Trends in DBS

6. Web and Database Caching

Theo Härder
www.haerder.de

Issues of Web caching

What is DB caching?

Research prototypes: DBProxy and DBCache

ACCache: Adaptive constraint-based DB caching

Implementation issues for a middleware-based cache system

Update models for DB caching

Main references:

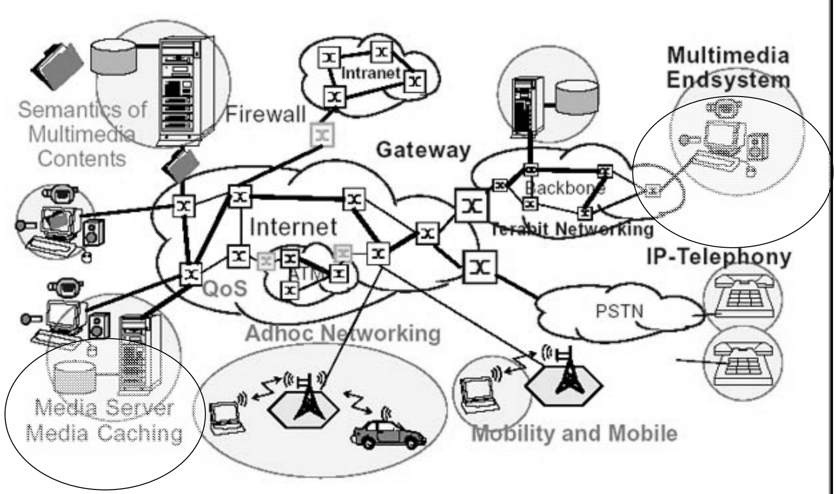
Theo Härder, Andreas Bühmann: Value Complete, Column Complete, Predicate Complete—Magic Words Driving the Design of Cache Groups, appears in VLDB Journal, 2006.
Theo Härder: Caching over the Entire User-to-Data Path in the Internet, in: Data Management in a Connected World, LNCS 3551, Springer, June 2005, pp. 67–89.

Trends in DBS

The Big Picture

Web caching
Approaches to DB caching
DBProxy
DBCache
Adaptive and constraint-based
Cost considerations
Implementation of ACCache
Update models for caches

- Main issue: support of query processing closer to the end user



The three most important parts of any Internet application are caching, caching, and, of course, caching … *Larry Ellison, Oracle CEO (2001)*

DBIS Datenbanken und Informationssysteme

Outline

- The client-to-server path and the user-to-data path through the Internet
  - Web caching is identifier-based
  - edge-side includes supporting caching of dynamic fragments
- What is database caching?
- DBProxy – materialized views
- DBCache – constraint-based DB caching
  - domain completeness
  - cache keys and referential cache constraints
  - correctness and safeness
- ACCache – enhanced cache adaptivity
  - value, column, and predicate completeness
  - probing alternatives
  - candidate values – selective cache loading and unloading
- Cost considerations for cache groups and federations
- Implementation of ACCache
- Update models for FE and BE databases

Trends in DBS

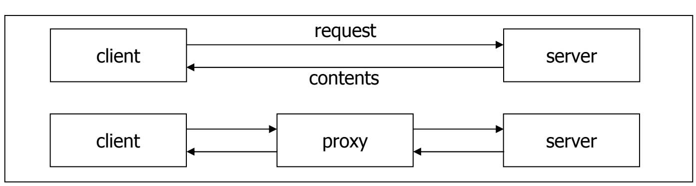
Web caching
Approaches to DB caching
DBProxy
DBCache
Adaptive and constraint-based
Cost considerations
Implementation of ACCache
Update models for caches

DBIS Datenbanken und Informationssysteme

© 2005 AG DBIS

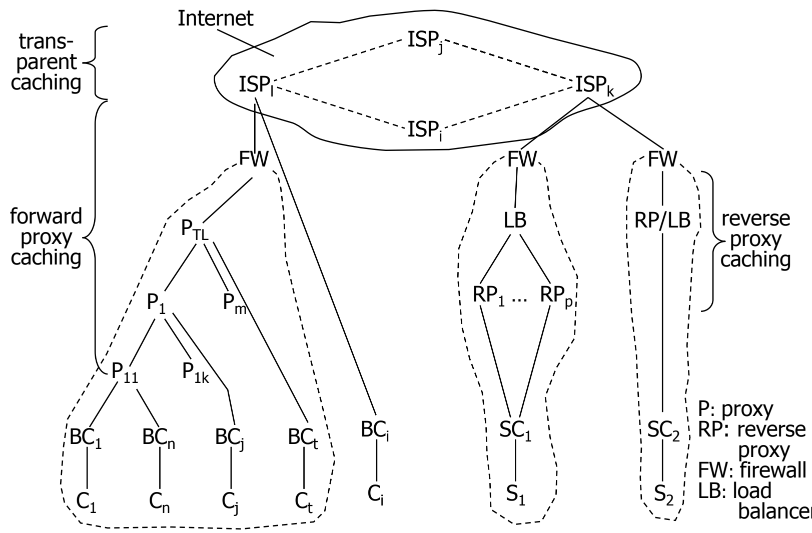
The Client-to-Server Path Through the Internet

- Conceptually, a Web request is processed as follows: A Web client sends a query containing a URL via HTTP and the Internet to a Web server (server, for short) **identified by the URL**. The server processes the request, generates the answer (typically an HTML or XML document), and sends it back to the client. To solve the performance and availability problems, we add, again conceptually, a Web proxy server somewhere in the client-to-server path. Such a proxy can be used in a number of ways, including
  - caching documents and parts thereof
  - converting data to HTML/XML format so it is readable by a client browser
  - providing Internet access for companies using private networks
  - selectively controlling access to the Internet based on the submitted URL
  - permitting and restricting client access to the Internet based on the client IP address



The Client-to-Server Path Through the Internet (2)

Web caching
Approaches to DB caching
DBProxy
DBCache
Adaptive and constraint-based
Cost considerations
Implementation of ACCache
Update models for caches



The Client-to-Server Path Through the Internet (3)

Web caching
Approaches to DB caching
DBProxy
DBCache
Adaptive and constraint-based
Cost considerations
Implementation of ACCache
Update models for caches

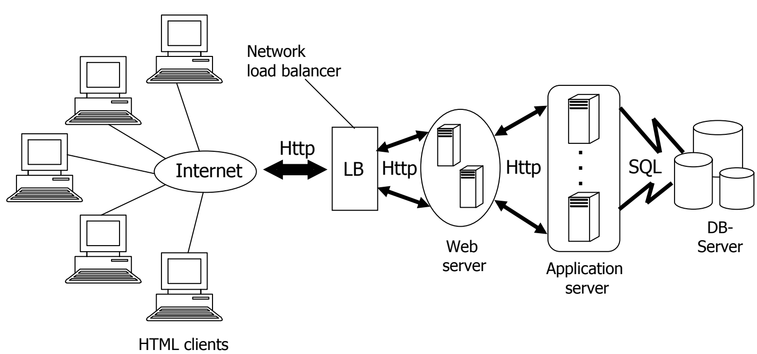
- *Browser cache*: For all user requests, this cache dedicated to the browser is first searched. If the specific content is located in the cache, it is checked to make sure that it is „fresh". Such a private cache is particularly useful if a user scrolls back in his request history or clicks a link to a page previously looked at.
- *Proxy cache*: While working on the same principle, but a much larger scale, such a cache is shared, performs demand-driven *pull caching*, and serves hundreds or thousands of users in the same way. It can be set up on the firewall or as a stand-alone device. Unless other search paths are specified, a cache miss sends the request to the next proxy cache in the client-to-server path.
- *Reverse proxy cache*: This kind of cache is an intermediary also known as "edge cache", "surrogate cache", or "gateway cache". While not demand-driven, such caches reverse their role as compared to proxy caches, because they are supplied by origin servers with their most recent offerings—a kind of *push caching*. Furthermore, they are not deployed by network administrators to save bandwidth and to reduce user-perceived delays which is characteristic for proxy caches, but they are typically deployed by Web masters themselves, to unburden the origin servers and to make their Web sites more scalable, reliable, and better performing.
- *Server cache*: It keeps generated content and enables reuse without interaction of the origin server. Intermediate results and deliverable Web documents help to reduce the server load and improve server scalability.

Replacement Strategies for Web Caches

- As compared to DB buffer management, Web caching is much more complex. Web objects need variable-length frames and are characterized by more factors that critically influence the replacement decision:
  - time of the last reference to the object (recency)
  - number of requests to an object while in the cache (frequency)
  - size of the Web object (size)
  - cost to fetch an object from its origin server (cost)
  - time of last modification, time when an objects gets stale (expiration time).

