Prof. Dr. 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:

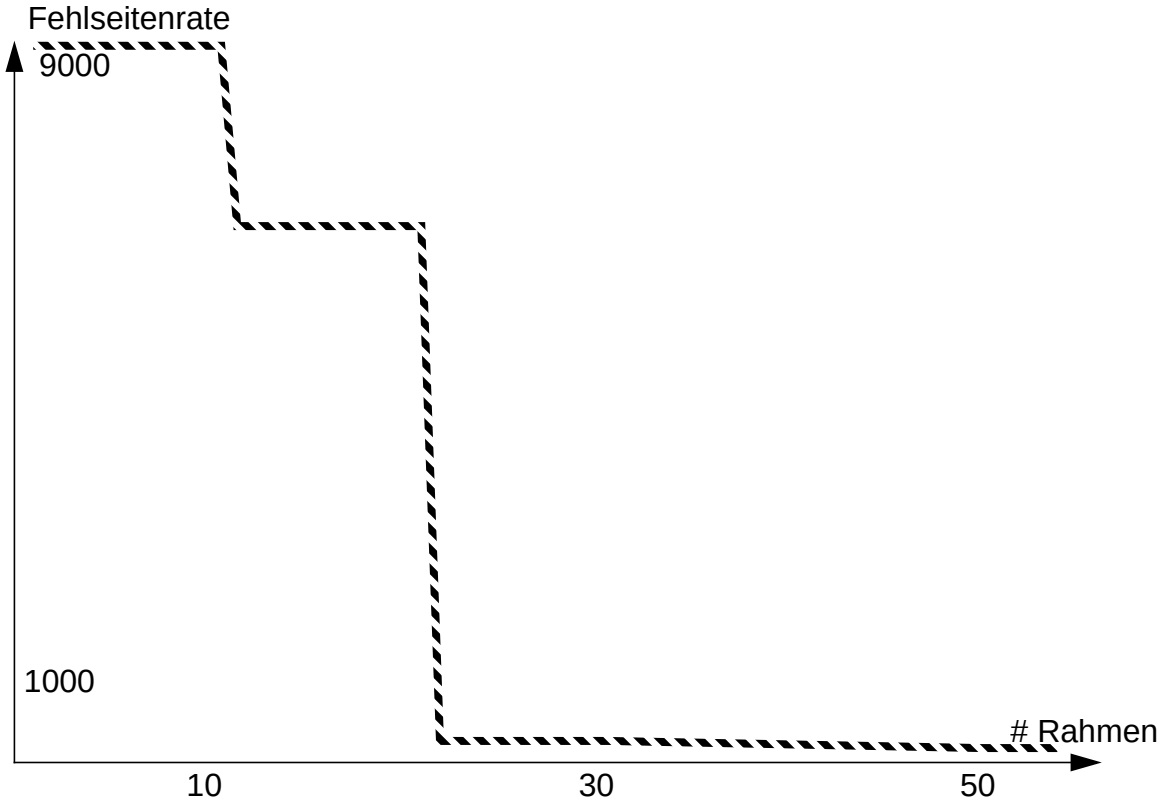
die 10 Seiten aus R1 bleiben in der ersten Schleife im Puffer	
10 Seitenfehler bei R1	
20 Seitenfehler bei R2	
20 Seitenfehler bei R’	50
300-mal lesen einer Seite aus R3	
dazu jedesmal 20 Seiten aus R’ lesen	
jedes Ergebnis in einer Seite von R’’ ablegen	300 x (20 + 1) +300
	6650 Seitenersetzungen

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
③	456	gold	SF	621
	555	gold	NY	c17
④	789	silver	SF	NULL
⑤	891	silver	LA	d07
	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 CacheGroup

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.

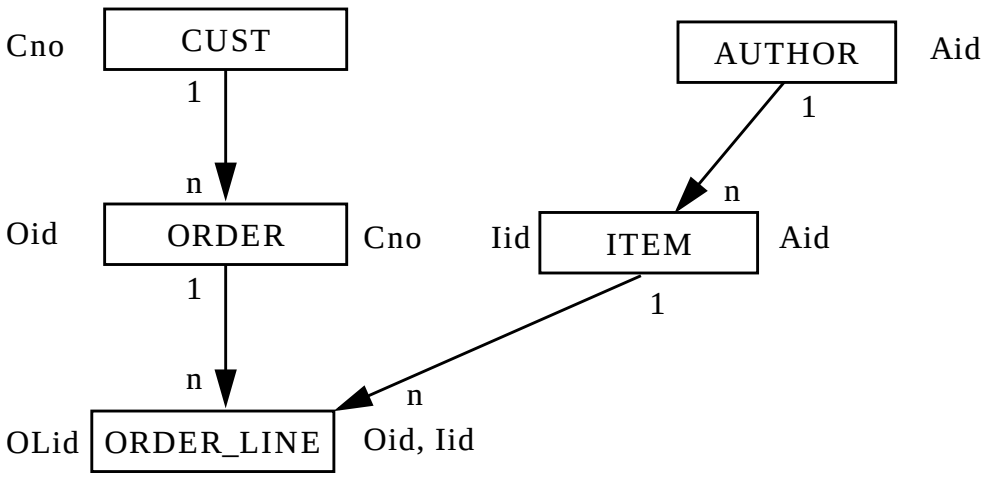
Tabelle ORDER_LINE (OL) habe u.a. die U-Spalte OLid und die NU-Spalten Oid und Iid.

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.

