Prof. Dr.-Ing. Dr. h. c. T. Härder
Fachbereich Informatik
Arbeitsgruppe Datenbanken und Informationssysteme
Universität Kaiserslautern

Übungsblatt 3 – Lösungsvorschläge

für die freiwillige Übung

Unterlagen zur Vorlesung: „wwwdvs.informatik.uni-kl.de/courses/DBSREAL/“

Aufgabe 1: *Hot-Set* Modell

186

Es seien 3 Relationen R1, R2 und R3 mit 10, 20 bzw. 30 Seiten gegeben. Jede Seite beinhalte 10 Tupel. Zur Berechnung der beiden 1:n-*Joins* ((R1 |X| R2) |X| R3) werde ein *Nested-Loop-Join* benutzt. Die Ergebnisse der *Joins* werden in Zwischenrelationen mit 20 bzw. 30 Seiten abgelegt.

```
foreach i in R2
    suche j in R1
    konstruiere i x j
    lege Ergebnis in R’ ab
end
foreach i in R3
    ... (wie oben mit R’ statt mit R1, Ergebnis in R’’)
end
```

Welchen Verlauf nimmt die Fehlseitenhäufigkeit in Abhängigkeit von der Anzahl der verfügbaren Pufferrahmen, wenn für die Seitenersetzung das LRU-Verfahren gewählt wird?

Es stehen 1-11 Seiten zur Verfügung:

(jeder Seitenzugriff verursacht einen Page Fault)	
200-mal lesen einer Seite aus R2	
dazu jedesmal 10 Seiten aus R1 lesen	
jedes Ergebnis in einer Seite von R’ ablegen	200 * (10 + 1) + 200
300-mal lesen einer Seite aus R3	
dazu jedesmal 20 Seiten aus R’ lesen	
jedes Ergebnis in einer Seite von R’’ ablegen	300 x (20 + 1) + 300
	9000 Seitenersetzungen

Es stehen 12-21 Seiten zur Verfügung:

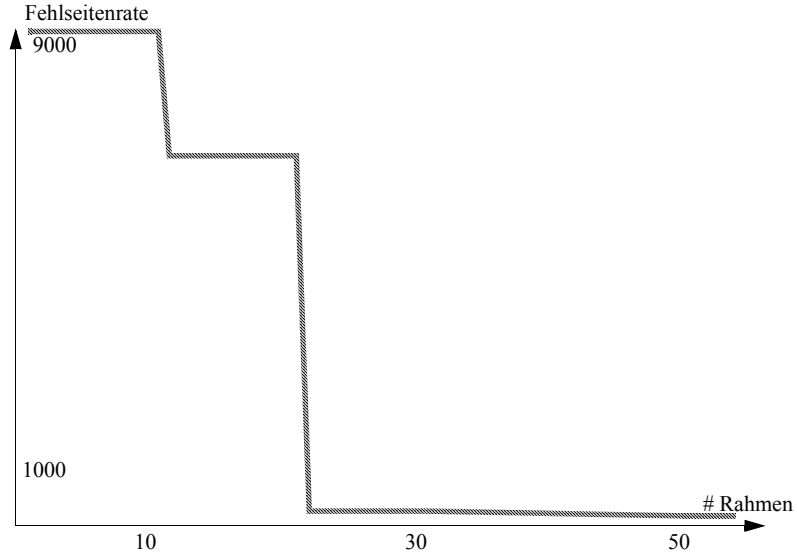
die 10 Seiten aus R1 bleiben in der ersten Schleife im Puffer	
10 Seitenfehler bei R1	
20 Seitenfehler bei R2	
20 Seitenfehler bei R’	50
300-mal lesen einer Seite aus R3	
dazu jedesmal 20 Seiten aus R’ lesen	
jedes Ergebnis in einer Seite von R’’ ablegen	300 x (20 + 1) +300
	6650 Seitenersetzungen