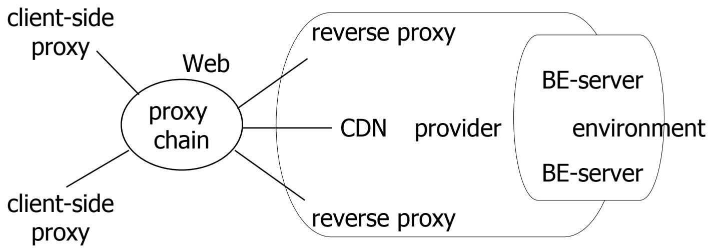
- The influence or interdependencies of these factors cannot be discussed in detail. We only summarize the resulting cache replacement strategies which typically exploit the first four factors above:
  - *Recency-based strategies* incorporate recency (and size and/or cost) into the replacement process.
  - *Frequency-based strategies* exploit frequency information (and size and/or cost) in the replacement decision.
  - *Recency/frequency-based strategies* consider both recency and frequency under fixed or variable cost/size assumptions.

The User-to-Data Path Through the Internet



- Caching of Web pages
  - static content in proxies in the client-to-server path
  - but:
    - more and more dynamically generated content, personalized documents
    - targeted marketing, 1-to-1 marketing,
    - interactive pages of online shops (e-commerce)
    - member logins of community pages, etc.

Improvement of Scalability



- Use of edge servers
  - trustworthy servers
  - execution of Web requests
    - reduces response time
    - avoids BE overload
- Relocation of application components which usually run in BE
- Provision of dynamic content

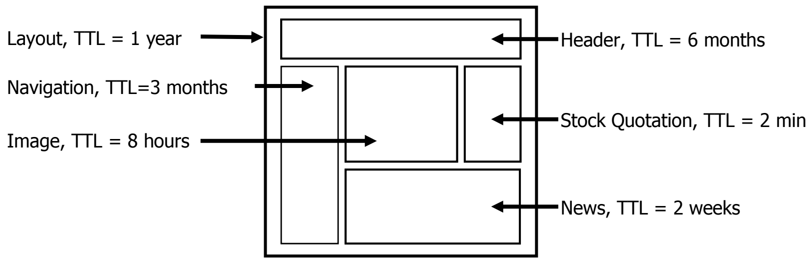
Validity of Cached Objects

- The ***Expires*** **HTTP header** is the basic means of controlling caches. It tells the cache how long the object is fresh for; after that time, the cache will always check back with the origin server to see if a document is changed.
- If no *Expires* value as the definite time limit is set (for ZeroTTL objects), the cache may estimate the freshness via ***Last-Accessed*** or ***Last-Modified***.
- **Cache-Control response headers** contain directives to specify what is cacheable, what may be stored, how to modify the expiration mechanism, and how to revalidate or reload objects. Useful *Cache-Control* response headers include:
  - *max-age=[seconds]* – specifies the maximum amount of time that an object will be considered fresh. Similar to Expires, this directive allows more flexibility.
  - *public* – marks the response as cacheable, even if it would normally be uncacheable, e.g., if the object is authenticated, the public directive makes it cacheable.
  - *no-cache* – forces caches (both proxy and browser) every time to submit the request to the origin server for validation before releasing a cached copy. This is useful to assure that authentication is respected (in combination with public), or to maintain rigid object freshness, without sacrificing all of the benefits of caching.
  - *must-revalidate* – tells the cache that it must obey any freshness information given about an object. By specifying this header, the cache is forced to strictly follow the given rules.

Fragmentation of Dynamic Web Documents

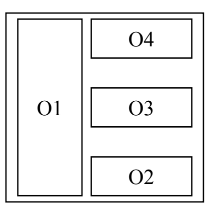
- When a few and small content fragments exhibit high update frequencies, the concept of **edge-side includes (ESI)** is particularly helpful. Dynamic Web pages are not handled as units anymore, instead fragment-enabled caching allows the management of such pages at a finer level of granularity.

Edge side includes (ESI) defines the de-facto standard



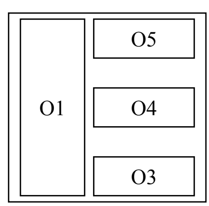
Personalization and Movement of Fragments

a) Template1 (t=t1, p=p1)



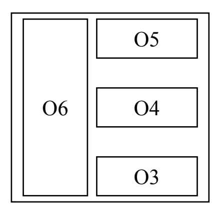

```
template {
<esi:include src=o1>
<esi:include src=o4/>
<esi:include src=o3/>
<esi:include src=o2/>
}
```

b) Template1 (t=t2, p=p1)




```
template {
<esi:include src=o1>
<esi:include src=o5/>
<esi:include src=o4/>
<esi:include src=o3/>
}
```

c) Template1 (t=t2, p=p2)



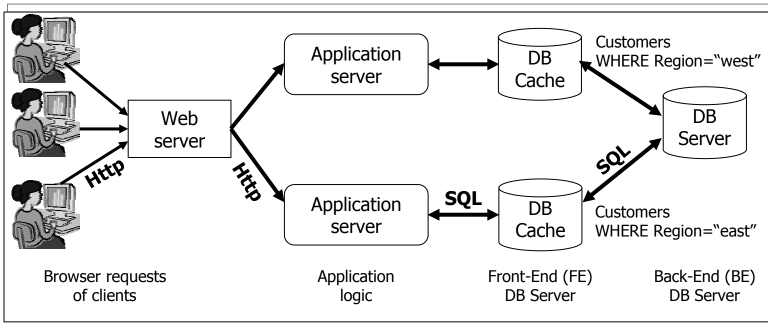

```
template {
<esi:include src=o6>
<esi:include src=o5/>
<esi:include src=o4/>
<esi:include src=o3/>
}
```

How much data has to be transferred?

	NY Times	India Times	Slashdot
Template Size	17 KB	15 KB	1.7 KB
Avg. Object Size	3.6 KB	4.8 KB	0.6 KB
Mapping Table Size	1.0 KB	0.8 KB	2.2 KB

Web caching
Approaches to DB caching
DBProxy
DBCache
Adaptive and constraint-based
Cost considerations
Implementation of ACCache
Update models for caches

Caching of DB Data

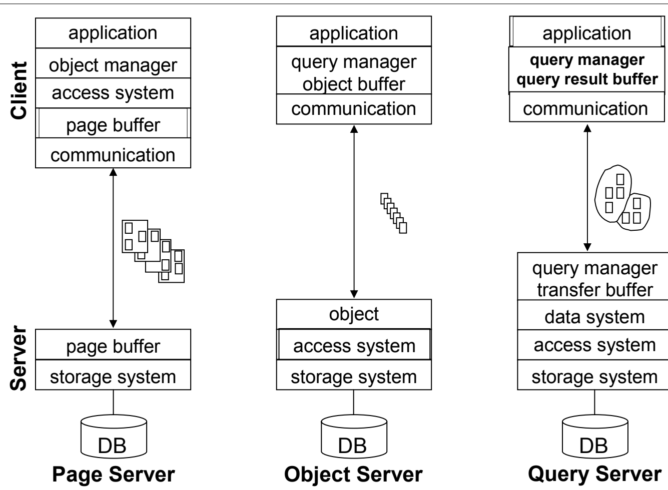


- Caching is a proven technique to improve
  - scalability and performance of large, distributed systems
  - response time and availability for the user
- DB caching near the application servers (similar to the SAP solution, but more general)
- New challenges
  - evaluation of descriptive (partial) DB queries in the cache
  - execution of update transactions?

Trends in DBS

Web caching
Approaches to DB caching
DBProxy
DBCache
Adaptive and constraint-based
Cost considerations
Implementation of ACCache
Update models for caches

Vertical Distribution of DBMS Services



- Vertical distribution of layers: DB functionality in Server and Client
- Seamless connection with the application program, but many clients
- Caching of predicate-complete DB subsets enabling PSJ operations!

Approaches to DB Caching

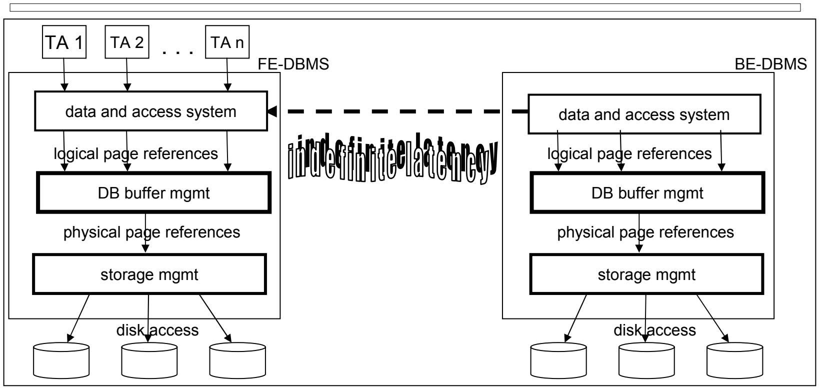
- Classical approaches
  - declarative caching (static), full-table caching, etc. not cost-effective and adaptable
  - materialized views V_i
    - queries typically refer to single cache tables: PS queries
    - result of query Q_A is contained in V_i, if predicate P_A is logically implied by predicate P_i (subsumption) and if the output attributes of O_A are contained in the OV_i attributes
    - adaptive?