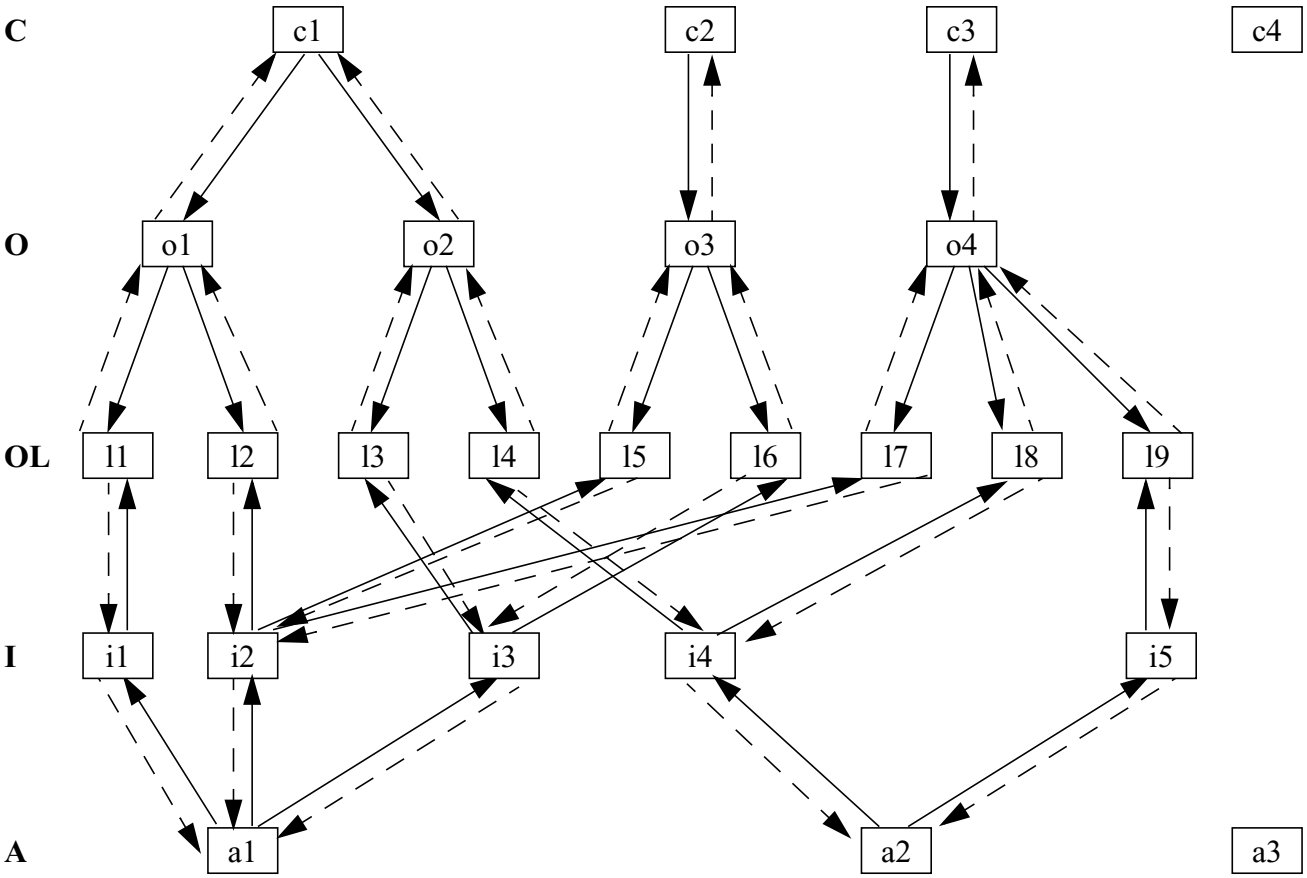
Im Backend-DB-Server (BE) seien für diesen Schemaausschnitt die referentiellen Integritätsbedingungen C.Cno → O.Cno und O.Oid → OL.Oid (als Primär-/Fremdschlüsselbeziehungen) definiert.

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.

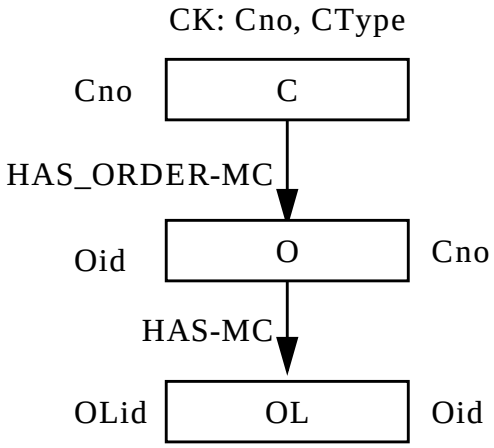
a1 habe den Wert ‘mayr’ in A.AName, während a2 und a3 ‘codd’ in A.AName haben.

Im Frontend-DB-Server (FE) sei die Wurzel-Tabelle FE-C einer CacheGroup 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.