Es stehen 22-51 Seiten zur Verfügung:

erste Schleife wie oben	50
20 Seiten aus R’ bleiben im Puffer, müssen aber	
noch eingelesen werden	20 + 30 + 30
	130 Seitenersetzungen

Es stehen mehr als 51 Seiten zur Verfügung:

erste Schleife wie oben	50
zweite Schleife wie oben, nur R’ schon im Puffer	30 + 30
	110 Seitenersetzungen



Aufgabe 2: Rekursives Laden einer Tabelle beim DBCache-Ansatz

Tabelle CUST habe u.a. zwei UNIQUE-Spalten Cno und Cid sowie zwei NON-UNIQUE-Spalten CType und CLoc. Als Cache Keys seien CType und CLoc definiert.

Die Belegung im BE sei

BE_CUST	Cno	CType	CLoc	Cid ...
	789	silver	SF	NULL
	891	silver	LA	d07
	333	platinum	SJ	a21
	444	unspec.	LA	a07
	123	gold	SJ	a14
	456	gold	SF	b21
	555	gold	NY	c17
	666	bronze	Chi	a49
	999	NULL	Chi	a54
	001	bronze	NULL	c18

a) Wie verläuft das erstmalige Laden bei FE-CUST, wenn Q1 das Prädikat (CType = ‘platinum’) auswertet?

FE_CUST	Cno	CType	CLoc	Cid ...
①	333	platinum	SJ	a21
②	123	gold	SJ	a14
	456	gold	SF	621
③	555	gold	NY	c17
④	789	silver	SF	NULL
⑤	891	silver	LA	d07
⑥	444	un spec.	LA	a07

Die Nummern geben die Reihenfolge des Ladevorgangs an.

b) Wo wird Q2 mit (Cid = ‘a21’) ausgewertet?

In FE-CUST!
Cid ist als U-Spalte implizit ‘domain complete’.Da der Wert a21 in FE-CUST gefunden wird, kann Q2 dort ausgewertet werden.

c) FE-CUST sei leer. Wie sieht die Belegung nach Auswertung von Q2 mit (Cid = ‘a21’) aus?
Q2 wird in BE-CUST ausgewertet. FE-CUST bleibt leer!

d) Was bewirkt Anfrage Q3 mit (CLoc = ‘Chi’) bei der Belegung nach Ausführung von c) ?

FE_CUST	Cno	CType	CLoc	Cid ...
①	666	bronze	Chi	a45
	999	NULL	Chi	a54
②	001	bronze	NULL	c18

Die Nummern geben die Reihenfolge des Ladevorgangs an.

Um rekursives Laden einer Cache-Tabelle zu vermeiden, darf nur ein Cache Key vom Typ NU sein!

Aufgabe 3: Laden einer Cache Group

Tabelle CUST (C) habe u.a. zwei Spalten Cno und Cid vom Typ UNIQUE (U) sowie zwei Spalten CType und CLoc vom Typ NON UNIQUE (NU).

Tabelle ORDER (O) habe u. a. die U-Spalte Oid und die NU-Spalte Cno.

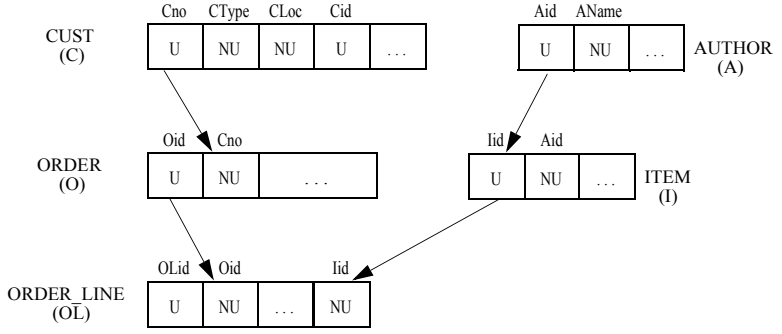
Tabelle ORDER_LINE (OL) habe u.a. die U-Spalte OLid und die NU-Spalten Oid und Iid.

