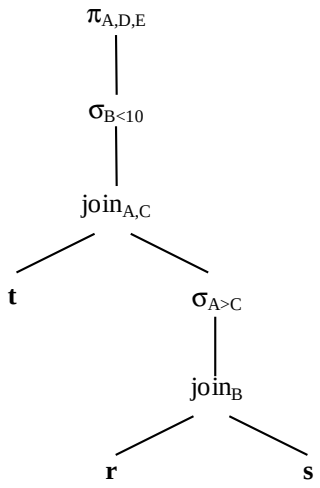
$$\begin{aligned} t &= 3 \cdot (M + N) \cdot IO \\ &= 3 \cdot (2000 + 5000) \cdot 10\,ms \\ &= 3\,min\,30\,s \end{aligned}$$

d) Der Puffer ist für R ausreichend groß dimensioniert. In diesem Fall ist der *Block Nested Loops Join* die optimale Methode mit einer Laufzeit von:

$$\begin{aligned} t_{blockjoin(R,S)} &= \\ t_{blockjoin(S,R)} &= (M + N) \cdot IO \\ &= (2000 + 5000) \cdot 10\,ms \\ &= 1\,min\,10\,s \end{aligned}$$

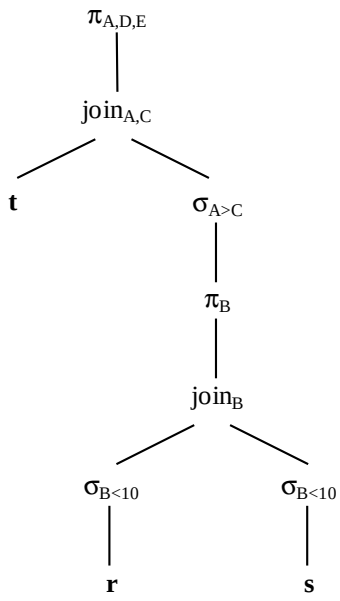
Aufgabe 16

a) Ausgangsbaum:

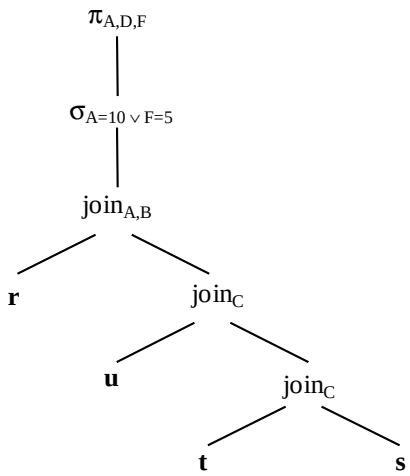


Die Selektion B<10 ist problemlos jeweils einzeln auf die beiden benutzten Tabellen **r** und **s** anwendbar und wird deshalb im Baum ganz nach unten geschoben. Unmittelbar nach dem Join auf **r** und **s** wird das Attribut B nicht mehr benötigt. Ich führe deshalb eine zusätzliche Projektion ein, die die im Baum weiter oben stehenden Operationen beschleunigen soll.

Optimierter Baum:

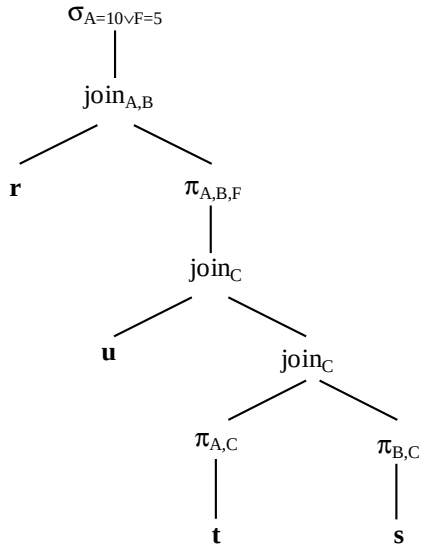


b) Ausgangsbaum:

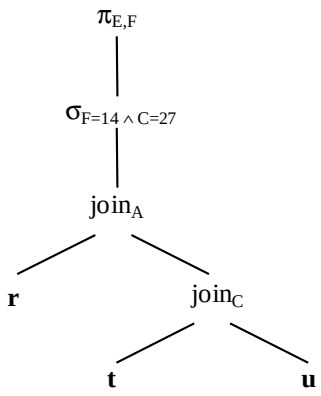


Die Selektion bezieht sich auf A, welches in den Relationen **r** und **t** auftaucht, und F, das man nur in **u** findet. Leider ist durch die Oder-Verknüpfung der Bedingungen keine Verschiebung nach unten möglich. Die Attribute D und E sind von keiner Bedeutung, ich blende sie daher durch neu eingeführte Projektionen an den Blättern aus. Sobald **s**, **t** und **u** verknüpft wurden, ist auch C irrelevant.