- Constraint-based caching
  - specification of parameterized predicates: adaptable to workload
  - constraints are guaranteed by the cache mgr
  - evaluation of PSJ queries in the cache on the resp. predicate extension
  - further application of Group-By, Having, Order-By, and other predicates

Trends in DBS

Web caching
Approaches to DB caching
DBProxy
DBCache
Adaptive and constraint-based
Cost considerations
Implementation of ACCache
Update models for caches

DB Caching – Crucial Differences!



- Conceptual requirements
  - data model dependent
  - dynamic
  - **adaptive**
- Performance requirements
  - DB-Caching close to the application
  - minimization of contact to BE-DBMS, e.g., no **ONC via wide-area networks**

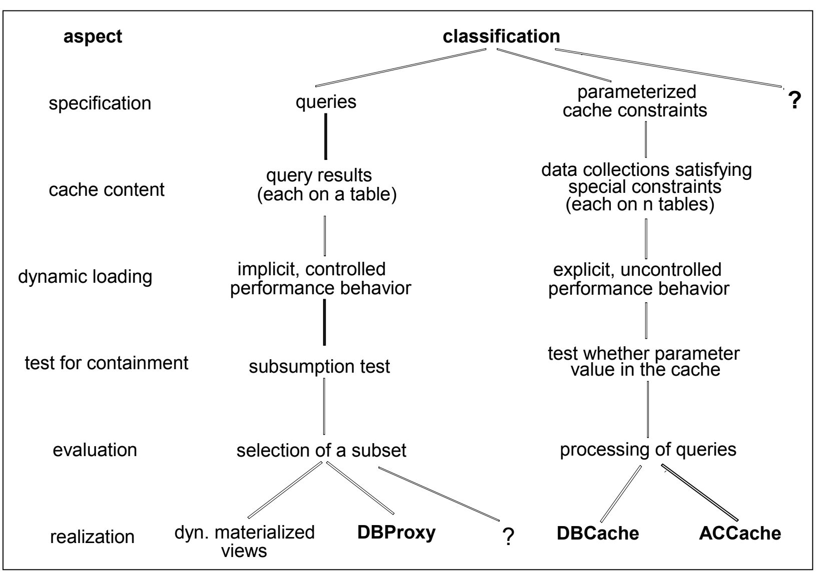
- Content of FE-DB must enable evaluation of **(partial) queries**
- DB modifications (at first) in the BE-DBMS
- Data consistency
  - often reference to staled DB-states
  - query can see several DB-states

DB Caching – What has to be Decided?

- What should the cache be used for?
- What should be kept in the cache?
- How should the cache content be specified?
- When are data loaded into the cache?
- Are overlapping data sets allowed in the cache?
- When will data be updated in the cache in case of BE updates?
- When will data be replaced resp. invalidated in the cache?

Trends in DBS
Web caching
Approaches to DB caching
DBProxy
DBCache
Adaptive and constraint-based
Cost considerations
Implementation of ACCache
Update models for caches

Classification of DB-Caching Approaches

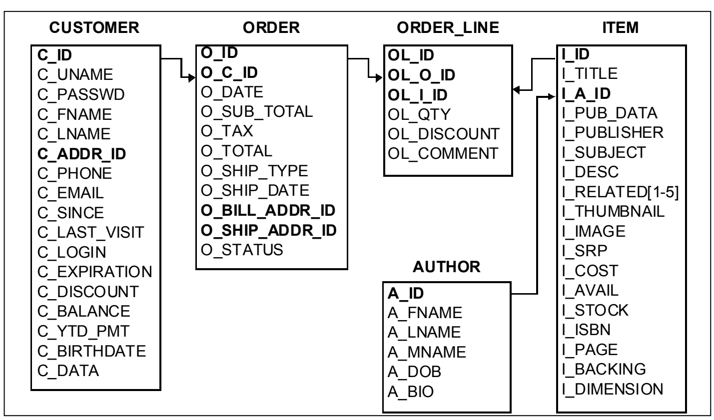


DBIS Datenbanken und Informationssysteme
© 2005 AG DBIS

DBProxy – Research Project (IBM Yorktown)

- What? – specification of cache content
  - dynamic decision, which views are worth to be cached
  - by a list of queries stored in a cache index
- When and where? – storing records in the cache
  - when the first query references the desired view
  - common storage of views
    - avoidance of replication
    - "common-schema storage-table policy"
  - nevertheless, there will exist overlapping tables (replicates in the cache)!
- How? – queries evaluated in the cache
  - reference single FE tables
  - if subsumption test is successful!

DBProxy – Simplified Schema of a Web Bookstore



- Table BE item
  - has 18 columns using i_id as primary key
  - query results concerning cost and srp (suggested retail price) are kept in table FE item in the cache

DBProxy – Methodological Aspects

- Queries
  - are rewritten in such a way that i_id is included;
  - avoidance of duplicates in a FE table
- Inserts
  - have to check whether FE table has to be extended by columns
  - optimization: definition of an "enclosing" table using pre-knowledge
- Storing various queries
  - causes undefined column values in a FE table ("fake" NULL values)
  - later evaluations in the cache are not allowed to use them!

DBProxy – Example

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 – Example with an Enclosing Table

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 – Subsumption Test for Queries

- Types of containment queries
  - completely contained: view of cache predicate Q_i
  - contained in Q_i and Q_j
  - only partially contained in one or several views kept in the cache
- Containment of Q_B (matching alg.)
  - result of Q_B is contained in that of Q_A, if the WHERE predicate of Q_B logically implies that of Q_A for all possible values of the table, e. g., of item
    +
    set of columns needed is contained in the table of Q_A or in the enclosing (but not dynamically extended) table
  - Q_B.wherep → Q_A.wherep   equivalent to the statement

> "Q_B.wherep AND (NOT (Q_A.wherep)) is not satisfiable"
>
> (i_cost < 5 AND NOT (i_cost > 25))
>
> ➔ therefore, Q_B is not contained in Q_A

DBProxy – Modification of DB Data

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
770	15	30	NULL
880	15	40	NULL

Q_1 (rows 5, 120, 340) · Q_2 (rows 340, 450, 620)

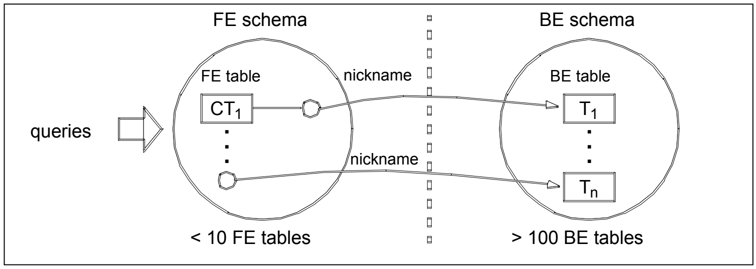
inserted by the consistency protocol

- DB data should change only slowly
  - time delay until updates reach a cache δ-consistency is guaranteed
  - updates in the DB server require the propagation of affected records possibly to several FE DBs
- Replacement or invalidation
  - only selective replacement of records
    - for Q_2 only i_id = {450, 620}
    - value abc must set to NULL
  - very complex in general, because overlapping sets of records (Q_i) are to be replaced

DBCache – Research Project (IBM ARC)

- What? – specification of cache content
  - selection of the BE tables for which FE tables are installed
  - specification of **Cache Constraints** (CCs)
    - **Cache Keys**
    - **Referential Cache Constraints** (RCCs)
- When and where? – storing records in the cache
  - when desired data is not found in the cache, the query is evaluated in the BE. Subsequently, the corresponding records are loaded into the cache
  - data collections are stored free of replication as **cache groups**
- How? – PSJ queries evaluated in the cache
  - cache groups contain **predicate extensions** (parameterized CCs)
  - allow for simple **equality predicates** and **equi-joins**

DBCache (2)



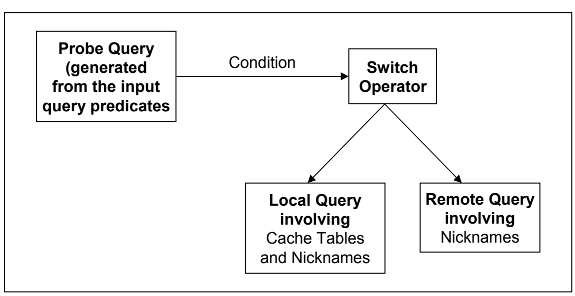
- Transparent use of DBCache
  - each BE table is existing in the FE schema as CT_i or as nickname
    - same names for schemata and elements
    - same number und types of columns
  - in this way, other relevant BE objects can be managed in the cache for each CT_i
    - logical objects: views, functions, constraints, stored procedures
    - physical objects: indexes

Trends in DBS

Web caching
Approaches to DB caching
DBProxy
DBCache
Adaptive and constraint-based
Cost considerations
Implementation of ACCache
Update models for caches

© 2005 AG DBIS

DBCache – Janus Plans



- query plan at first as "remote plan" only using nicknames
- generating of a Probe Query to test all possible equality predicates
- "Cloning" of the query graph where as many CT_i as possible are used, Local Query can be executed as federated query in DB2
- Remote Query is used if highest consistency is required

DBCache – Definitions

- Domain completeness (DC) of a column
  - if a column value is found in the cache, the cache mgr guarantees that all records having this column value are in the cache.
  - Unique (U) columns are automatically domain complete
  - DC of Non-Unique (NU) columns is enforced by the cache mgr
  - as a consequence, the evaluation of an **equality predicate**
    <ColName> = <value>
    is supported by such a column!