Tabelle AUTHOR habe u.a. die U-Spalte Aid und eine NU-Spalte AName, während Tabelle ITEM die U-Spalte Iid und die NU-Spalte Aid besitzt.

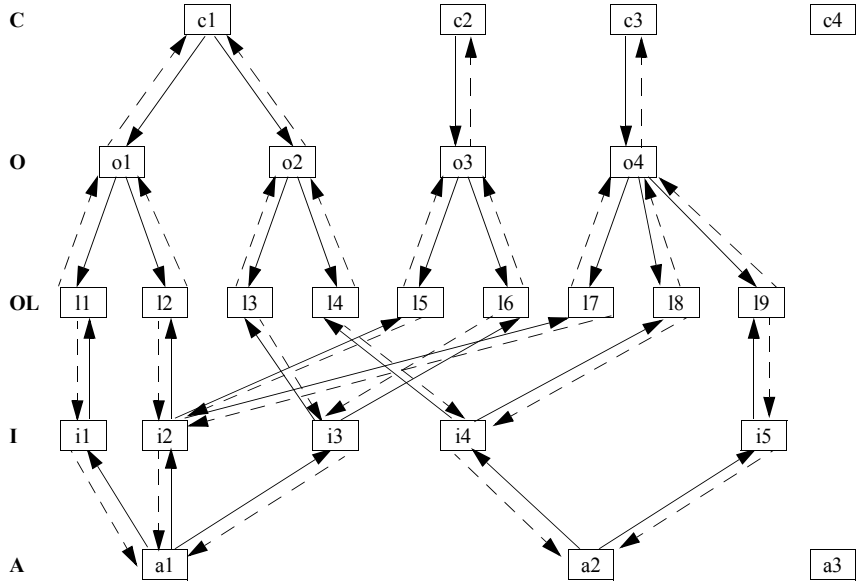
Im Backend-DB-Server (BE) seien für diesen Schemaausschnitt die referentiellen Integritätsbedingungen C.Cno → O.Cno und O.Oid → OL.Oid (als Primär-/Fremdschlüsselbeziehungen) definiert.

Weitere referentielle Integritätsbedingungen sind
A.Aid → I.Aid und I.Iid → OL.Iid.

Graphisch lässt sich dieser Schemaausschnitt wie folgt darstellen:



Ein Ausschnitt aus der BE-DB sei als folgende Graphstruktur veranschaulicht:



Der Pfeiltyp → veranschaulicht die (1:n)-Richtung und der Pfeiltyp - -> die (n:1)-Richtung der referentiellen Integritätsbedingung. Die Pfeile repräsentieren nur die Wertegleichheit von PS/FS oder FS/PS und sind nur eine verkürzende Darstellung im Vergleich zu Tabellen, die mit den entsprechenden Werten belegt sind.

Die Primärschlüsselwerte in C.Cno seien 123, 456, 789, 012 für c1, c2, c3 und c4.

c1 habe den Wert ‘gold’ und c3 den Wert ‘platinum’ in C.CType, während c2 und c4 den Wert ‘silver’ in C.CType besitzen.

Die Primärschlüsselwerte in A.Aid seien ‘a47’ in a1, ‘a53’ in a2 und ’a62’ in a3.

a1 habe den Wert ‘mayr’ in A.AName, während a2 und a3 ‘codd’ in A.AName haben.