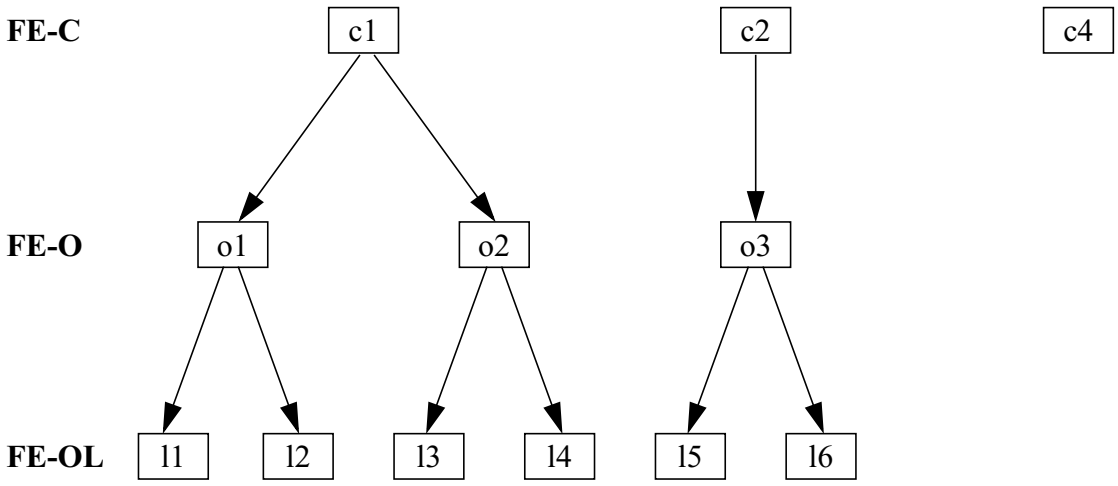
Im folgenden gehen wir immer von einem leeren Cache in der FE-DB aus.

a) Die CacheGroup 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?



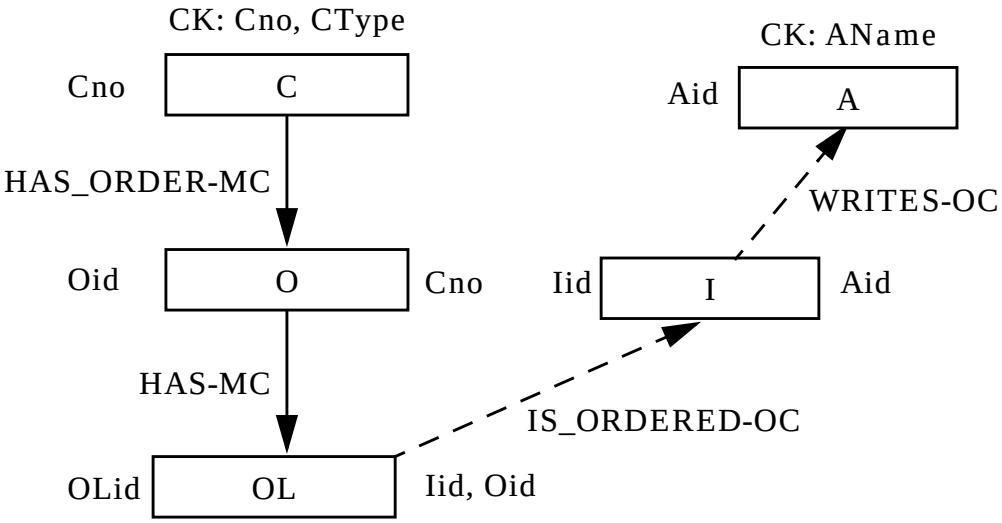
Geben Sie Beispiele für Verbundprädikate, die in der FE-DB ausgewertet werden können.

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

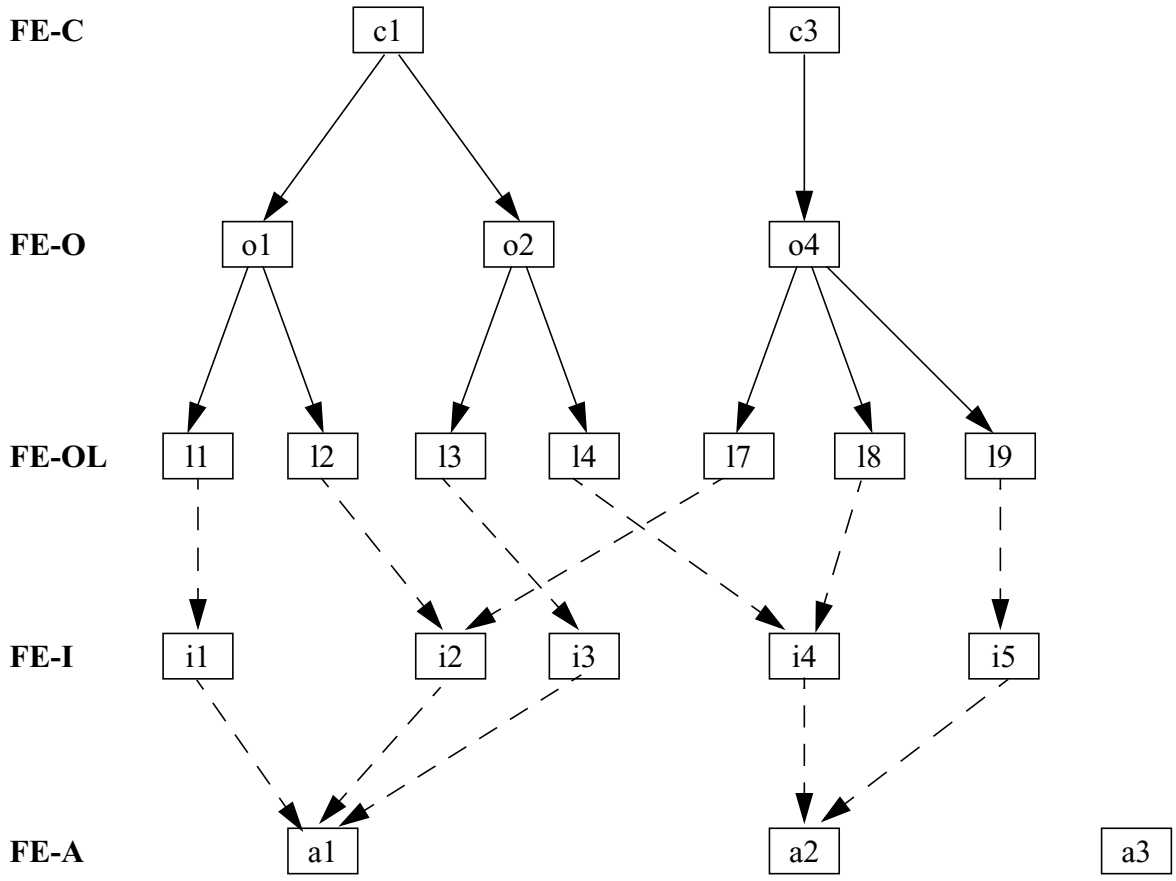
- C.CType = 'silver' AND C.Cno = O.Cno

- ...