- Cache Key
  - can be specified for an FE table
  - is related to a column and serves as a "filling point"
  - owns the property **"domain complete"**

Trends in DBS

Web caching | Approaches to DB caching | DBProxy | DBCache | Adaptive and constraint-based | Cost considerations | Implementation of ACCache | Update models for caches

Controlled Cache Filling

- cache keys:
  CType and CLoc
- cache key rule:
  at most one cache key may be of type NU
- why?
- assume query with
  'CType = platinum'
- an FE table may carry up to n cache Keys

	U	NU	NU	U
BE_C	Cno	CType	CLoc	Cid
	1	silver	SF	NULL
	2	silver	LA	a
	3	platinum	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	platinum	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

Trends in DBS

Web caching | Approaches to DB caching | DBProxy | DBCache | Adaptive and constraint-based | Cost considerations | Implementation of ACCache | Update models for caches

Cache Table Filling – Use of Cache Key Values

- FE-CUST has defined 2 cache keys: CType and Cno
- when the cache is empty, the query Where 'CType=gold' … has the following effect:
- query with 'Cid=a14' is evaluated in the cache; however, it would not result in cache filling
- cache filling after a query with 'Cno=789'

	U	NU	NU	U
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
	. . .	. . .	. . .	

FE_CUST	Cno	CType	CLoc	Cid …
	123	gold	SJ	a14
	456	gold	SF	b21
	789	silver	NY	NULL
	891	silver	LA	d07

Trends in DBS

Web caching | Approaches to DB caching | DBProxy | DBCache | Adaptive and constraint-based | Cost considerations | Implementation of ACCache | Update models for caches

Cache Groups

- Goal: processing of equi-joins in the cache
  - value-based relationships among cache tables (a kind of value-based table model) are enforced by the cache mgr: constraints for EP_i and EJ_k!
- Evaluation of a PSJ query

  ((EP_1 or … or EP_n) and EJ_1 and … and EJ_m)

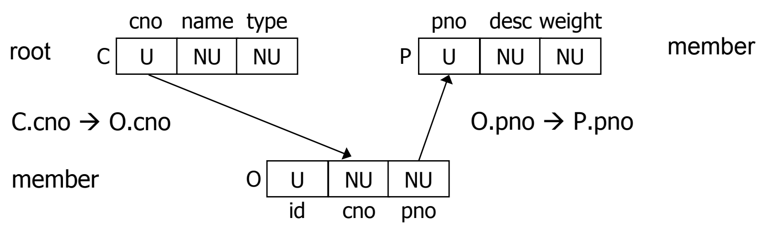
  is enabled by a specific cache group

Def.: Cache group
A cache group is a collection of cache tables linked by a set of RCCs. A distinguished cache table is called the root table R of the cache group and holds i cache keys (i>=1). The remaining cache tables are called member tables and must be reachable from R via the (paths of) RCCs.

Since full DB functionality is available, the results of queries can further be evaluated by aggregation functions such as *sum* or refined by selection predicates such as *like* or *null*, as well as by processing options like *distinct, order by, group by,* or *having*.

Cache Groups – Specification

- Referential cache constraints (RCCs) refer to pairs of columns of (normally) separate tables and are of type
  - U → U (1:1)
  - U → NU (1:n, e. g. PK/FK relationship or member constraint (MC))
  - NU → U (n:1, e. g. FK/PK relationship or owner constraint (OC))
  - NU → NU (n:m, e. g. cross constraint (XC))
- Example COP



Trends in DBS
Web caching
Approaches to DB caching
DBProxy
DBCache
Adaptive and constraint-based
Cost considerations
Implementation of ACCache
Update models for caches

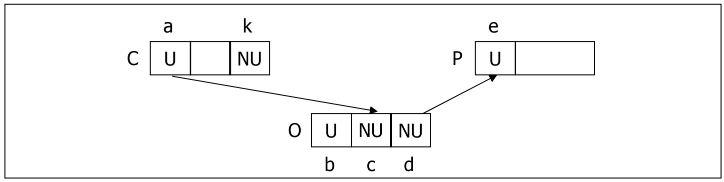
Cache Groups – Correctness

Def.: Referential cache constraint (RCC)
An RCC S.a → T.b *between two columns S.a and T.b is satisfied if and only if all values v in S.a are domain complete in T.b.*

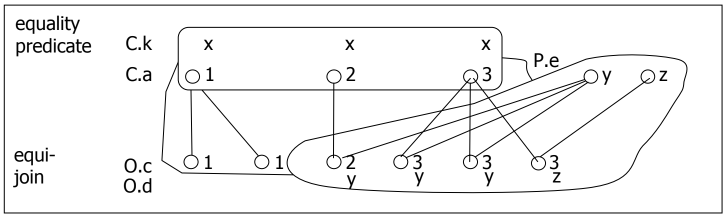
- Key property
  - cache contains only records which satisfy the specified cache constraints (CCs: RCCs and cache keys)
  - if a record is located in the cache satisfying a certain CC, then all records satisfying this CC are in the cache
- "Construction" of predicate extensions using RCCs
  - root table of a cache group owns cache key(s) (filling columns) and is connected with other FE tables by RCCs
  - an RCC guarantees that all records needed for the related **equi-join** are in the cache

Construction of Predicate Extensions

- Cache group COP



- Reference to C.k = x



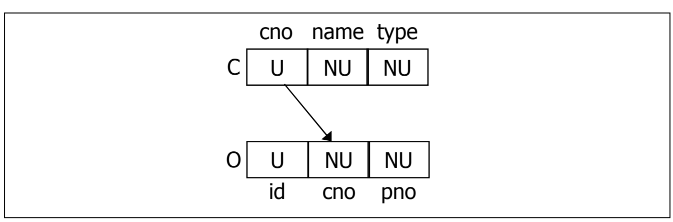
  - Where C.k = x
  - and C.a = O.c
  - and O.d = P.e

Trends in DBS

Web caching
Approaches to DB caching
DBProxy
DBCache
Adaptive and constraint-based
Cost considerations
Implementation of ACCache
Update models for caches

Cache Groups – Correctness (2)

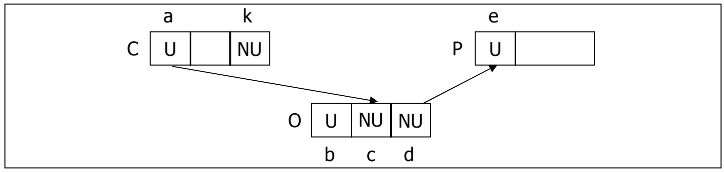
- Relationship between cache keys and domain completeness (column level)
  - CK column implies DC property, specifies **filling point**
  - however, not every DC column is CK column!
    - all columns of type U are domain complete by definition
    - some DC columns are a consequence of RCC specifications (induced domain completeness)



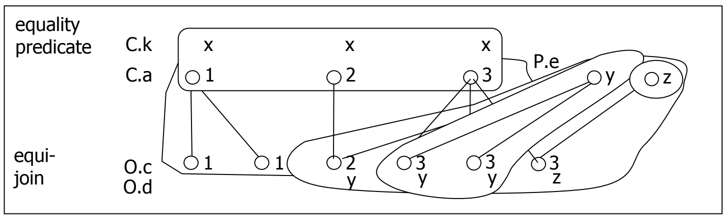
  - every DC column can be used as a potential **entry point**
    - probing checks dynamically whether a given value is present

Add-on Gain by Domain-Complete Columns

- cache group COP



- cache content



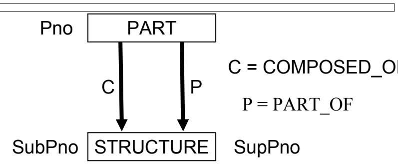
- DC columns are entry points: C.a, O.b, O.c, P.e
  - Where O.c = 3 and O.d = P.e
  - Where P.e = z
  - . . .