Optimierter Baum:

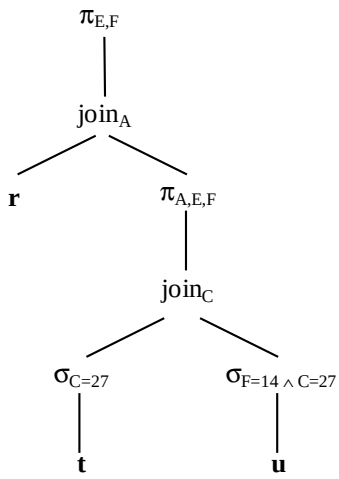


c) Ausgangsbaum:

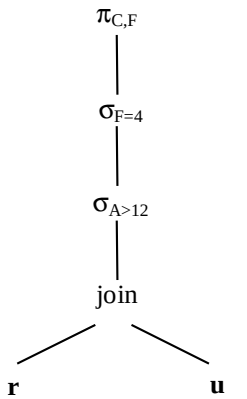


Da die Selektion beide Bedingungen durch ein Und verknüpft, kann sie problemlos zerlegt werden. Dies nutze ich aus und wende sie direkt auf **u** und **t** an, bevor diese gejoint werden. Nach dem Join ist das Attribut C nicht mehr von Relevanz, eine Projektion blendet es aus.

Optimierter Baum:



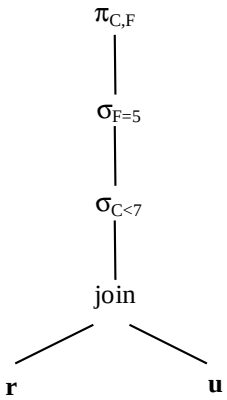
d) Ausgangsbaum:



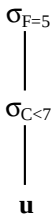
Die letzte Projektion enthält nur Attribute aus u. Da keine Selektion sich auf die beiden Ausgangsrelationen **r** und **u** bezieht, ist **r** überflüssig. Demzufolge kann man auch die Selektion auf A und die Projektion entfallen.

Optimierter Baum:

e) Ausgangsbaum:



Eigentlich können alle Aussagen aus Aufgabe d) wiederholt werden, die einzige Ausnahme besteht darin, dass *beide* Selektionen bestehen bleiben müssen.
Man kann überlegen, ob es Sinn macht, die beiden Selektionen zu vertauschen, da ein Test auf Gleichheit i.d.R. restriktiver als eine Ungleichung ist.



Aufgabe 17

a) Zunächst bestimme Reads-From- und Live-Reads-From-Relation von s_1:

$$s_1 = r_1(x)w_1(x)r_2(x)r_2(y)w_2(y)c_2w_1(y)c_1$$
$$RF(s_1) = \{(t_0,x,t_1),(t_1,x,t_2),(t_1,x,t_\infty),(t_0,y,t_2),(t_1,y,t_\infty)\}$$
$$LRF(s_1) = \{(t_0,x,t_1),(t_1,x,t_\infty),(t_1,y,t_\infty)\}$$

Es treten zwei nebenläufige Transaktionen auf, demzufolge lassen sich zwei serielle Schedules bilden:

$$t_1t_2 = r_1(x)w_1(x)w_1(y)c_1r_2(x)r_2(y)w_2(y)c_2$$
$$t_2t_1 = r_2(x)r_2(y)w_2(y)c_2r_1(x)w_1(x)w_1(y)c_1$$
$$RF(t_1t_2) = \{(t_0,x,t_1),(t_1,x,t_2),(t_1,x,t_\infty),(t_1,y,t_2),(t_2,y,t_\infty)\}$$
$$RF(t_2t_1) = \{(t_0,x,t_2),(t_0,y,t_2),(t_0,x,t_1),(t_1,x,t_\infty),(t_1,y,t_\infty)\}$$
$$LRF(t_1t_2) = \{(t_0,x,t_1),(t_1,x,t_\infty),(t_1,y,t_2),(t_2,y,t_\infty)\}$$
$$LRF(t_2t_1) = \{(t_0,x,t_1),(t_1,x,t_\infty),(t_1,y,t_\infty)\}$$

Der Schedule gehört zur Klasse VSR (View Serializability), da

$$LRF(t_2t_1) = LRF(s_1)$$

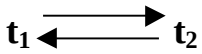
aber nicht FSR (Final State Serializability) wegen:

$$(t_0,y,t_2) \in RF(s_1)\ aber\ (t_0,y,t_2) \notin RF(t_1t_2)\ bzw.$$
$$(t_1,x,t_2) \in RF(s_1)\ aber\ (t_1,x,t_2) \notin RF(t_2t_1)$$

