4. Datenbank-Caching

Motivation

Unterschied: DB-Pufferverwaltung und DB-Caching

Klassifikation von DB-Caching-Verfahren

DBProxy: Einsatz von materialisierten Sichten

DBCache: Einsatz von Cache Groups

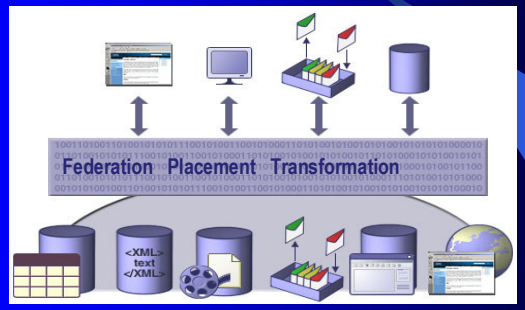
Das neue Zauberwort

DB-Caching – Vision

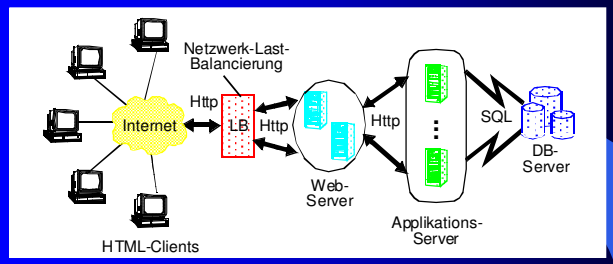
The three most important parts of any Internet application are caching, caching, and, of course, caching ... Larry Ellison

Informationsintegration

- Globales Szenario (N. Mattos)

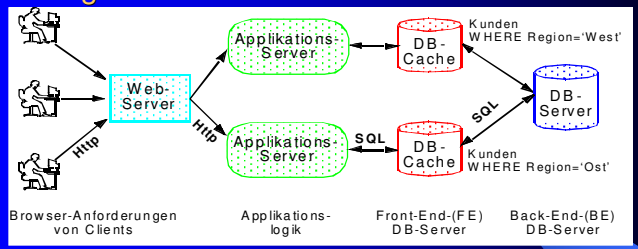


Antwortzeit bei DB-basierten Web-Anwendungen



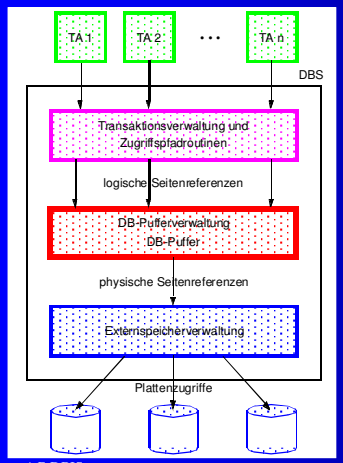
- Caching von Web-Seiten
  - bei statischen Inhalten in Proxies entlang des Aufrufweges
  - aber:
    - immer mehr dynamisch generierte Inhalte, personalisierte Web-Seiten
    - zielgerichtete Werbung, (1:1)-Marketing, interaktives E-Commerce

Caching von DB-Daten



- Caching ist bewährte Technik zur Verbesserung von
  - Skalierbarkeit und Leistungsverhalten großer, verteilter Systeme
  - Antwortzeit und Verfügbarkeit für den Benutzer
- DB-Caching „in der Nähe“ des Applikations-Servers (ähnlich wie bei SAP-Lösung, nur allgemeiner)
- Neue Herausforderungen
  - deskriptive DB-(Teil)-Anfragen im Cache
  - Durchführung von Änderungstransaktionen?

DB-Pufferverwaltung – Was ist zu entscheiden?

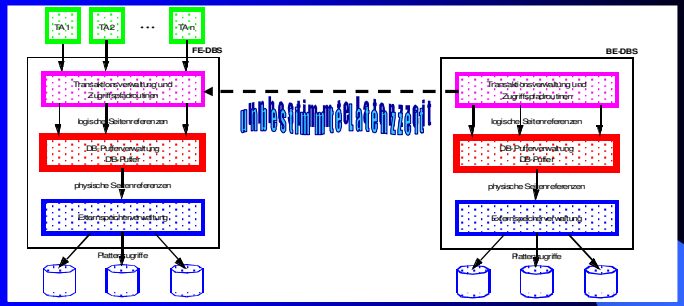


- Auch bei der DB-Pufferkapazität gilt das Moore´sche Gesetz seit 1970!
- Verarbeitungsanforderung
  - Was?
    - Granulat: Seite
  - Wann?
    - durch logische Seitenreferenzen
      - bedarfsgetrieben (on demand)
      - vorab (prefetching)
- Pufferinhalt bestimmt durch
  - Lokalität aller Seitenreferenzen
  - Clock, LRU-K, LRD, …
- DB-Aktualisierungen
  - durch Seitenoperationen
  - bei Commit der ändernden Transaktion sichtbar
- Datenkonsistenz
  - jüngster Zustand der Seiten
  - keine Kopien und damit keine Replikation