Trends in DBS

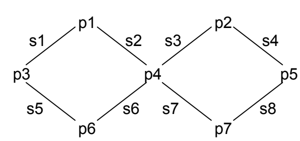
Web caching
Approaches to DB caching
DBProxy
DBCache
Adaptive and constraint-based
Cost considerations
Implementation of ACCache
Update models for caches

Recursion – Example

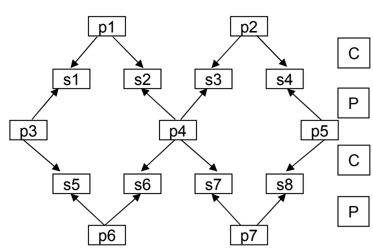
- schema of BoM
  - arrows represent PK/FK relationships and
  - are referential integrity constraints



- Gozinto graph GG1 with edge identifiers

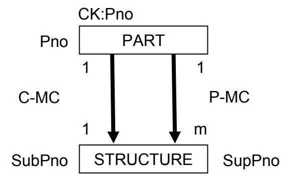


- GG1 as a graph
  - with PK/FK relationships for C and P
  - in the BE

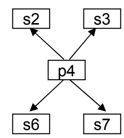


Recursion – Example (2)

- definition of cache group G1
  - MC: member constraint
  - OC: owner constraint correspond to (U → NU) and (NU → U) relationships



- assignment in FE
  - initially empty
  - 'Pno=4' (p4) be referenced in Q1

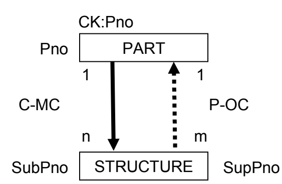


Trends in DBS

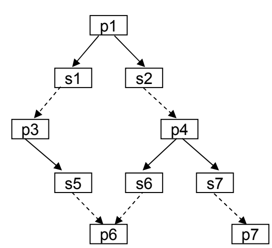
Web caching
Approaches to DB caching
DBProxy
DBCache
Adaptive and constraint-based
Cost considerations
Implementation of ACCache
Update models for caches

Recursion – Example (3)

- definition of cache group G2



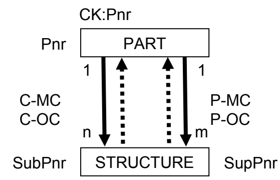
- assignment in FE
  - initially empty
  - 'Pno=1' (p1) be referenced in G2



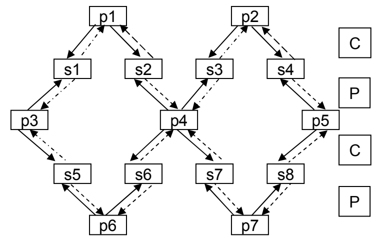
DBIS Datenbanken und Informationssysteme
© 2005 AG DBIS

Recursion – Example (4)

- definition of cache group G4



- assignment in FE
  - initially empty
  - ‘Pno=1’ (p1)
    be referenced in G4



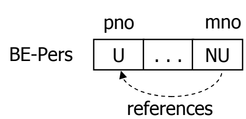
Trends in DBS

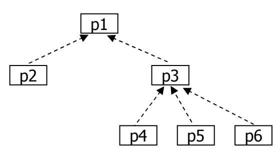
Web caching
Approaches to DB caching
DBProxy
DBCache
Adaptive and constraint-based
Cost considerations
Implementation of ACCache
Update models for caches

© 2005 AG DBIS

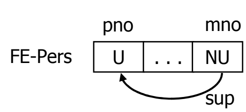
Safeness – Prevent Performance Surprises

BE schema

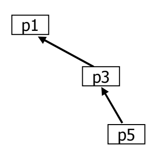


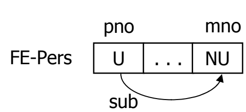


cache groups: pno is CK

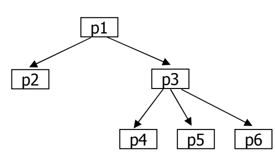


‘where pno = p5’





‘where pno = p1’



⟹ double recursion if sup and sub are specified in a cache group

Trends in DBS

Web caching
Approaches to DB caching
DBProxy
DBCache
Adaptive and constraint-based
Cost considerations
Implementation of ACCache
Update models for caches

DBCache – Conclusions

- DBCache
  - based on cache constraints at the column level (RCCs, cache keys)
  - limitation of cache group use
    - orthogonality between concepts is missing
      - dependency among DC and cache keys may enforce unwanted records into the cache
    - not all cache-evaluatable queries are recognized
    - safeness requirements are overly restrictive
  - poor adaptivity
    - DC of cache keys may enforce loading records with low reference probability or unloading those with high re-reference probability

  ⟹ concepts are too coarse

- Refinement of DBCache concepts needed
  - mechanisms to refine domain completeness and to effectively restrict its value set are important to "survive"
  - separation of value and column completeness
  - improved filling and probing mechanism
  - relationship between
    candidate values and value completeness (instance level) instead of
    cache keys and domain completeness (column level)

Trends in DBS

Web caching
Approaches to DB caching
DBProxy
DBCache
Adaptive and constraint-based
Cost considerations
Implementation of ACCache
Update models for caches

ACCache – Adaptive Constraint-Based Caching

- Reuse and adaptation of DBCache concepts
  - RCC
  - cache group
  - but: top-down approach
- Goal: evaluation of a given predicate in the cache
  - cache mgr uses cache constraints to guarantee predicate completeness

> **Def.: Predicate completeness**
> *A collection of tables is said to be predicate complete w.r.t. predicate Q if it contains all records needed to evaluate Q, i.e., its predicate extension.*

  - simplest form: equality predicates
    their evaluation requires value completeness

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

Trends in DBS

Web caching
Approaches to DB caching
DBProxy
DBCache
Adaptive and constraint-based
Cost considerations
Implementation of ACCache
Update models for caches

ACCache – Adaptive Constraint-Based Caching (2)

- Evaluation of range predicates
  - a more general type of completeness condition has to assure that all values of a *specified interval* are value complete
  - restricted to ordered types (e.g., integer or strings)

Def.: Interval completeness
An interval r is said to be interval complete (complete for short) in a column S.c if and only if all records of $\sigma_{c \in r} S_B$ *are in S (making all individual values of r value complete in S.c).*

- Equi-join predicates
  - require the use of RCCs: (adapted from DBCache)
  - filling mechanism already discussed

Def.: Referential cache constraint (RCC)
An RCC S.a → T.b *from a source column S.a to a target column T.b is satisfied if and only if all values v in S.a are value complete in T.b.*

Trends in DBS

Web caching
Approaches to DB caching
DBProxy
DBCache
Adaptive and constraint-based
Cost considerations
Implementation of ACCache
Update models for caches

Probing for Entry Points

- Observations
  - RCCs allow us to draw conclusions about predicate extensions that are in the cache, but only if we can rely on some value being complete and serving as an entry point.
  - U columns are value complete
  - only NU columns need to be considered carefully
- Gaining control over complete values
  - choose a set $C_{S.f}$ of values out of the domain of S.f and enforce completeness of these values whenever they appear in the cache
  - then existence of a value in the cache is sufficient and can be checked by a probe query
- Simplest case:
  - list with all domain values: (cache key in DBCache)
  - achieves column completeness (~ domain completeness)

Def.: Column completeness (CC)
A column S.c is said to be column complete (complete for short) if and only if all values v in S.c are value complete.

Probing for Entry Points (2)

- Refinement needed
  - complete columns for probing and filling facilitate cache management
  - but: low-selectivity columns or single values in columns with skewed value distributions may cause cache filling actions involving huge sets of records never used later

- Observations
  - the source column of an RCC controls which values are complete in its target column
  - a column T.b can have an arbitrary number of control columns (incl. zero). Whenever a column S.c we would like to use as an entry point for a predicate S.c = v has at least one control column, we have the option of probing in the control columns of S.c

Def.: Control column
S.a is a control column of T.b if there is an RCC S.a → T.b.

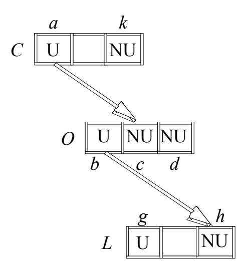
  - this alternative makes the cache usable in a more flexible way than performed in DBCache

Trends in DBS

Web caching
Approaches to DB caching
DBProxy
DBCache
Adaptive and constraint-based
Cost considerations
Implementation of ACCache
Update models for caches

Probing for Entry Points (3)

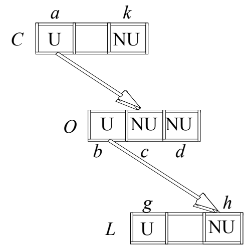
- Example COL: designed for C.k = x and C.a = O.c and O.b = L.h



  - if probing verifies the existence of single values for C.a = 1, O.b = α or L.g = z, respectively, we can evaluate, in addition to the predicate type COL is designed for, the three predicates
    - C.a = 1 and C.a = O.c and O.b = L.h
    - O.b = α and O.b = L.h
    - L.g = z
  - since O.c has C.a as a control column, we can evaluate O.c = 3 and O.b = L.h, if we probe successfully for the value C.a = 3

Negative Caching

- Further advantage of control columns
  - they enable negative caching, that is, the representation of knowledge in the cache that something does not exist, cannot or does not give an answer



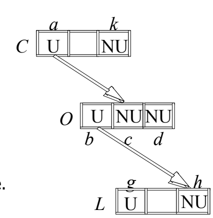
  - assume, we have a customer C.a = 1 that has not placed any orders yet (i.e., there are no records in O_B where O_B.c = 1)
  - direct probing: P' = (O.c = 1 and O.b = L.h) can only be evaluated in the cache, if probing verifies the existence of value 1 in O.c
  - probing in control column: if we probe in C.a instead and find value 1 in the cache, we then know that it must be complete in O.c

Probing Strategies

- Probing alternatives
  - if S.c is column complete, we can probe directly in S.c
  - if S.c has at least one incoming RCC, we can probe in a control column of S.c.