Die Konflikte sind:

$$conf(s_1) = \{(w_1(x),r_2(x)),(r_2(y),w_1(y),(w_2(y),w_1(y)))\}$$

Leider ist der Konfliktgraph zyklisch, daher ist s_1 nicht Bestandteil der Klasse CSR (Conflict Serializability):



Aufgrund der Regel

$$CSR \subset VSR \subset FSR$$

hätte ich die Untersuchung abbrechen können, als klar war, dass s_1 nicht in VSR ist und somit auch nicht CSR sein kann.. Ich nutzte diese Aufgabe aber, um mir selbst den Algorithmus weiter zu verdeutlichen und mein Verständnis dafür zu schärfen.

b) Der erste Schritt besteht erneut in der Bestimmung der Reads-From- und Live-Reads-From-Relation:

$$s_2 = r_1(y)r_3(z)r_2(y)w_1(y)w_1(x)c_1w_2(x)w_2(u)c_2w_3(x)c_3$$
$$RF(s_2) = \{(t_3,x,t_\infty),(t_0,y,t_1),(t_0,y,t_2),(t_1,y,t_\infty),(t_0,z,t_3),(t_0,z,t_\infty),(t_2,u,t_\infty)\}$$
$$LRF(s_2) = \{(t_3,x,t_\infty),(t_0,y,t_1),(t_1,y,t_\infty),(t_0,z,t_\infty),(t_2,u,t_\infty)\}$$

Diesmal sind drei Transaktionen in der Schedule vorhanden, demzufolge existieren 3!=6 serielle Schedules:

$$t_1t_2t_3 = r_1(y)w_1(y)w_1(x)c_1r_2(y)w_2(x)w_2(u)c_2r_3(z)w_3(x)c_3$$
$$t_1t_3t_2 = r_1(y)w_1(y)w_1(x)c_1r_3(z)w_3(x)c_3r_2(y)w_2(x)w_2(u)c_2$$
$$t_2t_1t_3 = r_2(y)w_2(x)w_2(u)c_2r_1(y)w_1(y)w_1(x)c_1r_3(z)w_3(x)c_3$$
$$t_2t_3t_1 = r_2(y)w_2(x)w_2(u)c_2r_3(z)w_3(x)c_3r_1(y)w_1(y)w_1(x)c_1$$
$$t_3t_1t_2 = r_3(z)w_3(x)c_3r_1(y)w_1(y)w_1(x)c_1r_2(y)w_2(x)w_2(u)c_2$$
$$t_3t_2t_1 = r_3(z)w_3(x)c_3r_2(y)w_2(x)w_2(u)c_2r_1(y)w_1(y)w_1(x)c_1$$

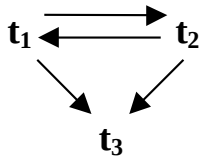
Für diese müssten jeweils die Reads-From- und Live-Reads-From-Relationen ermittelt werden. Sobald ich jedoch die erste Übereinstimmung finde, kann ich abbrechen:

$$RF(t_1t_2t_3) = \{(t_3,x,t_\infty),(t_0,y,t_1),(t_1,y,t_2),(t_1,y,t_\infty),(t_0,z,t_3),(t_0,z,t_\infty),(t_2,u,t_\infty)\}$$
$$RF(t_1t_3t_2) = \{(t_2,x,t_\infty),(t_0,y,t_1),(t_1,y,t_2),(t_1,y,t_\infty),(t_0,z,t_3),(t_0,z,t_\infty),(t_2,u,t_\infty)\}$$
$$RF(t_2t_1t_3) = \{(t_3,x,t_\infty),(t_0,y,t_2),(t_0,y,t_1),(t_1,y,t_\infty),(t_0,z,t_3),(t_0,z,t_\infty),(t_2,u,t_\infty)\}$$
$$\Rightarrow \textit{Übereinstimmung } (VSR)$$
$$LRF(t_2t_1t_3) = \{(t_3,x,t_\infty),(t_0,y,t_1),(t_1,y,t_\infty),(t_0,z,t_\infty),(t_2,u,t_\infty)\}$$
$$\Rightarrow \textit{Übereinstimmung } (FSR)$$

Der Schedule s_2 gehört sowohl sowohl zur Klasse VSR als auch zur Klasse FSR, da ich ein seriellen Schedule finden konnte, der die gleiche Reads-From- bzw. Live-Reads-From-Relation aufweist. Abschließend überprüfe ich noch CSR:

$$conf(s_2) = \{(w_1(x),w_2(x)),(w_1(x),w_3(x)),(w_2(x),w_3(x)),(r_2(y),w_1(y))\}$$