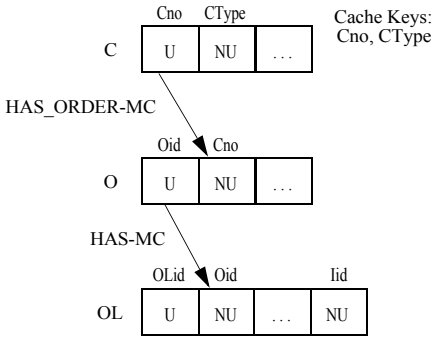
Im Frontend-DB-Server (FE) sei die Wurzel-Tabelle FE-C einer Cache Group definiert durch die Cache Keys C.Cno und C.CType. Weiterhin habe Tabelle FE-A einen Cache Key A.AName. Als RCCs (referential cache constraints) lassen sich beispielsweise C.Cno → O.Cno, O.Oid → OL.Oid, A.Aid → I.Aid und I.Iid → OL.Iid spezifizieren. Diese RCCs sind vom Type U → NU und werden auch als Member-Constraints (MC) bezeichnet. Solche MCs gewährleisten, dass für jeden Owner einer solcher (1:n)-Beziehung alle zugehörigen Member sich im Cache befinden. Die dazu umgekehrte Beziehung vom Typ NU → U heißt auch Owner-Constraint (OC), was besagt, dass, sobald sich ein Member dieser Beziehung im Cache befindet, auch der zugehörige Owner sich im Cache befinden muss.

Beispiel: Die RCCs C.Cnr → O.Cno, O.Oid → OL.Oid usw. sind Member-Constraints und können

mit HAS_ORDER-MC bzw. HAS-MC bezeichnet werden. Die RCCs O.Cno → C.Cno und OL.Oid → O.Oid dagegen sind Owner-Constraints und lassen sich mit HAS_ORDER-OC bzw. HAS-OC bezeichnen.

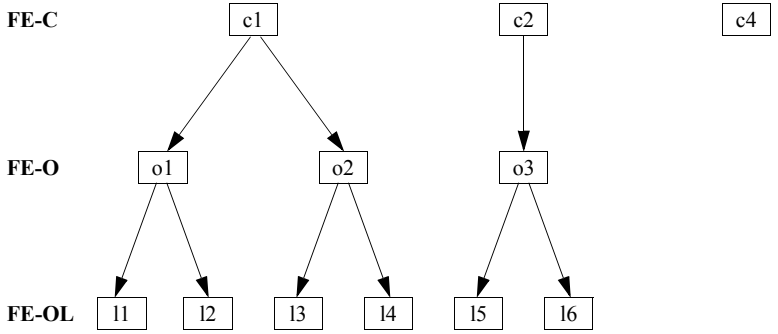
Im folgenden gehen wir immer von einem leeren Cache in der FE-DB aus.

a) Die Cache Group G1 sei wie folgt definiert:



In Anfrage Q1 werde das Prädikat (Cno = 123) verwendet. In einer zweiten Anfrage Q2 wird nach (CType = ‘silver’) gesucht.

Welche Belegung von G1 stellt sich nach Ausführung beider Anfragen ein?



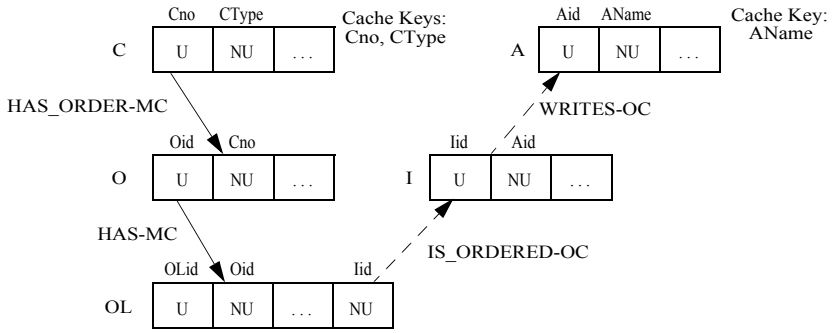
Geben Sie Beispiele für Verbundprädikate, die in der FE-DB ausgewertet werden können.