  ⇒ selection of a probing alternative based on the probing costs, e.g., is there an index on the probed column?

- Strategies to **minimize average costs** (regard order and probabilities)
  - if column S.c is not (always) complete, the best to do is this:
    - probe in each control column of S.c in turn.
      - if v is found, success (S.c is an entry point).
      - otherwise failure (S.c cannot be used as entry point).
  - if column S.c is always complete, a more sophisticated probing strategy is possible
    - probe in S.c.
      - If v is found, success.
      - if negative caching is impossible or not cared about, failure.
    - probe in each control column of S.c in turn.
      - if v is found, success (negative caching).
      - otherwise failure.



If referential integrity constraints valid in the backend DB are known by the cache manager, it can immediately exclude negative caching in some cases (e.g., when an owner constraint in the cache corresponds to a foreign key constraint in the backend DB)

Filling the Cache

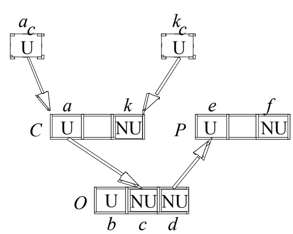
- Parameterized loading mechanism
  - use of selected filling columns
  - together with sets of candidate values

- Candidate values (CVs)
  - should have high re-reference probabilities
  - specified by the DBA or by monitoring Web requests
  - when a CV is used in a query, its extension is loaded into the cache

Def.: Candidate value
A candidate value v for a filling column S.f belongs to the domain of $S_B.f$
Whenever the predicate S.f = v is referenced by a query,
v is made value complete in S.f

- CVs can be specified as a list or a range of values or by some other predicate
- CVs can be expressed positively (recommendations) or negatively (stop-words)
- the set of all candidate values of a filling column is denoted by $C_{S.f}$

Complete Columns

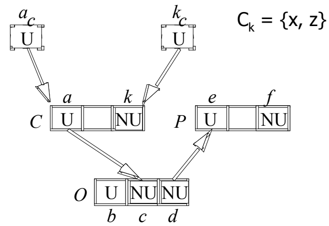


- Artificial control columns a_c and k_c
  - allow uniform probing mechanisms
  - enable selected filling via single control columns (decouple the filling mechanism and make the cache key rule obsolete)

- Completeness of NU columns
  - C.k is not complete
  - O.c is complete; additional RCCs ending in O.c would not abolish the completeness of O.c, though any additional RCC ending in a different column would

⇒ A cache column S.c is complete if it is the only column of S that is loaded via one or more RCCs.

Independence of Control Columns



- Interplay of filling columns, candidate values, and control columns a_c and k_c
  - assume two predicates C.k = v with different values v arriving at the cache and producing cache misses
  - value v = x is a candidate value. Because of the cache miss it is inserted into k_c; this makes x complete in C.k and subsequently loads all dependent records into the cache
  - value v = y is not a candidate value. Despite the cache miss it is neither inserted into k_c nor made complete in C.k

⇨ If a new record with C.k = z is loaded into the cache due to a cache miss on filling column C.a, z is not made complete, because the cache miss did not occur on C.k.

Cache Groups - Summary

- Evaluation of a PSJ query
  ((EP_1 or … or EP_n) and EJ_1 and … and EJ_m)

  is enabled by a specific cache group

> **Def.: Cache group**
> *A cache group is a collection of cache tables linked by a set of RCCs. A distinguished cache table is called the root table R of the cache group and holds i filling columns (i>=1). The remaining cache tables are called member tables and must be reachable from R via the (paths of) RCCs.*

- A column T.c is a potential entry point if
  - it has control columns (i.e., it has incoming RCCs) or
  - it is a complete column

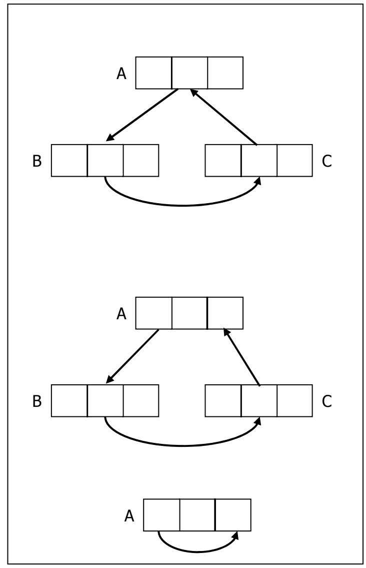
- A column T.c is complete (at all times) if
  - it is a U column,
  - it is a column with an (self-)RCC T.c → T.c, or
  - it is the only column in table T with incoming RCCs

Trends in DBS

Web caching
Approaches to DB caching
DBProxy
DBCache
Adaptive and constraint-based
Cost considerations
Implementation of ACCache
Update models for caches

Cache Groups – Controlled Filling (Safeness)

- Homogeneous RCC cycles
  - if the BE tables contain the same set of values for the cycle columns C_A, C_B, and C_C, these columns are CC
  - filling stops after a single round – no recursive load behavior
- Heterogeneous RCC cycles
  - NU or U columns in a cycle can provoke recursive cache filling
  - possible on a single table
  - heterogeneous RCCs are prohibited



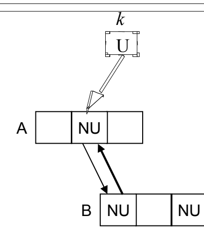
Trends in DBS

Web caching
Approaches to DB caching
DBProxy
DBCache
Adaptive and constraint-based
Cost considerations
Implementation of ACCache
Update models for caches

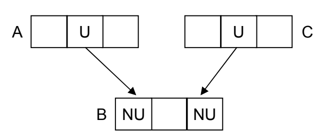
DBIS Datenbanken und Informationssysteme
© 2005 AG DBIS

Cache Groups – Controlled Filling (2)

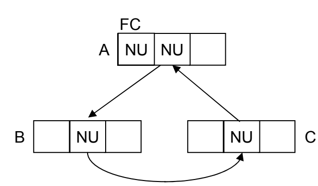
- Column completeness
  - UNIQUE columns (explicit)
  - a single filling column if there is no incoming RCC on another column
  - a column if it owns the only incoming RCC(s)
  - AB-XC and BA-XC hold if B is not filled via other paths



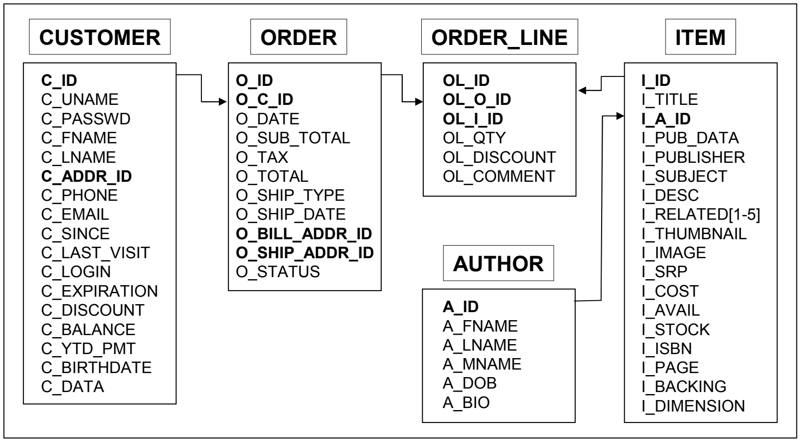
- No column completeness
  - B is also filled via other paths



- Recursion-free filling of G
  - **no heterogeneous RCC cycles** in G
  - each table is involved in at most one homogeneous NU cycle
  - filling columns (U/NU) next to the columns of a homogeneous NU cycle do not matter

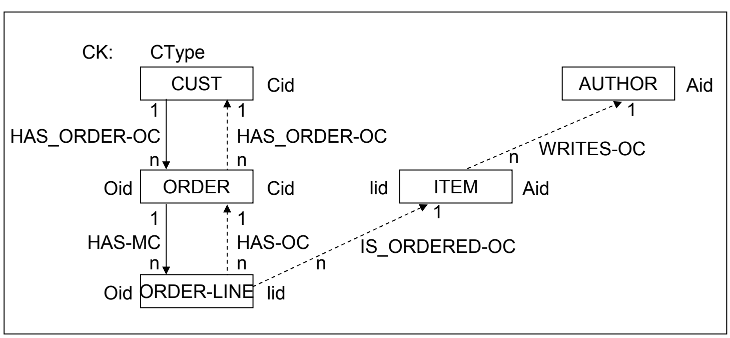


Simplified Schema of a Web Bookseller



- collection of tables, for example
  - BE-item has 18 columns using i_id as primary key
  - simplest solution: 1:1 correspondence to table FE-item in the cache

Cache Groups – Application Example



- specification of a (maximal) cache group for the Web bookseller
- cache group entry via a equality predicate test (probing)
- when probing is successful, each RCC guarantees that the related join can be evaluated in the cache

Cache Filling – Cost Considerations

	a		k		b
O	U		NU		NU

a single value is filled into O.k where O.k is a cache key

- Standard assumption of query processing
  - uniform value distribution in columns
  - stochastic independence between columns
  - cardinalities of columns and tables in the backend are known
- How many records are loaded into O?

$$n_O = N_O / c_{O.k}$$

- How many new values arrive in O.b?
  (what is the expected number d of colors when n balls are drawn without replacement from a bucket with N balls (in c different and uniformly distributed colors)?)