DB-Pufferverwaltung – Vollständigkeitsbedingung

- Alle Datensätze, die ein Anfrageprädikat erfüllen,
  - sind **vollständig** in der DB gespeichert
  - müssen **garantiert** durch Zugriffspfade erreicht werden
- Anfrageoptimierung übernimmt die Abbildung
  - auf eine Folge von Seitenreferenzen
  - DB-Pufferschnittstelle: Fix <page>, Unfix <page>
  - **Zugriffsplan:** 4711, 0815, 1234, …
- Jede Seitenreferenz allokiert Seite im DB-Puffer
  - Seite erfüllt Vollständigkeitsbedingung für Fix-Aufruf
  - Zugriffsplan holt **Prädikatsextension** der Anfrage in den DB-Puffer und erfüllt damit ihre **Vollständigkeitsbedingung**

DB-Caching – entscheidende Unterschiede!

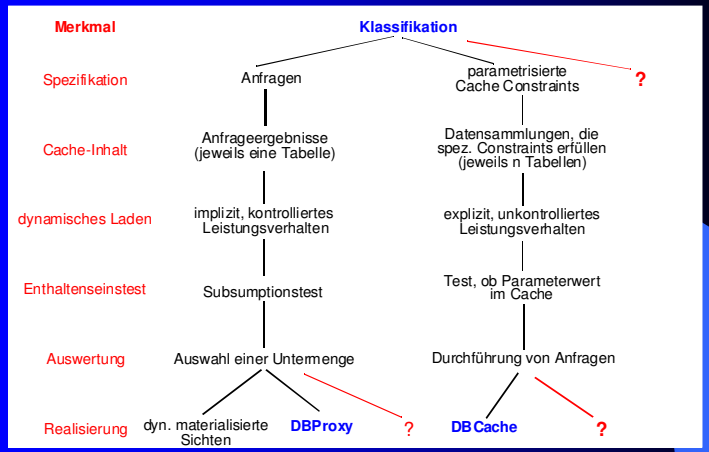


- Anforderung an das Konzept
  - datenmodellabhängig
  - dynamisch
  - **adaptiv**
- Leistungsanforderung
  - DB-Caching in AW-Nähe
  - versucht u. a., **ONC über Weitverkehrsnetze** zu vermeiden
- Inhalt der FE-DB muss Auswertung von **(Teil-)Anfragen** erlauben
- DB-Aktualisierungen (zunächst) im BE-DBS
- Datenkonsistenz
  - Referenz auf oft veraltete DB-Zustände
  - Anfrage kann mehrere DB-Zustände sehen

DB-Caching – Was ist zu entscheiden?

- Wozu soll der Cache genutzt werden?
- Was soll im Cache gehalten werden?
- Wie wird der Cache-Inhalt spezifiziert?
- Wann werden die Daten in den Cache geladen?
- Sind überlappende Datenmengen im Cache zugelassen?
- Wann werden Daten im Cache aktualisiert?
- Wann werden Daten im Cache ersetzt bzw. invalidiert?

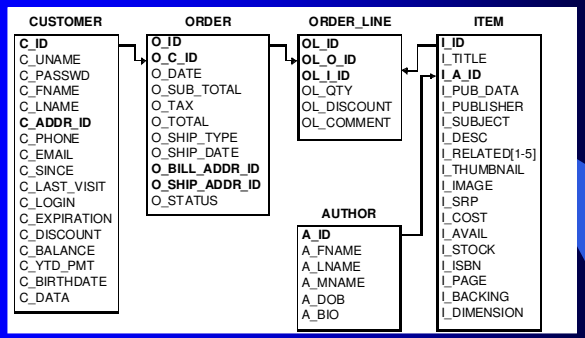
Klassifikation von DB-Caching-Verfahren



DBProxy – Forschungsprojekt (IBM Yorktown)

- Was? – Spezifikation des Cache-Inhaltes
  - dynamische Entscheidung, welche Sichten sich vorzuhalten lohnen
  - durch Liste von Anfragen, die im Cache-Index beschrieben sind
- Wann und wo? – Speicherung im Cache
  - bei erstmaliger Anfrage, wenn sie gewünschte Sicht betrifft
  - gemeinsame Speicherung von Sichten
    - Vermeidung von Replikation
    - „common-schema storage-table policy“
  - trotzdem entstehen i. Allg. überlappende Tabellen (Replikate im Cache)!
- Wie? – Anfragen im Cache
  - beziehen sich auf einzelne FE-Tabellen
  - wenn Subsumptionstest erfolgreich!