- C.Cno = 123 AND C.Cno = O.Cno AND O.Oid = OL.Oid

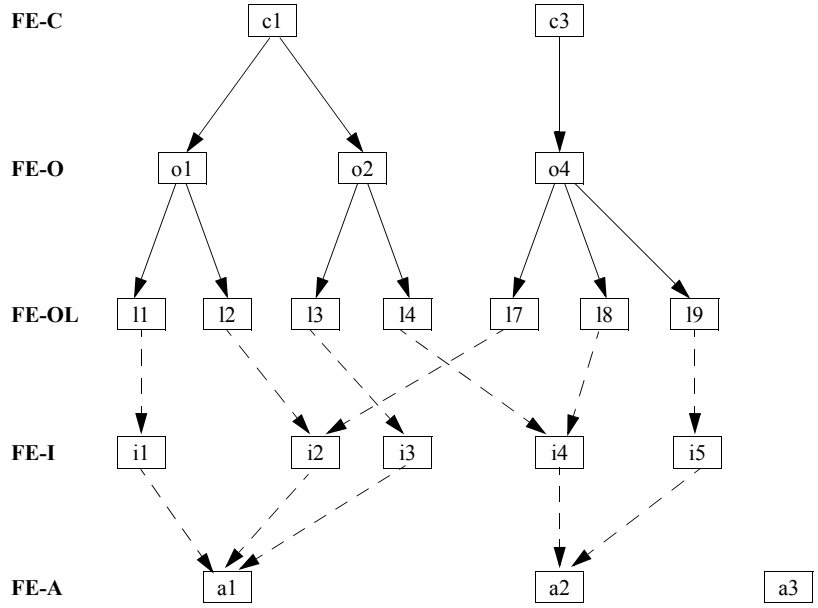
- C.CType = ‘silver’ AND C.Cno = O.Cno

- ...

b) Cache Group G2 sei wie folgt definiert:



Q3 referenziert (C.CType = ‘gold’). Wie sieht die Belegung von G2 nach Ausführung von Q3 aus?

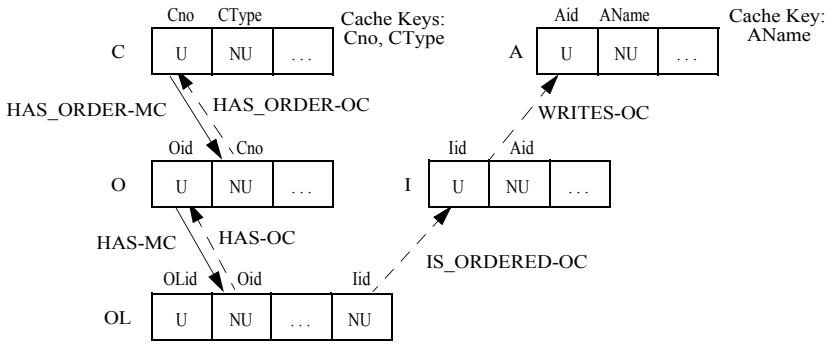


In dieser Situation referenziert Q4 (A.AName = ‘codd’).

Was ändert sich an der Belegung von G2?

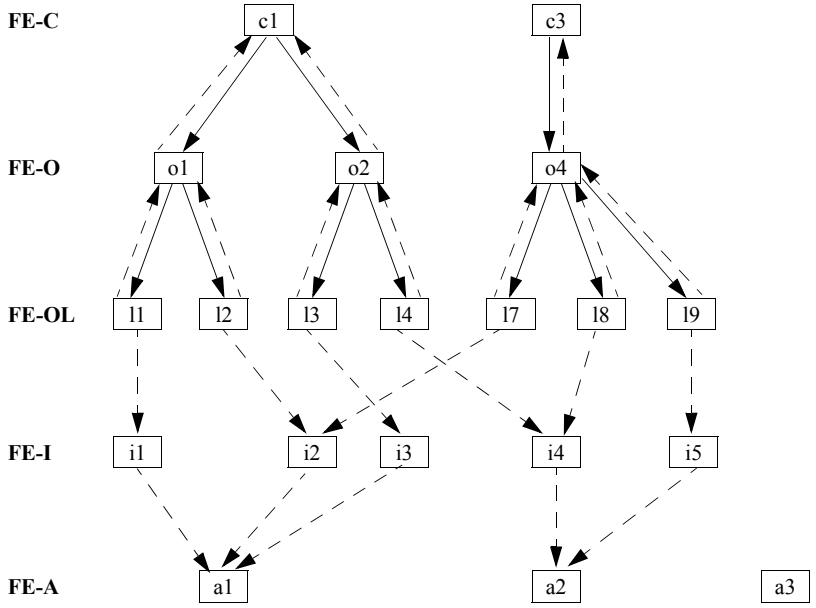
Nichts!
Q4 kann im Cache beantwortet werden.