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

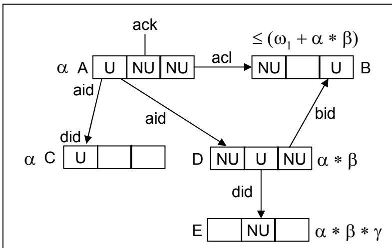
Trends in DBS

Web caching
Approaches to DB caching
DBProxy
DBCache
Adaptive and constraint-based
Cost considerations
Implementation of ACCache
Update models for caches

DBIS Datenbanken und Informationssysteme
© 2005 AG DBIS

Cache Groups – Filling Behavior

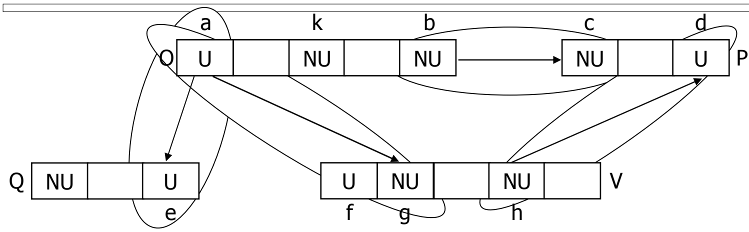
- referencing a candidate value
  - each value of ack occurs in α records
  - RCC of type U→ NU inserts β resp. γ records per owner
  - which effect has RCC of type NU → NU ?
- how many distinct values are to be expected in a NU column, if a value of a DC column enters the cache?
  - n = α
  - c = Card (acl)
  - N = N_A
- $\omega_1 = d * N_B / c$



reachability graph of a sample cache group G1

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

Cache Filling – Cost Considerations



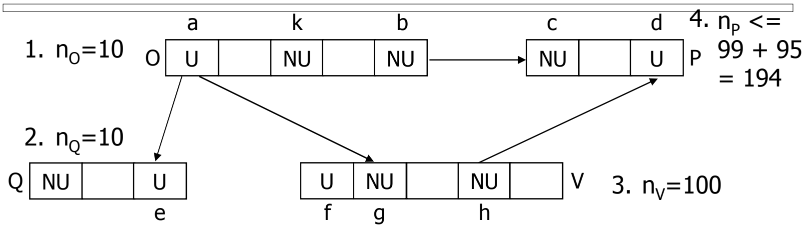
- U → U $\quad n_Q = n_O \cdot N_Q / c_{Q.e} \quad (n_Q \le n_O)$
- U → NU $\quad n_V = n_O \cdot N_V / c_{V.g}$
- NU → U $\quad n_{P2} = f(N_V, c_{V.h}, n_V) \cdot N_P / c_{P.d}$
- NU → NU $\quad n_{P1} \le n_{O.b} \cdot N_P / c_{P.c} = f(N_O, c_{O.b}, n_O) \cdot N_P / c_{P.c}$
- because of duplicates loaded into P

$$n_P = n_{P1} + n_{P2} - n_{P1} \cdot n_{P2} / N_P$$

General Scheme for Table Population Induced by an RCC

target table filling n_T		target column T.j unique	target column T.j non-unique
source column S.i	unique	$n_S \cdot 1$	$n_S \cdot \frac{N_T}{c_{T.j}}$
	non-unique domain complete	$n_S \frac{c_{S.i}}{N_S} \cdot 1$	$n_S \frac{c_{S.i}}{N_S} \cdot \frac{N_T}{c_{T.j}}$
	non-unique, non-domain-complete	$f(N_S, c_{S.i}, n_S) \cdot 1$	$f(N_S, c_{S.i}, n_S) \cdot \frac{N_T}{c_{T.j}}$

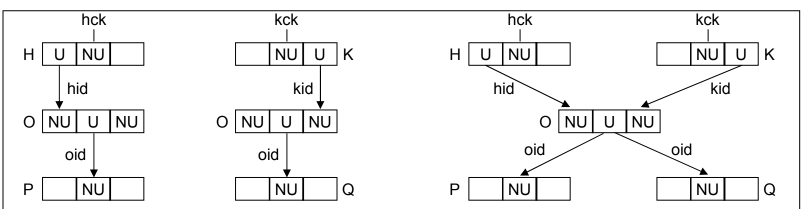
Evaluation of Single Cache Groups



Assumptions: N_O, N_Q, N_P = 10^4 ; N_V = 10^5 ; NU cols: card = 10^3 ($c_{V.g}$= 10^4)

1. List the topological sort order of *G*'s tables in TSO.
2. Compute the expected population n_S of root table using the cardinality of c_{keff}.
3. While there are tables in TSO not visited,
   visit the next table T and obtain the expected table population n_T:
   a) For each incoming RCC R_i , compute the population n_{Ti} using the already computed n_{Si} of its source table, the type of R_i, and the cardinality of its target column $c_{T.i}$.
   b) n_T is obtained by applying the combination-of-events model using all determined n_{Ti}.

Cache Group Federations – Saving and Penalty



c) Reachability graphs of cache group G2 and G3 d) Connectivity graph of federation F

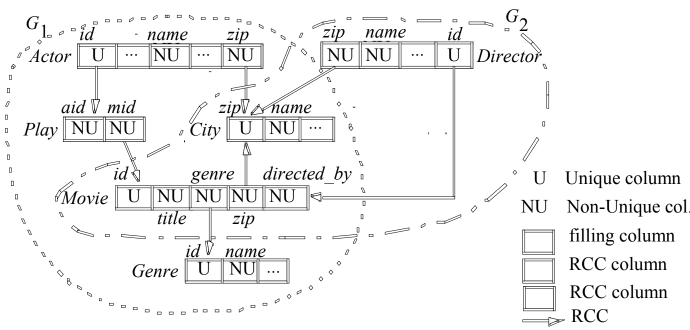
- Cost for
  - G2: $n_{G2} = \alpha_2 * (1 + \beta_2 * (1 + \gamma_2))$ = 1110
  - G3: $n_{G3} = \alpha_3 * (1 + \beta_3 * (1 + \gamma_3))$ = 1110
- Saving/penalty for F
  - $\Delta n_{O|F} = - \alpha_2 * \beta_2 * \alpha_3 * \beta_3 / N_O$ = -1
    $\Delta n_{P|F} = \alpha_3 * \beta_3 * \gamma_2$ = 1000
  - $\Delta n_{Q|F} = \alpha_2 * \beta_2 * \gamma_3$ = 1000

Web caching
Approaches to DB caching
DBProxy
DBCache
Adaptive and constraint-based
Cost considerations
Implementation of ACCache
Update models for caches

Implementation of ACCache

- What specific tasks have to be provided by a DB caching system?

- Running example



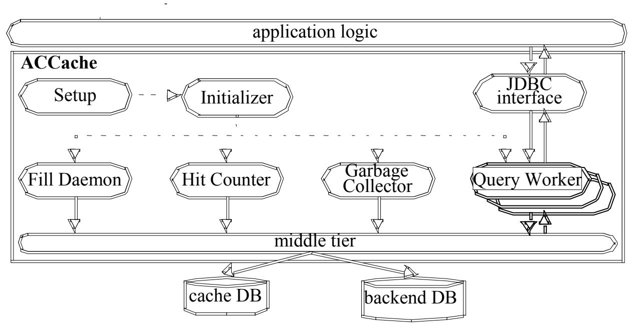
Trends in DBS

Web caching
Approaches to DB caching
DBProxy
DBCache
Adaptive and constraint-based
Cost considerations
Implementation of ACCache
Update models for caches

ACCache – Implementation Considerations

- Development of an adequate architecture
  - we strive for a solution which is independent of a specific DBMS and exclusively rests on the availability of some SQL engine
  - therefore, we go for a flexible solution based on middleware concepts.
  - we try to combine the advantages of different existing systems

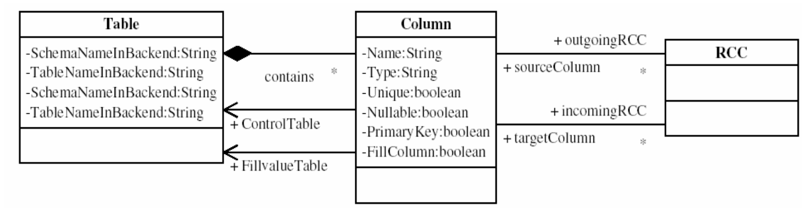
- Overall architecture



DBIS Datenbanken und Informationssysteme
© 2005 AG DBIS

ACCache – Implementation Considerations

- Initializing DB cache processing
  - creation and specification of appropriate meta-data



  - allocation of the cache tables and their related control tables
  - specification of filling value tables
  - creation of appropriate indexes