DBProxy – Vereinfachtes Schema eines Web-Buchhändlers



- Tabelle BE-item
  - hat 18 Spalten mit i_id als Primärschlüssel
  - es werden Anfrageergebnisse hinsichtlich Kosten und Verkaufspreis (srp: suggested retail price) in Tabelle FE-item im Cache gehalten

DBProxy – Verfahrensaspekte

- Anfragen
  - werden so umgeschrieben, dass sie i_id enthalten;
  - Vermeiden von Duplikaten in einer FE-Tabelle
- Einfügen
  - Es ist zu prüfen, ob FE-Tabelle mit Spalten zu erweitern ist
  - Optimierung: Definition einer „umfassenden“ Tabelle mit Vorabwissen
- Speicherung verschiedener Anfragen
  - in einer FE-Tabelle erzeugt unbelegte Spaltenwerte („fake“ NULL values)
  - spätere Cache-Auswertung darf sie nicht benutzen!

DBProxy – Beispiel

FE-item

i_id	i_cost	i_srp	i_isbn
5	14	8	NULL
120	15	22	NULL
340	16	13	abc
450	NULL	18	cde
620	NULL	20	efg

Retrieved by Q_1
SELECT i_cost, i_srp FROM item
WHERE i_cost BETWEEN 14 AND 16

Retrieved by Q_2
SELECT i_srp, i_isbn FROM item
WHERE i_srp BETWEEN 13 AND 20

DBProxy – Beispiel mit umfassender Tabelle

FE-item

i_id	i_cost	i_srp	i_isbn
5	14	8	ghi
120	15	22	ijk
340	16	13	abc
450	25	18	cde
620	30	20	efg

Retrieved by Q_1
SELECT i_cost, i_srp, i_isbn FROM item
WHERE i_cost BETWEEN 14 AND 16

Retrieved by Q_2
SELECT i_cost, i_srp, i_isbn FROM item
WHERE i_srp BETWEEN 13 AND 20

DBProxy – Subsumptionstest bei Anfragen

- Enthaltenssseinstypen von Anfragen
  - vollständig enthalten: Sicht von Cache-Prädikat Q_i
  - enthalten in Q_i und Q_j
  - nur teilweise enthalten in einer oder mehreren im Cache gehaltenen Sichten
- Enthaltensein von Q_B (Matching-Alg.)
  - Ergebnis von Q_B ist enthalten in dem von Q_A, wenn das WHERE-Prädikat von Q_B das von Q_A für alle möglichen Werte der Tabelle, z. B. von item, logisch impliziert
    +
    Menge der benötigten Spalten ist in der von Q_A oder in der umfassenden (aber nicht dynamisch erweiterten) Tabelle enthalten
  - Q_B.wherep → Q_A.wherep äquivalent zur Aussage

„Q_B.wherep AND (NOT (Q_A.wherep)) ist nicht erfüllbar“

(i_cost < 5 AND NOT (i_cost > 25))

→ Folglich ist Q_B nicht in Q_A enthalten!

DBProxy – Aktualisierung

FE-item

i_id	i_cost	i_srp	i_isbn	
5	14	8	NULL	Q_1
120	15	22	NULL	
340	16	13	abc	
450	NULL	18	cde	Q_2
620	NULL	20	efg	
770	15	30	NULL	
880	15	40	NULL	

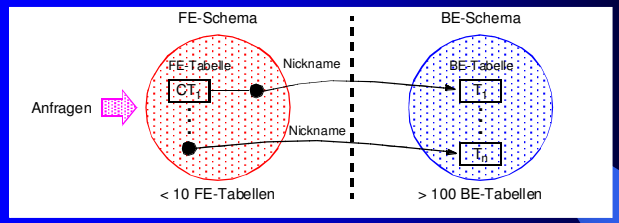
eingefügt durch Konsistenzprotokoll

- DB-Daten sollten sich nur langsam verändern
  - es wird δ-Konsistenz gewährleistet
  - Änderungen im DB-Server verlangen das Propagieren von betroffenen Sätzen möglicherweise in mehrere FE-DBs
- Ersetzung oder Invalidierung
  - nur selektives Entfernen von Sätzen
    - bei Q_2 nur i_id = {450, 620}
    - Wert abc muss auf NULL gesetzt werden
  - i. Allg. sehr komplex, da sich überlappende Satzmengen (Q_i) zu entfernen sind

DBCache – Forschungsprojekt (IBM ARC)

- Was? – Spezifikation des Cache-Inhaltes
  - Festlegung der BE-Tabellen, für die FE-Tabellen gehalten werden
  - Spezifikation von **Cache Constraints** (CCs)
    - **Cache Keys**
    - **Referential Cache Constraints** (RCCs)
- Wann und wo? – Speicherung im Cache
  - Sind gewünschte Daten nicht im Cache, wird die Anfrage im BE ausgewertet. Die entsprechenden Daten werden in den Cache geladen
  - Datensammlungen werden als **Cache Groups** replikationsfrei gespeichert
- Wie? – PSJ-Anfragen im Cache
  - Cache Groups enthalten **Prädikatsextensionen** (parametrisierte CCs)
  - erlauben einfache **Gleichheitsprädikate** und **Gleichverbunde**

DBCache (2)