b) CacheGroup G2 sei wie folgt definiert:



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

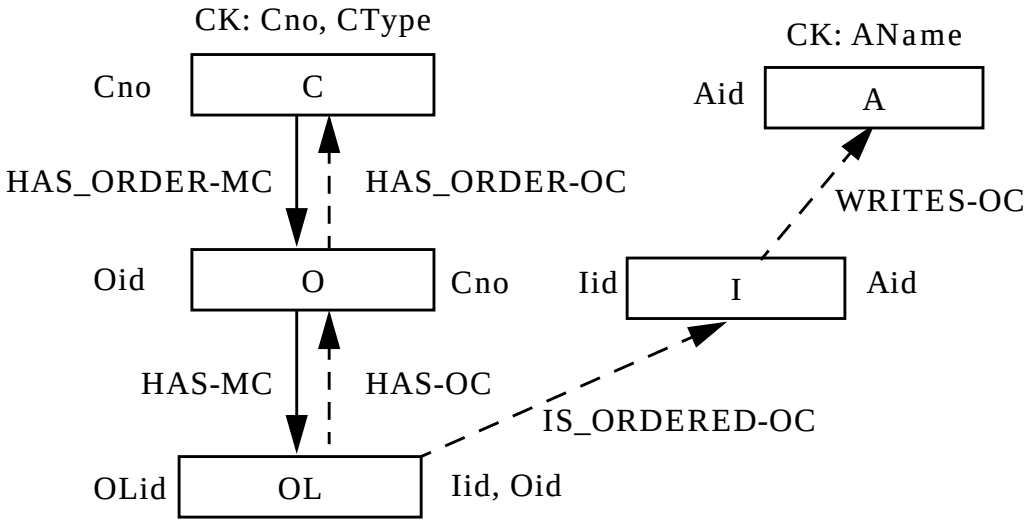


In dieser Situation referenziert Q4 (A.AName = ‘codd’).

Was ändert sich an der Belegung von G2?

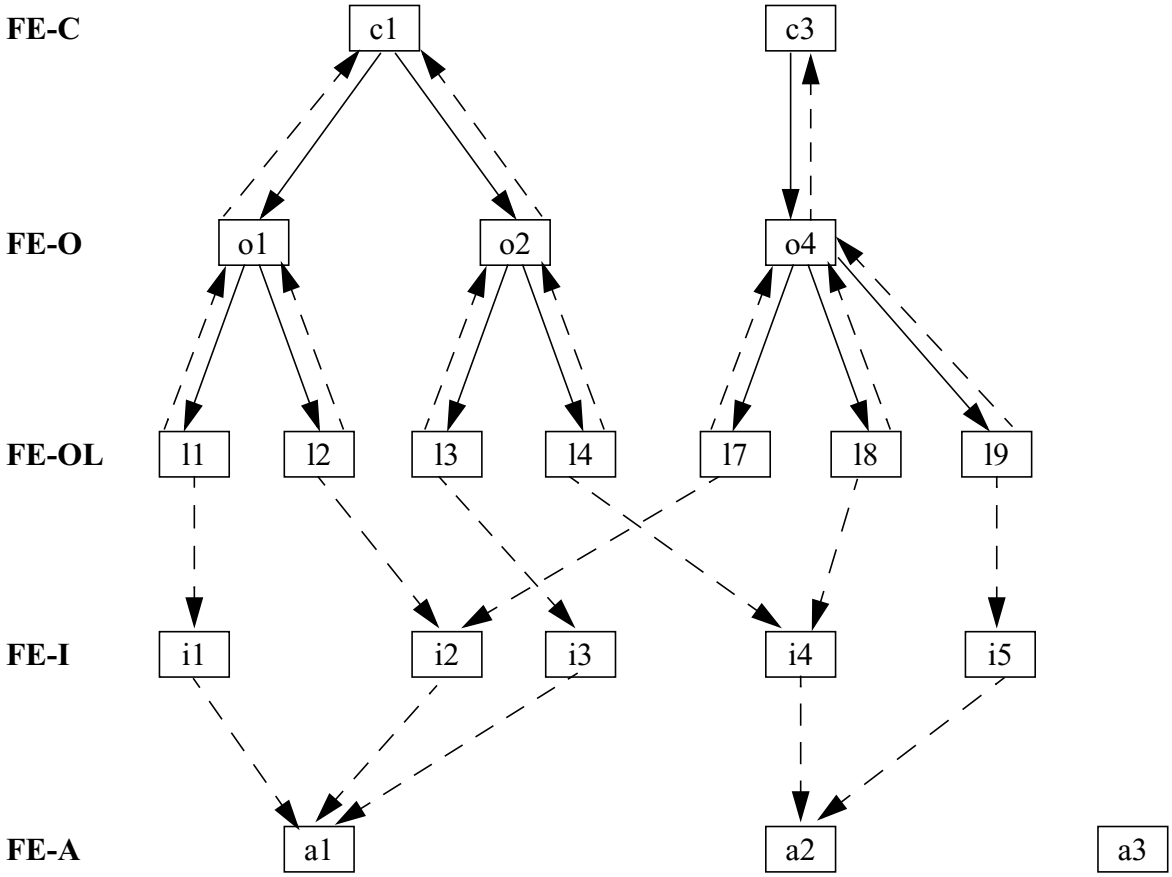
Nichts!
Q4 kann in Cache beantwortet werden.