Im dazugehörigen Graphen tritt leider ein Zyklus zwischen t_1 und t_2 auf, sodass s_2 nicht zur Klasse CSR gehört:



c) Zuerst entferne ich alle zur abgebrochenen Transaktion t_4 gehörenden Zugriffe:

$$s_3 = w_1(x)r_2(x)w_3(y)c_3w_1(y)c_1c_2$$

Wie üblich als nächstes die Reads-From- und Live-Reads-From-Relation von s_3:

$$\begin{aligned} s_3 &= w_1(x)r_2(x)w_3(y)c_3w_1(y)c_1c_2 \\ RF(s_3) &= \{(t_0,x,t_2),(t_1,x,t_\infty),(t_1,y,t_\infty)\} \\ LRF(s_3) &= \{(t_1,x,t_\infty),(t_1,y,t_\infty)\} \end{aligned}$$

Alle 6 möglichen seriellen Schedules:

$$\begin{aligned} t_1t_2t_3 &= w_1(x)w_1(y)c_1r_2(x)c_2w_3(y)c_3 \\ t_1t_3t_2 &= w_1(x)w_1(y)c_1w_3(y)c_3r_2(x)c_2 \\ t_2t_1t_3 &= r_2(x)c_2w_1(x)w_1(y)c_1w_3(y)c_3 \\ t_2t_3t_1 &= r_2(x)c_2w_3(y)c_3w_1(x)w_1(y)c_1 \\ t_3t_1t_2 &= w_3(y)c_3w_1(x)w_1(y)c_1r_2(x)c_2 \\ t_3t_2t_1 &= w_3(y)c_3r_2(x)c_2w_1(x)w_1(y)c_1 \end{aligned}$$

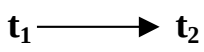
Und ihre Reads-From- bzw. Live-Read-From-Relationen:

$$\begin{aligned} RF(t_1t_2t_3) &= \{(t_1,x,t_2),(t_1,x,t_\infty),(t_3,y,t_\infty)\} \\ RF(t_1t_3t_2) &= \{(t_1,x,t_2),(t_1,x,t_\infty),(t_3,y,t_\infty)\} \\ RF(t_2t_1t_3) &= \{(t_0,x,t_2),(t_1,x,t_\infty),(t_3,y,t_\infty)\} \\ RF(t_2t_3t_1) &= \{(t_0,x,t_2),(t_1,x,t_\infty),(t_1,y,t_\infty)\} \\ &\Rightarrow \ddot{U}bereinstimmung\ (VSR) \\ LRF(t_2t_3t_1) &= \{(t_1,x,t_\infty),(t_1,y,t_\infty)\} \\ &\Rightarrow \ddot{U}bereinstimmung\ (FSR) \end{aligned}$$

Erneut konnte ich einen passenden seriellen Schedule finden, sodass s_3 sowohl FSR als auch VSR ist. Jetzt die Untersuchung auf CSR:

$$conf(s_3) = \{(w_1(x), r_2(x))\}$$

Der Graph ist azyklisch und s_3 damit in CSR:



Aufgabe 18

a) Der Schedule s_1:

$$s_1 = w_1(x) r_2(x) w_2(y) w_1(y) c_1 c_2 w_3(x) w_3(y)$$

s_1 ist RC (Recoverable), da t_2 und t_3 nur von t_1 abhängen und dessen Commit c_1 zuerst erfolgt. Leider liest t_2 den Wert x von t_1 zu einem Zeitpunkt, wo c_1 noch nicht erfolgt ist, d.h. s_1 ist nicht in ACA (Avoiding Cascading Aborts). Wegen

$$ST \subset ACA \subset RC$$

kann s_1 auch nicht ST (Strict) sein.

b) Für den Schedule s_2:

$$\begin{aligned} s_2 &= w_1(x) r_4(x) w_4(y) r_2(x) w_3(y) c_3 w_4(z) a_4 w_1(y) c_1 c_2 \\ &= w_1(x) r_2(x) w_3(y) c_3 w_1(y) c_1 c_2 \end{aligned}$$

ist RC gegeben, da nur t_2 von t_1 abhängt und c_1 vor c_2 geschieht. Erneut kann ACA aber nicht garantiert werden, da t_2 auf x zu einem Zeitpunkt zugreift, wo t_1 noch nicht committed wurde. Demzufolge gilt ST nicht.

c) s_3:

$$s_3 = r_1(x) w_1(x) r_2(y) r_1(y) w_3(z) w_2(y) c_2 r_1(z) w_1(y) w_3(x) c_1 c_3$$

ist nicht RC, da t_1 lesend auf z zugreift, welches von t_3 bearbeitet wurde, aber c_1 vor c_3 erfolgt. Schedule s_3 ist damit weder ACA noch ST.

d) Schedule s_4:

$$s_4 = r_1(x) w_1(x) r_2(y) r_1(y) w_3(z) w_2(y) c_2 r_1(z) w_1(y) c_1 w_3(x) c_3$$

erfüllt die gleichen Bedingungen wie s_3 und ist aus diesem Grunde auch weder RC noch ACA oder ST.