- Transparenter Einsatz von DBCache
  - Jede BE-Tabelle ist im FE-Schema als CT_i oder als Nickname vorhanden
    - gleiche Namen für Schemata und Elemente
    - gleiche Anzahl und Typen der Spalten
  - Für jede CT_i können so andere relevante BE-Objekte im Cache gehalten werden
    - logische Objekte: Sichten, Funktionen, Constraints, gespeicherte Prozeduren
    - physische Objekte: Indexe

Cache Groups – Definitionen

- Bereichsvollständigkeit einer Spalte
  - Wenn ein Spaltenwert im Cache gefunden wird, garantiert der Cache-Mgr, dass alle Sätze mit diesem Wert sich im Cache befinden. UNIQUE-Spalten sind somit immer bereichsvollständig.
  - Folglich wird die Auswertung eines **Gleichheitsprädikates** <ColName> = <value> durch eine solche Spalte unterstützt!
- Cache Key
  - kann für eine FE-Tabelle spezifiziert werden
  - bezieht sich auf eine Spalte und dient als „Füllpunkt“
  - besitzt die Eigenschaft **„bereichsvollständig“** (domain complete, DC)
  - Mechanismen zur Einschränkung seiner Wertemenge sind lebenswichtig (~ Stoppwort- oder Empfehlungsliste)

Cache Groups – Beispiel zu Cache Keys

- Tabelle CUST habe zwei UNIQUE-Spalten (U) Cnr und Cid und zwei Spalten CType und CLocation vom Typ NON UNIQUE (NU)
- Für CUST seien im Cache CType und Cnr als Cache Keys definiert
- Anfrage mit ‘CType=gold‘ führt bei leerem FE-CUST zu folgender Belegung:
- Anfrage mit ‘Cid=a14‘ wird im Cache ausgeführt. Diese Anfrage hätte jedoch nicht zum Laden des Cache geführt
- Cache-Belegung nach Anfrage mit ‘Cnr=789‘

BE_CUST	Cnr	CType	CLocation	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	
	...	...	...		

FE_CUST	Cnr	CType	CLocation	Cid	...
FE_CUST	Cnr	CType	CLocation	Cid	...
	123	gold	SJ	a14	
	456	gold	SF	b21	
	789	silver	NY	NULL	
	891	silver	LA	d07	

Cache-Key-Regel zum kontrollierten Laden

- Für eine FE-Tabelle dürfen n Cache Keys deklariert werden
- CType (NU) und CLocation (NU) seien Cache Keys
- Höchstens ein Cache Key darf die Eigenschaft NU besitzen.

  Warum muss diese Einschränkung eingeführt werden?

BE_C	Cno	CType	CLoc	Cid
	1	silver	SF	NULL
	2	silver	LA	a
	3	plat.	SJ	b
	4	unspec.	LA	c
	5	gold	SJ	d
	6	gold	SF	e
	7	gold	NY	f
	8	bronze	Chi	g
	9	NULL	Chi	h

FE_C	Cno	CType	CLoc	Cid
	3	plat.	SJ	b
	5	gold	SJ	d
	6	gold	SF	e
	7	gold	NY	f
	1	silver	SF	NULL
	2	silver	LA	a
	4	unspec.	LA	c

Cache Groups – Korrektheit

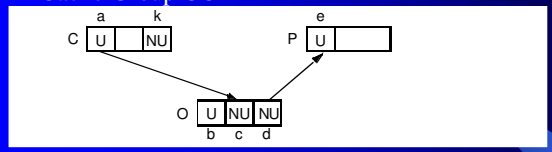
- Wie wird der Zusammenhang zwischen FE-Tabellen spezifiziert?
  - Achtung: Einführung eines wertbasierten Tabellenmodells
  - RCCs beziehen sich auf Paare von Spalten (in der Regel) verschiedener Tabellen und sind vom Typ
    - U → U (1:1)
    - U → NU (1:n, z. B. PS/FS-Beziehung oder Member-Constraint)
    - NU → U (n:1, z. B. FS/PS-Beziehung oder Owner-Constraint)
    - NU → NU (n:m)
- Zentrale Eigenschaft
  - Im Cache sind nur Sätze gespeichert, die spezifizierte CCs erfüllen
  - Wird ein Satz im Cache gefunden, der ein bestimmtes CC erfüllt, sind alle dieses CC erfüllenden Sätze im Cache

Cache Groups – Korrektheit (2)

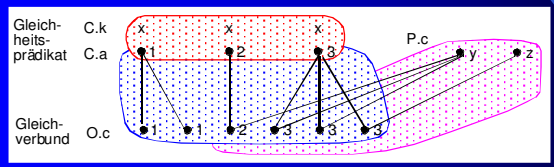
- „Konstruktion“ von Prädikatsextensionen durch RCCs
  - Wurzel-Tabelle einer Cache Group besitzt Cache Key(s) und ist mit anderen FE-Tabellen durch RCCs verknüpft
  - Ein RCC garantiert, dass alle Sätze, die für entspr. **Gleichverbund** benötigt werden, im Cache sind
  - So lassen sich Cache Groups für Verbunde im Cache bilden
- Zusammenhang zwischen CK und DC
  - CK-Spalte impliziert DC-Eigenschaft, spezifiziert **Füllpunkt**
  - Nicht jede DC-Spalte ist CK-Spalte!
  - Jede DC-Spalte stellt potenziellen **Einstiegspunkt** dar
    - Probing ermittelt dynamisch, ob DC für einen speziellen Wert gegeben ist
    - Test eines **Gleichheitsprädikats**