c) CacheGroup G3 sei wie folgt definiert:



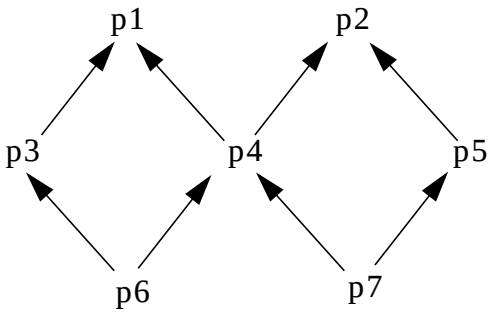
Wie sieht die Belegung von G3 nach Ausführung von Q3 und Q4 aus?

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



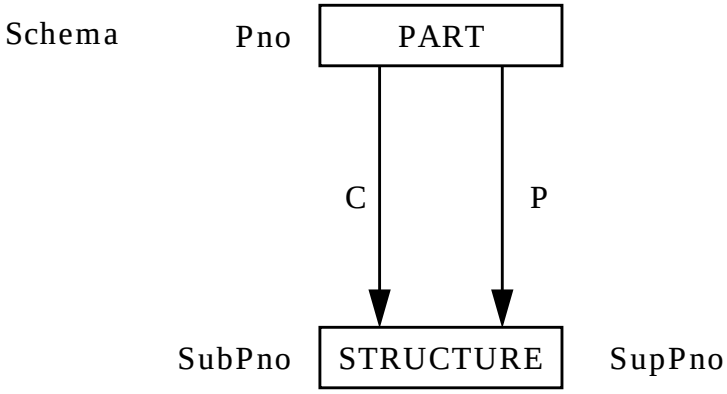
Aufgabe 4: Rekursives Laden von CacheGroups

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:



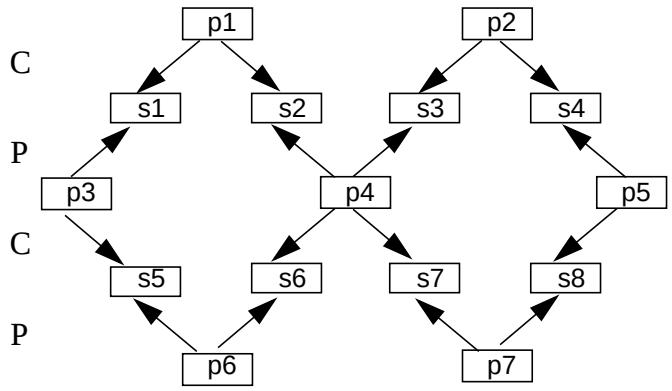
C = COMPOSED_OF

P = PART_OF

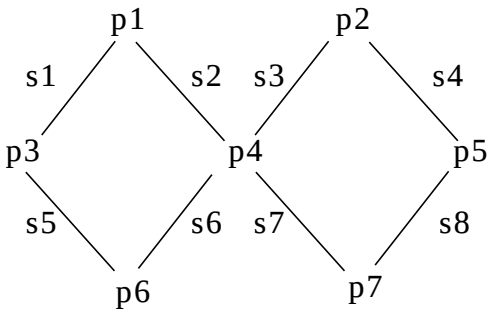
Gemäß dieses Schemas sei die obige Stückliste GG1 in der Backend-DB (BE-DB) in den Tabellen BE-PART und BE-STRUCTURE gespeichert.

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



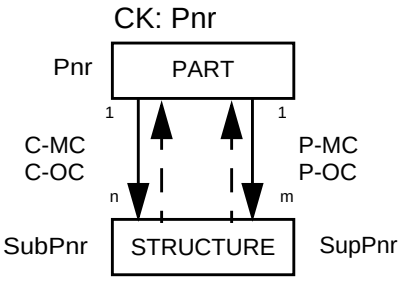
Kantenbezeichnungen



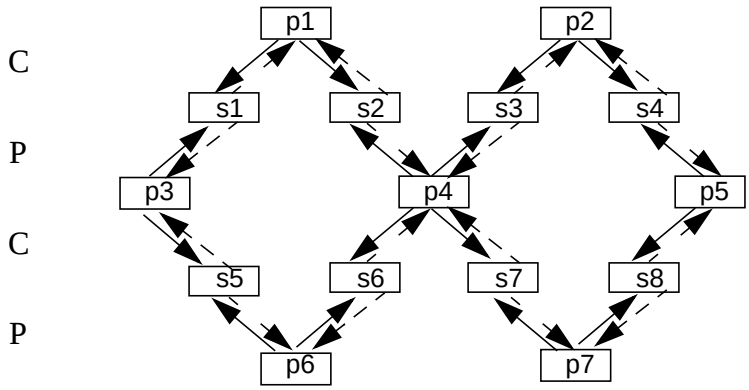
Im Cache sei der Cache Key Pno für FE-PART definiert.

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

a) Gegeben sei folgende Definition der CacheGroup G1:



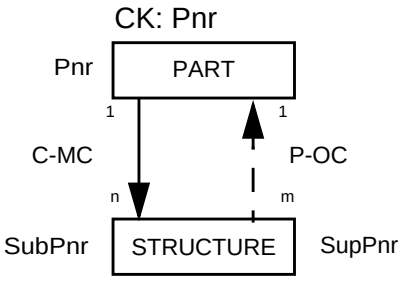
Wie sieht die Belegung von G1 aus, wenn in Anfrage Q1 'Pno = 1' (p1) referenziert wurde?