c) Cache Group G3 sei wie folgt definiert:

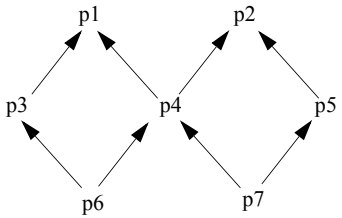


Wie sieht die Belegung von G3 nach Ausführung von Q3 und Q4 aus?

G3 entspricht G2. Die Beziehungen HAS-OC und HAS_ORDER-OC sind hier redundant, weil eine wertebasierte Beziehung in beiden Richtungen durchlaufen werden kann.

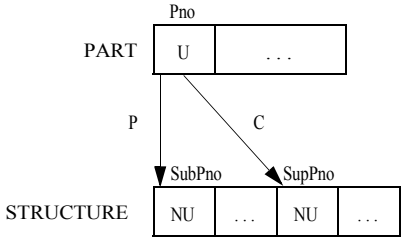


Aufgabe 4: Rekursives Laden von Cache Groups
Der Gozinto-Graph GG1 einer Stückliste sei



Die Pfeilrichtung entspricht der Beziehung PART_OF

Im Relationenmodell lässt sich eine Stückliste wie folgt modellieren, wobei ein Pfeil eine PS/FS-Beziehung (referentielle Integritätsbedingung) auf Typebene ausdrückt:



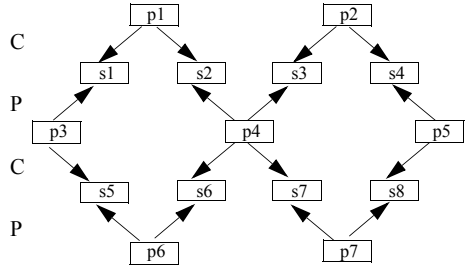
C = COMPOSED_OF

P = PART_OF

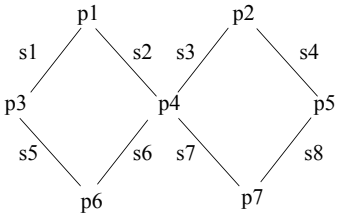
Gemäß dieses Schemas sei die obige Stückliste GG1 in der Backend-DB (BE-DB) in den Tabellen BE-PART und BE-STRUCTURE gespeichert.

Für GG1 wählen wir eine Graphdarstellung, die sich als kompakte Alternativform der Tabellendarstellung für PART und STRUCTURE auffassen lässt. GG1 lässt sich nach obigem Schema folgendermaßen veranschaulichen, wobei jeweils die PS/FS-Beziehungen zwischen den Sätzen (Tupel) als Pfeile dargestellt sind. Die Kantenbezeichnungen wurden nach folgender Darstellung gewählt:

GG1 als Graph
mit PS/FS-Beziehungen für C und P



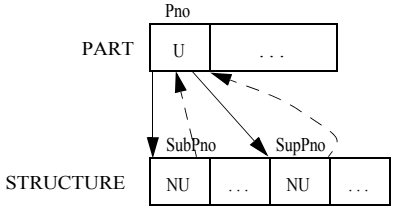
Kantenbezeichnungen



Im Cache sei der Cache Key Pno für FE-PART definiert.