Konstruktion von Prädikatsextensionen

- Cache Group COP



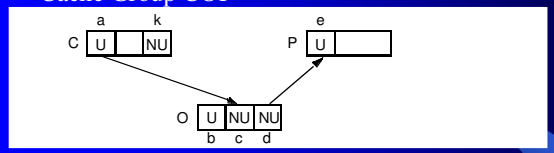
- Referenz auf C.k = x



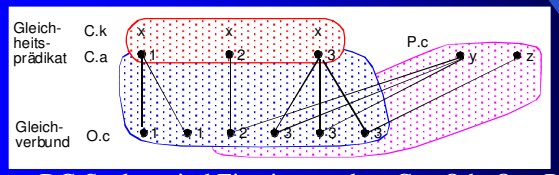
  - Where C.k = x
  - and C.a = O.c
  - and O.d = P.e

Zusatzgewinn durch Einstiegspunkte

- Cache Group COP



- Cache-Belegung

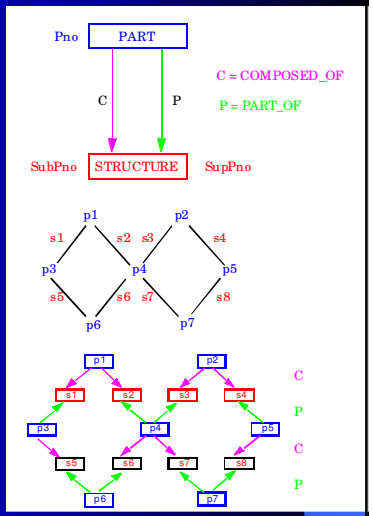


- DC-Spalten sind Einstiegspunkte: C.a, O.b, O.c, P.e
  - Where C.c = 3 and O.d = P.e
  - Where P.e = z

…

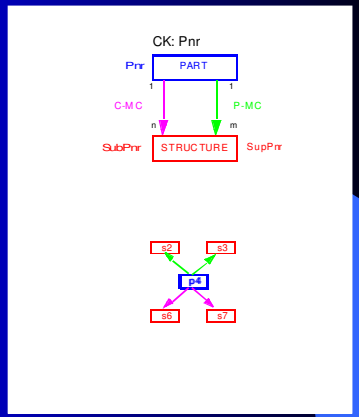
Cache Groups – Beispiel

- Schema einer Stückliste
  - Pfeile stellen PS/FS-Beziehungen dar und
  - sind referentielle Integritäts-bedingungen
- Gozinto-Graph GG1 mit Kantenbezeichnungen
- GG1 als Graph
  - mit PS/FS-Beziehungen für C und P
  - entspricht der BE-Belegung



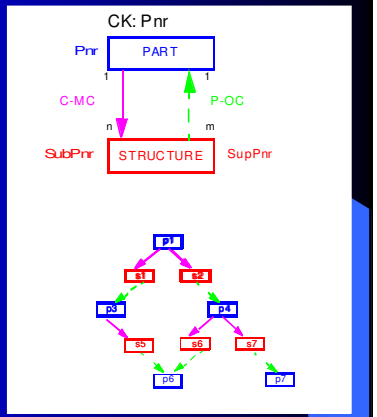
Cache Groups – Beispiel (2)

- Definition der Cache Group G1
  - MC: Member Constraint
  - OC: Owner Constraint entsprechen (U → NU)- und (NU → U)-Beziehung
- FE-Belegung
  - anfänglich leer
  - in Q1 werde ‘Pnr=4‘ (p4) referenziert



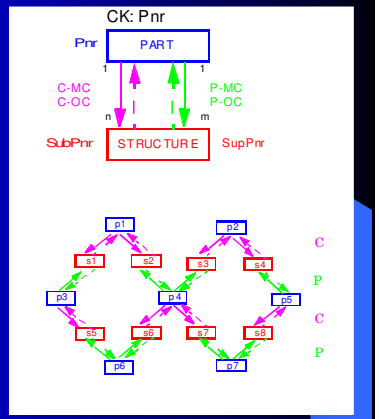
Cache Groups – Beispiel (3)

- Definition der Cache Group G2
- FE-Belegung
  - anfänglich leer
  - in Q2 werde ‘Pnr=1‘ (p1) referenziert



Cache Groups – Beispiel (4)