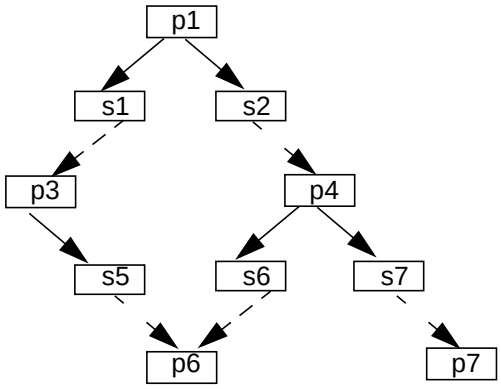
Welche Belegung wäre aus einer Referenz von p4 ('Pno = 4') oder p6 ('Pno = 6') resuliert?

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

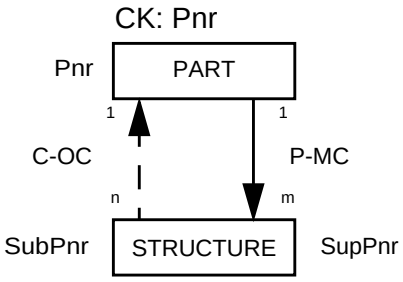
b) Gegeben sei die nachfolgende Definition von G2:



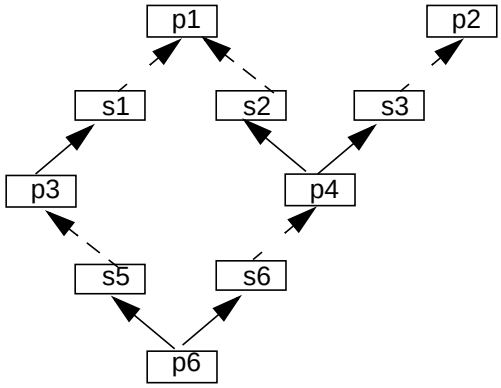
Wie sieht die Cache-Belegung aus, wenn ‘Pno = 1’ (p1) referenziert wurde?



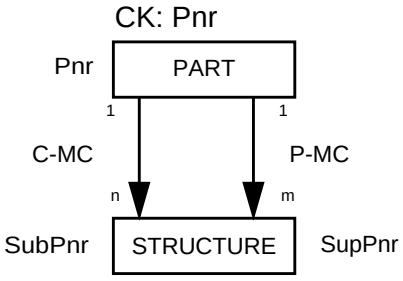
c) Gegeben sei die nachfolgende Definition von G3:



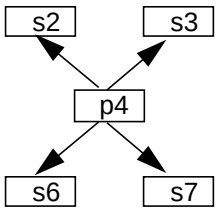
Wie sieht die Cache-Belegung aus, wenn ‘Pno = 6’ (p6) referenziert wurde?



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 CacheGroup G rekursives Laden vermieden wird?

Es dürfen in G keine heterogenen RCC-Zyklen, in denen verschiedene Spalten in **einer** Tabelle referenziert werden, auftreten.
In CacheGroups 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.

Aufgabe 5: Direkte und Indirekte Satzadressierung 12

a) Vergleichen Sie den Speicherplatzbedarf für folgende Adressierungsprinzipien:

1. logische Byte-Adresse
2. TID-Konzept
3. DBK-Konzept
4. DBK/PPP-Konzept.

Die Ausprägungen eines zu adressierenden Satztyps seien jeweils auf ein Segment der Größe 2^{32} Byte beschränkt. Es können mehrere Satztypen in einem Segment gespeichert sein. Welche Faktoren werden durch die einzelnen Adressierungskonzepte bei gegebener Pointerlänge begrenzt:

N_{REC} : Anzahl der Sätze,
S_l : logische Satzlänge,
L_s : Seitengröße,
m_k : Anzahl der Seiten (im Segment)?

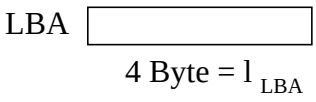
Wie hoch ist der gesamte Speicherplatzbedarf für die Adressierung eines Satzes, wenn n verschiedene Pointer auf den Satz zeigen? Die Anzahl der Überläufer bei TID-Konzept liege bei 10%.

Mögliche Faktoren, die begrenzen:

- N_{rec} : Anzahl der Sätze

- S_l : logische Satzlänge $S_l = 128 = 2^7$
- L_s : Seitenlänge $L_s = 2^{12} = 4$ KByte
- m : Anzahl Seiten [im Segment]

a) **1) logische Byteadresse hat Länge l_{LBA}**



Vor jedem Satz steht ein Satzkennzeichen der Länge $l_T = l_{SKZ}$ Bytes (4 Byte)

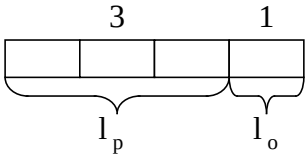
$m = \frac{2^{lLBA}}{L_S} = \frac{2^{32}}{2^{12}} = 2^{20}$ Seiten [im Segment]

$N_{rec} = \frac{2^{lLBA}}{S_l} = \frac{2^{32}}{2^7} = 2^{25}$ Datensätze

Aufwand für n Adressierungen: $n \cdot l_{LBA} + l_{SKZ} = (4n + 4)$ Bytes

n = 10 : 44 Bytes

2) TID



Pointer auf Seite Offset in Seite

$m = 2^{lp}$ Seiten [im Segment]