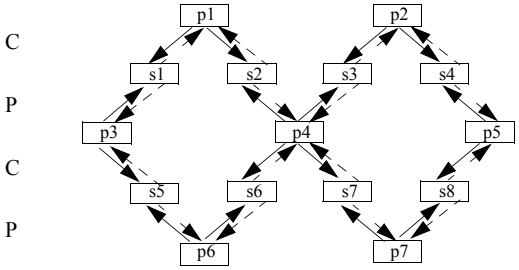
Das Laden des Cache in der FE-DB beginnt immer bei leeren Tabellen FE-PART und FE-STRUCTURE.

a) Gegeben sei folgende Definition der Cache Group G1:



CK: Pno

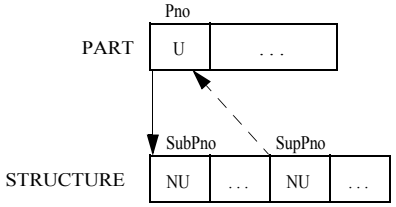
Wie sieht die Belegung von G1 aus, wenn in Anfrage Q1 ‘Pno = 1’ (p1) referenziert wurde?



Welche Belegung hätte sich bei einer Referenz von p4 (‘Pno = 4’) oder p6 (‘Pno = 6’) ergeben?

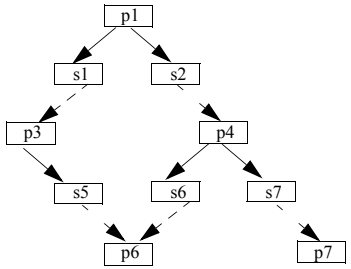
Bei Referenz irgendeines Cache Keys wird immer die gesamte GG1-Struktur geladen.

b) Gegeben sei die nachfolgende Definition von G2:

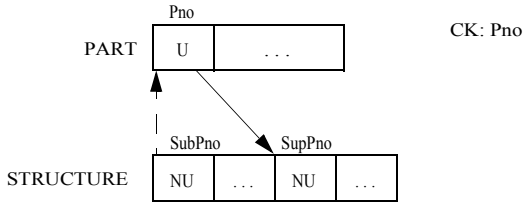


CK: Pno

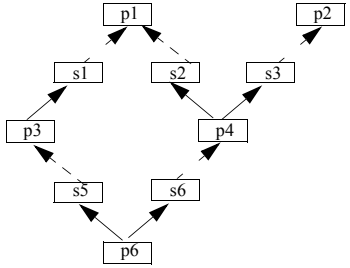
Wie sieht die Cache-Belegung aus, wenn ‘Pno = 1’ (p1) referenziert wurde?



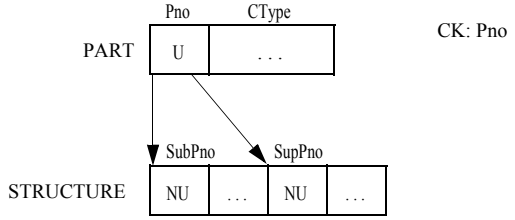
c) Gegeben sei die nachfolgende Definition von G3:



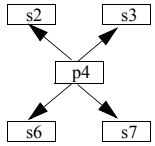
Wie sieht die Cache-Belegung aus, wenn ‘Pno = 6’ (p6) referenziert wurde?



d) Gegeben sei die nachfolgende Definition von G4:



Wie sieht die Cache-Belegung aus, wenn ‘Pno = 4’ (p4) referenziert wurde?



e) Was ist eine notwendige Bedingung, dass in einer Cache Group G rekursives Laden vermieden wird?

Es dürfen in G keine heterogenen RCC-Zyklen, in denen verschiedene Spalten in **einer** Tabelle referenziert werden, auftreten.
In Cache Groups mit homogenen RCC-Zyklen setzt sich das Laden nicht rekursiv fort.

Def.:

- A homogeneous RCC cycle in a cache group is formed by a path where only a single column per table is involved. As a consequence of this definition, for each referential cache constraint in the cycle, its reverse referential cache constraint holds.
- A heterogeneous RCC cycle is formed by a path where in at least one participating table more than one column is involved.