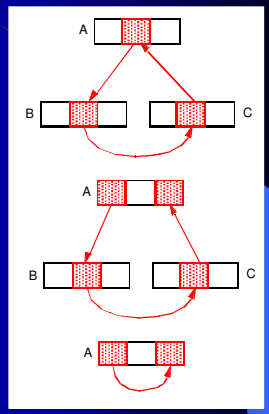
- Definition der Cache Group G4



- FE-Belegung
  - anfänglich leer
  - in Q4 werde 'Pnr=1' (p1) referenziert

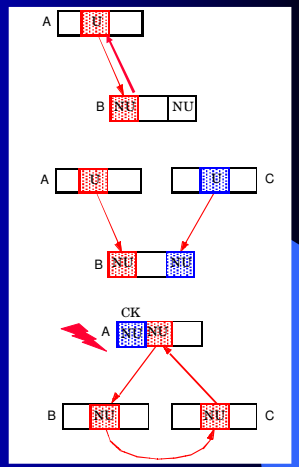
Cache Groups – kontrolliertes Laden (safeness)

- Homogener RCC-Zyklus
  - Wenn die BE-Tabellen für die Zyklus-Spalten C_A, C_B und C_C jeweils dieselbe Wertemenge besitzen, sind diese Spalten DC
  - Laden setzt sich i. Allg. rekursiv nicht fort
- Heterogene RCC-Zyklen
  - NU- oder U-Spalten im Zyklus können rekursives Laden hervorrufen
  - auf einer Tabelle möglich

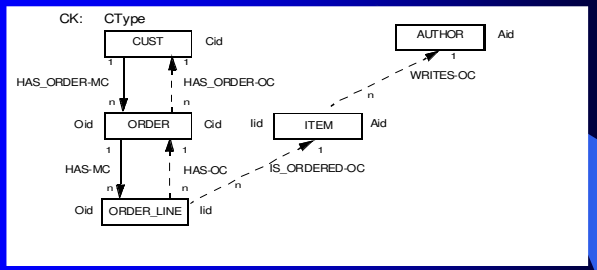


Cache Groups – kontrolliertes Laden (2)

- Explizites DC
  - UNIQUE-Spalten
  - Cache-Keys
- Implizites DC
  - Es gilt AB-MC und AB-OC, wenn B nicht über andere Pfade gefüllt wird
  - Kein DC – B wird auch über weitere Pfade gefüllt
- Rekursionsfreies Laden von G
  - **keine heterogenen RCC-Zyklen** in G
  - Jede Tabelle ist höchstens in einem homogenen NU-Zyklus involviert
  - kein Cache Key (NU) auf einer Tabelle neben den Spalten eines homogenen NU-Zyklus
  - Beachtung der **Cache-Key-Regel**

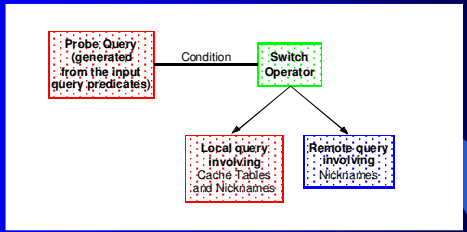


Cache Groups - Anwendungsbeispiel



- Im BE werde die DB eines Web-Buchhändlers verwaltet
- Einstieg über Test eines Gleichheitsprädikats (Probing)
- Bei Erfolg garantiert jedes RCC, dass der zugehörige Verbund (Equi-Join) im Cache ausgeführt werden kann

DBCache – Janus-Pläne

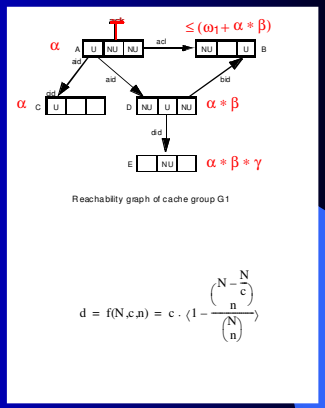


- Anfrageplan zunächst als „entfernter Plan“ nur mit Nicknames
- Generieren einer Probe Query zur Überprüfung aller möglichen Gleichheitsprädikate
- „Clonen“ des Anfragegraphs, wobei möglichst viele CT_i benutzt werden, Local Query kann in DB2 als föderierte Anfrage ausgeführt werden
- Remote Query wird z. B. benutzt, wenn höchste Konsistenz verlangt wird

Cache Groups – Füllverhalten

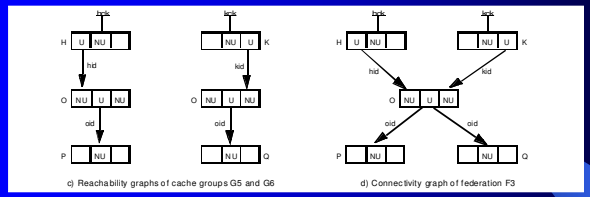
- Referenzieren eines Cache Key
  - α Sätze haben jeweils einen Wert von ack
  - RCC mit U→ NU füge β bzw. γ Sätze pro Owner ein
  - Welche Wirkung hat RCC mit NU → NU ?
- Wie viele verschiedene Werte d in einer NU-Spalte sind zu erwarten, wenn ein Wert einer DC-Spalte in den Cache kommt?
  - n = α
  - c = Card (acl)
  - N = N_A