$N_s = 2^{lo}$ Datensätze / Seite; s wird aber auch durch Seitengröße und Satzlänge begrenzt

$$N_s = \frac{L_s}{S_l}$$

$$\Rightarrow 2^{l0} \geq \frac{L_s}{S_l}$$

$N_{rec} = 2^{lp} \cdot N_s$ Datensätze

Aufwand : $s = n (l_p + l_o) + l_{slot} + l_{SKZ}$

$P_{overflow}$: 10 % $+ P_{ov} \cdot n \cdot (l_p + l_o + l_{slot})$

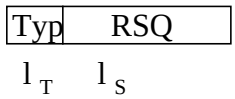
$= (4n + 6 + 0.1 \cdot 6)$ Byte

L_{slot} : 2 Byte $n = 10 \rightarrow 46{,}6$ Byte

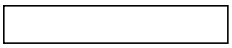
→ Behälter

→ Seiteninterner Verweis

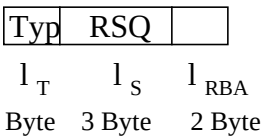
3) DB-Key / PP

POINTER 

Typkennzeichen ein Byte
im DBK und Pageheader

DBTT
PP 

l_{pp} = 3 Byte = 24 Bit

Seite 

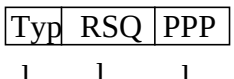
$m = 2^{l_{pp}} = 2^{24}$ Seiten

$N_{rec} = 2^{l_s} = 2^{24}$ Datenseiten je Typ bei 2^8 Typen

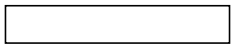
Aufwand: $s = n (l_t + l_s) + l_{pp} + l_T + l_s + l_{RBA}$
$= 4n + 3 + 4 + 2$ Byte
$= 4n + 9$ Byte

für $n = 10 \rightarrow$ 49 Byte

4) DB-Key / PP / PPP

POINTER 

1 3 3 Byte

DBTT
PP 

l_{pp} = 3 Byte

Seite 

$m = 2^{l_{pp}} = 2^{24}$ Seiten

$N_{rec} = 2^{l_s} = 2^{24}$ Datensätze

Aufwand: $s = n (l_t + l_s + l_{pp}) + l_{pp} + l_T + l_s + l_{RBA}$
$= 7n + 9$ Byte

für $n = 10 \rightarrow$ 79 Byte

b) Schätzen sie den Zugriffsaufwand bei den verschiedenen Adressierungskonzepten ab. Wieviele physische E/A-Zugriffe sind bei einer Folge wahlfreier Zugriffe über einen Indexbaum mit 3 Stufen im Mittel zu erwarten, bis der gesuchte Datensatz gefunden ist,

1. beim TID-Konzept
2. beim DBK-Konzept
3. beim DBK/PPP-Konzept?

Der DB-Puffer besitze 40 Seitenrahmen zu 4 K Byte; die Ersetzungsstrategie sei LRU. Es seien $N_{REC}=10^5$ Datensätze über 10^4 Seiten verteilt. Der Indexbaum habe neben der Wurzel 9 Seiten auf der 2. Indexstufe und 1000 Seiten auf der 3. Indexstufe. Die Zuordnungstabelle bestehe aus Seiten der Größe 4 KB und besitze PP-Einträge der Länge 4 Byte. Die Anzahl der Überläufer sowie die Anzahl der falschen PPPs sei 10%.

Vergleichen sie die Anzahl der Zugriffe, wenn über die interne logische Adresse (TID, DB-Key) direkt zugegriffen wird.

b) Annahme :Wurzel + 2. Indexstufe bleiben durch LRU im DB-Puffer, bei anderen Seitentypen verhält sich LRU wie FIFO. 10 Pufferrahmen durch Wurzel und 2. Indexstufe ständig belegt. Im Schnitt 15 Indexseiten der 3. Stufe und 15 Datenseiten im Puffer.

Begründung:

Die Zugriffe erfolgen in der Reihenfolge 1., 2., 3. Indexseite, Datenseite ⇒ Jeder 4. Zugriff referenziert die Wurzel, die dadurch nicht verdrängt wird. Nach 9 Zugriffen sind 1 + 9 + 9 + 9 = 28 Rahmen belegt. Beim 10. Zugriff wird wieder die Wurzel referenziert. Die Seite der 2. Indexstufe befindet sich auch schon im Puffer. Bei gleichmäßiger Verteilung wird also auch die 2. Stufe des Index-Baumes immer im Puffer gefunden.

Wahlfreier Zugriff → Anzahl der Zugriffe, wenn Seite nicht im Puffer steht;

$\frac{\text{Anz. Rahmen}}{\text{Anz. Seiten}}$ gibt an, wie wahrscheinlich es ist, daß eine Seite im Puffer zu finden ist.

Wie häufig findet man die Seite nicht: $1 - \frac{\text{Anz. Rahmen}}{\text{Anz. Seiten}}$

1) TID

1.1 Index

$$N_{TID} = \underbrace{\left(1 - \frac{15}{1000}\right)}_{\text{Index}} + \underbrace{\left(1 - \frac{15}{10000}\right)}_{\text{Daten}} + \underbrace{0{,}1 \cdot \left(1 - \frac{15}{10000}\right)}_{\text{Überlauf}}$$

$$= 2{,}08335$$

1.2 **Direktzugriff über TID ohne Index** (40 Datenseiten):

$$N_{direkt} = 1{,}1 \cdot \left(1 - \frac{40}{10000}\right) = 1{,}0956$$

2) DB-Key / PP-Konzept

DBTT = 4096 / 4 = 1028 Einträge / Seite $\Rightarrow N_{REC}/1028 \approx 100$ DBTT-Seiten

2.1 Index

$$N_{PP} = \underbrace{\left(1 - \frac{10}{100}\right)}_{\text{DBTT}} + \underbrace{\left(1 - \frac{10}{1000}\right)}_{\text{Index}} + \underbrace{\left(1 - \frac{10}{(10000)}\right)}_{\text{Daten}}$$