$\omega_1 = d^* N_B/c$



$$d = f(N,c,n) = c \cdot \left\langle 1 - \frac{\binom{N-\frac{N}{c}}{n}}{\binom{N}{n}} \right\rangle$$

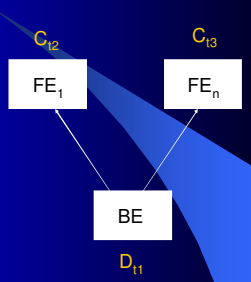
Cache Group Federations – Ent- und Belastung



- Kosten für
  - G5: $n_{G5} = \alpha_5 * (1 + \beta_5 * (1 + \gamma_5))$ = 1110
  - G6: $n_{G6} = \alpha_6 * (1 + \beta_6 * (1 + \gamma_6))$ = 1110
- Belastung für F3
  - $\Delta n_{OIF3} = - \alpha_5 * \beta_5 * \alpha_6 * \beta_6 / N_O$ = -1
  - $\Delta n_{PIF3} = \alpha_6 * \beta_6 * \gamma_5$ = 1000
  - $\Delta n_{QIF3} = \alpha_5 * \beta_5 * \gamma 6$ = 1000

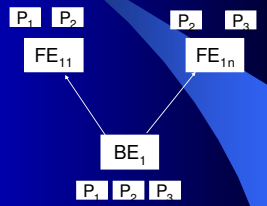
Aktualisierung – Konsistenzgarantien

- δ-Konsistenz zum Ursprung
  - $C_t = D_{t-\delta}$
  - $D_{t1} \geq C_{t2}$ und $D_{t1} \geq C_{t3}$
- Monotone Zustandsübergänge
  - Cache ändert sich mit der Zeit nur „vorwärts“
  - Wenn eine Cache-AW D_{t1} und später D_{t2} sieht, dann ist $t_2 \geq t_1$
- Unmittelbare Sichtbarkeit von Änderungen
  - Wenn eine AW eine Änderung mit Commit abschließt, müssen ihre Auswirkungen und die aller früheren Änderungen bei einer nachfolgenden Anfrage sichtbar sein



Aktualisierung der DB

- Verschiedene Modi (update propagation modes)
  - DAIA: deferred apply – immediate apply
    - alle Änderungen im BE
    - asynchrones Propagieren zu den FEs
  - IADA: immediate apply – deferred apply
    - alle Änderungen im FE innerhalb einer TA; dazu müssen zu ändernde Sätze im FE sein
    - asynchrones Propagieren zum BE und dann zu den anderen FEs
  - IAIA: immediate apply – immediate apply
    - synchrone Änderungen in FE und BE
    - innerhalb derselben TA



Aktualisierung einer FE-Tabelle (BE verzögert)

- einfachster Fall
  - isolierte BE-Tabelle
  - alle Änderungen in der zugehörigen FE-Tabelle
  - Primärschlüssel: Cno
    Cache Keys: Cid (U), CType (NU)
- Insert
  - Einfügen der neuen Sätze
  - ggf. Nachladen von Sätzen bei NU-DC-Bedingung
  - Wie wird UNIQUE im FE getestet?
  - Trigger-Aktionen müssen auf FE-Tabelle beschränkt sein
  - NULL-Semantik bei DC?
- Delete
  - im Cache: Cno = 8
  - Daten im BE: Cno = 3, Cno = 6
    - Holen
    - Löschen

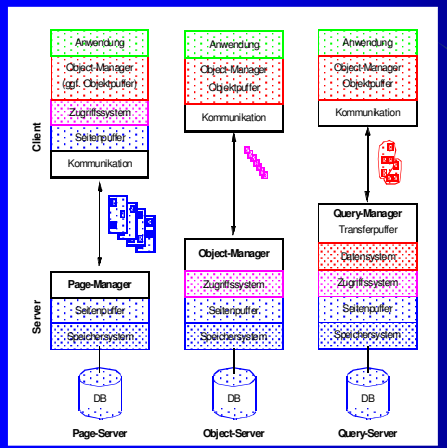
BE_C	Cno	CType	CLoc	Cid
	1	silver	SF	NULL
	2	silver	LA	a
	3	plat.	SJ	b
	4	NULL	LA	c
	5	NULL	SJ	d
	6	gold	SF	e
	7	gold	NY	f

FE_C	Cno	CType	CLoc	Cid
	8	silver	NY	c
	1	silver	SF	NULL
	2	silver	LA	a
	5	NULL	SJ	d
	3	plat.	SJ	b
	6	gold	SF	e

Gibt es noch andere Ideen?

- Das sind alles Implementierungen!
- Was sind die Konzepte dahinter?
- Gibt es neue, bisher unbekannte Ideen?

Der Schlüssel zur Lösung



Lösung ist datenmodellabhängig!

Objekte müssen entsprechend der gewählten Abstraktionsebene bereitgestellt werden

Einfach bei Page-Server

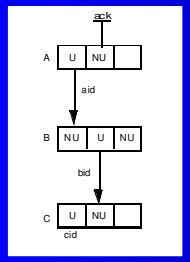
- Fix, Unfix <page>

und Object-Server

- read, write object <oid>
- get object <simple search arg>

aber bei Query-Server?

Gleichheitsprädikate – Nutzung der DC-Eigenschaft



A.ack = <value1> And
A.aid = B.aid And B.bid = C.bid
oder
A.ack = <value1> And A.aid = B.aid
oder
A.ack = <value1>

Aber auch für jede DC-Spalte in A:

A.aid = <value2> And
A.aid = B.aid And B.bid = C.bid
oder
A.aid = <value2> And A.aid = B.aid
oder
A.aid = <value2>

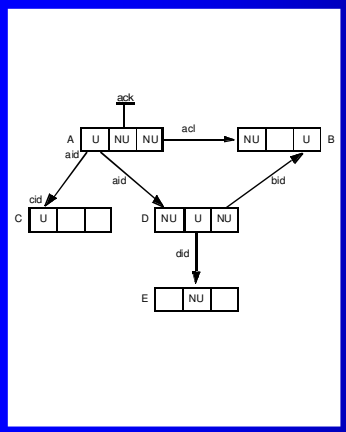
und auch für jede DC-Spalte in B:

B.bid = <value3> And B.bid = C.bid
oder
B.bid = <value3>

und auch für jede DC-Spalte in C:

C.cid = <value4>

Wie sieht das auf komplexen CGs aus?



- CG gewährleistet Auswertung von parametrisierten Prädikaten und enthaltenen Teilprädikaten
  - (A.ack = <value> And A.aid = C.cid)
  - (A.ack = <value> And A.aid = D.aid And D.did = E.did)
  - (A.ack = <value> And A.aid = D.aid And D.bid = B.bid)
  - (A.ack = <value> And A.acl = B.acl)
- Sind noch komplexere Prädikate auf der Wurzeltabelle wünschenswert?

Die neuen Zauberworte

Value completeness (VC). *A value v is said to be value complete in a column S.c if and only if all records of $\sigma_{c=v}S_B$ are in S.*

Domain completeness (DC). *A column S.c is said to be domain complete if and only if all values v in S.c are value complete.*

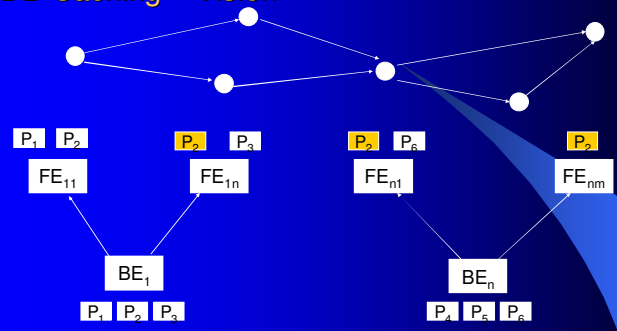
Value range completeness (RC). *A value range r = (l <= x <= u) is called range complete in a column S.c if and only if all records of $\sigma_{c>=l \text{ and } c<=u}S_B$ are in S (making all individual values in S.c value complete).*

Predicate completeness (PC). *A collection of tables is said to be predicate complete with respect to predicate P if it contains all records needed to evaluate P, that is, its predicate extension.*

Zauberwort: Prädikatsvollständigkeit (PC)

- Caching-Verfahren bisher mit PC für
  - Selektion (DBProxy, full-table caching, materialisierte Sichten)
  - Gleichheitsprädikate und Verbunde (DBCache)
- Andere Klassen von Verfahren für SQL-PC bei
  - Bereichsprädikate und Verbunde:
    z. B. A.ack Between <val5> And <val6>
  - Aggregationsfunktionen
  - Rekursion
  - …
- Offensichtlich lässt sich PC auch für andersartige deskriptive Datenmodelle anwenden
  - XQuery-Operationen
  - …

DB-Caching – Vision



- Datenhaltung als
  - VDBS
  - FDBS
  - Grid- oder P2P-Architektur
    (peer-based data management is a natural extension of data integration, A. Halevy)

Zusammenfassung

- Anwendungsnahes Caching wird immer wichtiger für
  - Antwortzeiten
  - Skalierbarkeit
- DBProxy-Ansatz
  - explizites Laden des Cache zusammen mit Anfrage, gut kontrollierbar
  - Subsumptionstest – begrenzte Anfragemächtigkeit
- DBCache-Ansatz
  - implizites Laden des Cache zur Erfüllung der Cache Constraints
  - schwierig zu kontrollieren, deshalb feinere Abstimmung erforderlich
  - flexible Abwicklung von Verbunden
- Sofortige Aktualisierung im Cache
  - allgemeine Lösungen?
  - Kosteneffektivität?
- PC angewendet auf Typen von Prädikaten bestimmt Lösungsraum!