10 Seiten : DBTT
10 Seiten : Index
10 Seiten : Daten

$$= 2{,}889$$

2.2 Direktzugriff ohne Indexstruktur

$$N_{direkt} = \underbrace{\left(1 - \frac{20}{100}\right)}_{\text{DBTT}} + \underbrace{\left(1 - \frac{20}{10000}\right)}_{\text{Daten}} = 1{,}798$$

20 DBTT-Seiten
20 Daten-Seiten

3) DB-Key / PP / PPP-Konzept

3.1 mit Index (2 DBTT-Seiten (nur in 10% benutzt!), 14 Index-Seiten, 14 Datenseiten)

$$N_{PPP} = \underbrace{\left(1 - \frac{14}{1000}\right)}_{\text{Index}} + \underbrace{\left(1 - \frac{14}{10000}\right)}_{\text{Daten}} + 0{,}1 \cdot \underbrace{\left(\underbrace{1 - \frac{2}{100}}_{\text{DBTT}} + \underbrace{1 - \frac{14}{10000}}_{\text{Daten}} \right)}_{\text{Fehler}}$$

$$= 2{,}182$$

3.2 ohne Index (4 DBTT-Seiten, 36 Datenseiten)

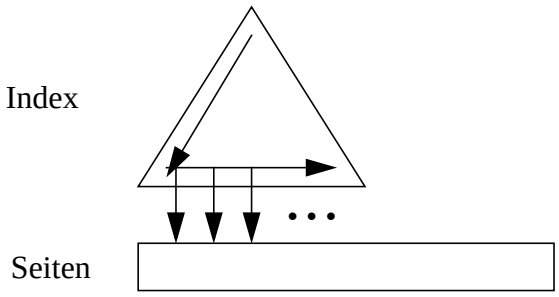
$$N_{direkt} = \left(1 - \frac{36}{10000}\right) + 0{,}1 \cdot \left(1 - \frac{4}{100} + 1 - \frac{36}{10000}\right) = 1{,}192$$

c) Ermitteln Sie die Anzahl der physischen Zugriffe bei der fortlaufenden Verarbeitung aller Datensätze über eine Indexstruktur, wenn mit

1. dem TID-Konzept
2. dem DBK-Konzept
3. dem DBK/PPP-Konzept

adressiert wird. Welche Ergebnisse erhalten Sie unter den Annahmen von b) im *best case* (Clusterbildung) und im *worst case*?

Die Verarbeitung erfolgt derart, daß die Wurzel und die Seite ganz links auf der 2. Indexstufe gelesen werden. Danach werden sequentiell alle Seiten der 3. Indexstufe gelesen und jeweils die zugehörigen Datenseiten.



1) **TID**: Wurzel 2. Index 3. Index Datenseite

best case: 1 + 1 + 1000 + $10000 + a_1 \times N_{REC} \times \left(1 - \frac{38}{10000}\right)$

= **20964**

worst case: 1 + 1 + 1000 + $N_{REC} \cdot \left(1 - \frac{39}{10000}\right) + \alpha_1 \times N_{REC} \times \left(1 - \frac{38}{10000}\right)$

= **100602**

2) **DB-Key / PP**

best case Clusterbildung DBTT + Datenseiten

: Wurzel + 2. Index + 3. Index + DBTT sequ. + Daten sequ.

1 + 1 + 1000 + 100 + 10 000

= **11102**

best case Clusterbildung Datenseiten

: Wurzel + 2. Index + 3. Index + DBTT + Datenseiten sequ.

1 + 1 + 1000 + N_{REC}(1-38/100)+ 10000

= **73002**

worst case : Wurzel + 2. Index + 3. Index + DBTT + Daten

$$1 + 1 + 1000 + N_{REC} \cdot \left(1 - \frac{19}{100}\right) + N_{REC} \cdot \left(1 - \frac{19}{10000}\right)$$

= **180 810**

3) **DB-Key / PP/PPP**

best case (Clusterung der Daten):
(DBTT irrelevant, da im Normalfall kein Zugriff)

Wurzel	+ 2. Index	+3. Index	+ Daten
1	+1	+ 1000	+10000

$$+ \text{Fehler} = \left(p_{ov} \cdot N_{REC} \cdot \left(1 - \frac{19}{100} + 1 - \frac{19}{10000}\right)\right)$$

$$+ 0{,}1 \cdot N_{REC} \cdot \left(1 - \frac{19}{100} + 1 - \frac{19}{10000}\right)$$

$$= 11002 + 10000 \cdot (0{,}81 + 0{,}9981)$$

$$= 11002 + 10000 \cdot 1{,}8081$$

= **29083**

worst case (Clustering Daten):

Wurzel	+ 2. Index	+3. Index	+
1	+1	+ 1000	+

Daten + (Daten + DBTT)

$$N_{REC} \cdot \left(1 - \frac{35}{10000}\right) + 0{,}1 \cdot N_{REC} \cdot \left(1 - \frac{4}{100} + 1 - \frac{35}{10000}\right)$$

$$= 11002 + 99650 + 9600 + 9965$$

= **120217**

Zusammenfassender Vergleich:

Adressierung:

	Formel	n =10
LBA	4n + 4	44
TID	4 n + 6,6	46,6
DBK	4n + 9	49
PPP	7n + 9	79

Seitenzugriffe

seq. Zugriff	Datencluster	Daten + DBTTcluster	Keine Clusterung
TID	20964		100612
DBK	73002	11102	180810
PPP	29083	29083	120217
wahlfreie Zugriffe	Index	direkt	
TID	2,08335	1,0956	
DBK	2,889	1,798	
PPP	2,182	1,192	