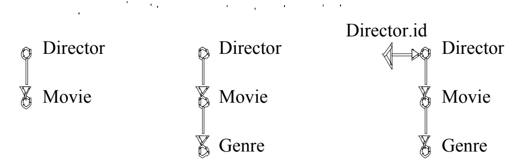
- Setup of prepared statements
  - existence queries used by the Query Workers for probing
  - insert statements used by the Fill Daemon to load new records
  - update queries to modify information in control tables
  - delete statements to unload records from cache tables

Query Worker

- QW is responsible for processing user queries
  - incoming queries are validated against an SQL grammar (subset)
  - assume the following potential candidate for cache processing

```
SELECT d.name, m.title, g.name FROM Director d, Movie m, Genre g
WHERE d.id = m.directed_by AND m.genre = g.id AND d.id = '711'
ORDER BY g.name ASC
```

  - after checking for correct syntax, the query is decomposed into its different clauses
  - for equi-join predicates, it checks for existing RCCs using the meta-data and builds a cache group evaluation graph (CEG)



a) after processing the first predicate
b) after processing the second predicate
c) after processing all predicates

Web caching
Approaches to DB caching
DBProxy
DBCache
Adaptive and constraint-based
Cost considerations
Implementation of ACCache
Update models for caches

Query Worker (2)

- Query considered

  SELECT d.name, m.title, g.name FROM Director d, Movie m, Genre g
  WHERE d.id = m.directed_by AND m.genre = g.id AND d.id = '711'
  ORDER BY g.name ASC

  - prepared statement for related existence queries for probing potential entry points have the form

  SELECT 1 FROM TABLE (VALUES 1) AS tmp
  WHERE EXISTS (SELECT * FROM <cache table> WHERE column = ?)

  - assume, probing is successful for equality predicate d.id = '711'. Then, column d.id can be used as an entry point (added as an anchor to CEG)
  - CEG enables the generation of the modified query to be evaluated in the cache

  SELECT d.name, m.title, g.name
  FROM CA_Director d, CA_Movie m, CA_Genre g
  WHERE d.id = m.directed_by AND m.genre = g.id AND d.id = '711'
  ORDER BY g.name ASC

Trends in DBS

Web caching
Approaches to DB caching
DBProxy
DBCache
Adaptive and constraint-based
Cost considerations
Implementation of ACCache
Update models for caches

Fill Daemon

- If probing fails,
  - the value looked up is not complete in the cache and the query has to be processed in the BE DB
  - if the value is a CV, a msg is sent to the FD to load it into the cache
  - assume actor name 'Bond' is a CV and the resp. predicate extension has to be loaded:
    - loading the related Actor records force Play and City records into the cache.
    - the inserted Play records require the filling of Movie records and these, in turn, Genre and City records
    - beware of duplicates!
- Top-down filling
  - the filling process iteratively loads a sequence of cache tables starting with the control table
  - as an example, we list the insert statement for the Actor table:

  INSERT INTO CA_Actor SELECT * FROM Actor a WHERE a.name = 'Bond'
  AND a.name NOT IN (SELECT name FROM CA_Actor)

⟹ long X locks must be kept until filling is successfully finished!

DBIS Datenbanken und Informationssysteme

Fill Daemon (2)

- Key observation
  - when loading the cache tables bottom-up, we can fill each table in a separate transaction thereby providing cache consistency and only need to lock until the resp. cache table is loaded
  - more precisely, we have to define so-called atomic zones which can be loaded independently
  - in the simplest case, when no cycles are present, each table is an atomic zone
- Bottom-up filling
  - determine the loading sequence of the zones by topological sorting
  - in the example. (Genre, City), Movie, Play, Actor, and finally the control table A.name
  - reverse RCC path be R_n, R_{n-1}, …, R_1 where the target table to be filled is R_n and the source table of R_1 is the root table
  - generic form of prepared insert statements:

INSERT INTO <cache table> (
SELECT * FROM <corresponding backend table> WHERE <R_n target col.> IN (
SELECT <R_n source col.> FROM <R_n source table> WHERE <R_{n-1} target col.> IN (
. . . (SELECT <R_1 source col.> FROM <R_1 source table> WHERE <filling col.> = ?))))

Trends in DBS

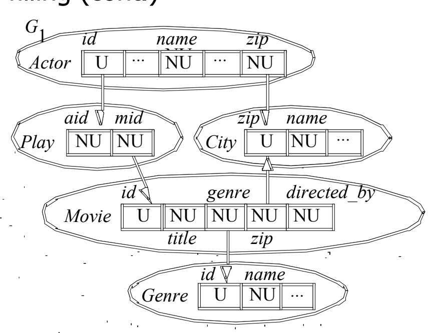
Web caching
Approaches to DB caching
DBProxy
DBCache
Adaptive and constraint-based
Cost considerations
Implementation of ACCache
Update models for caches

DBIS Datenbanken und Informationssysteme

© 2005 AG DBIS

Fill Daemon (3)

- Bottom-up filling (cont.)



  - Insert statement for the first table to be loaded

```
INSERT INTO CA_Genre SELECT * FROM Genre g WHERE g.id IN (
    SELECT m.genre FROM Movie m WHERE m.id IN (
      SELECT p.mid FROM Play p WHERE p.aid IN (
        SELECT a.id FROM Actor a WHERE a.name = 'Bond'
  ))) AND g.id NOT IN (SELECT id FROM CA_Genre)
```

⟹ trade-off between potentially higher concurrency during filling process and the need for more complex queries to be evaluated in the BE DB!

Hit Counter and Garbage Collector

- Hit Counter
  - is responsible for recording statistical data used by the Garbage Collector for its cache replacement strategy
  - records each entry point found while a query was analyzed
  - collects statistical information on CVs of $C_{S.f}$ in columns such as *hitcounter* and *lastaccess*
- Garbage Collector
  - is responsible for controlling the size of the cached data by periodically checking whether or not a pre-specified cache filling level (high-water mark) is observed
  - if necessary, it initiates one or more deletions of cache instances by removing a CV from a control table.
  - as a consequence, the entire predicate extension for the removed CV has to be deleted from the cache thereby preserving the cache constraints

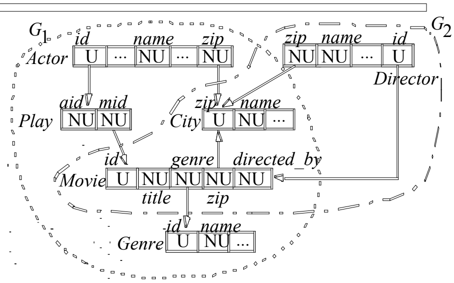
⟹ records belonging to multiple predicate extensions must not be deleted!

Trends in DBS

Web caching
Approaches to DB caching
DBProxy
DBCache
Adaptive and constraint-based
Cost considerations
Implementation of ACCache
Update models for caches

Hit Counter and Garbage Collector (2)

- Deletion
  - use of enhanced LRU
  - assume, CV ‘Spielberg’ is to be removed from G_2
  - remove CV from $C._{S.f}$ and control table K1
  - remove predicate extension for D.name = ‘Spielberg’
  - start with



DELETE FROM CA_Director WHERE (name IS NOT IN (SELECT name FROM K1))

  - deletion procedure follows the RCC paths using this base template

DELETE FROM <cache table> WHERE (<RCC target column> IS NOT IN (SELECT <RCC source column> FROM <RCC source table>))

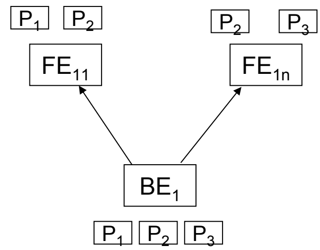
  - removing records from City with two incoming RCCs requires

DELETE FROM CA_City WHERE (zip IS NOT IN (SELECT zip FROM CA_Movie)) AND (zip IS NOT IN (SELECT zip FROM Director))

⟹ what happens to RCC-dependent records in table Genre?

Update Models for DB Caching

- Different modes of update propagation
  - **DAIA:** deferred apply – immediate apply
    - all updates in the BE
    - asynchronous propagation to the FEs
  - **IADA:** immediate apply – deferred apply
    - all updates in the FE within transaction boundaries; all records to be modified have to be in the cache
    - asynchronous propagation to the BE and subsequently to the other FEs
  - **IAIA:** immediate apply – immediate apply
    - synchronous updates in FE and BE
    - within the same transaction



IADA – Modifications in a Single FE Table

- Most simple case
  - isolated BE table
  - all modifications in the related FE table
  - primary key: Cno
    cache keys: Cid (U), CType (NU)
- Insert
  - inserts of the new records:
    Cno = 8
  - possibly reload of records to satisfy an NU-DC condition
  - Cno = 5:
    NULL semantics for DC?
  - how to test UNIQUE in the FE?
  - trigger actions must be restricted to the FE table
- Delete
  - in the cache: Cno = 8
  - data in the BE: Cno = 3, Cno = 6
    - fetch
    - delete

BE_C	Cno	CType	CLoc	Cid
	1	silver	SF	NULL
	2	silver	LA	a
	3	platinum	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	platinum	SJ	b
	6	gold	SF	e

IADA – Modifications in a Single FE Table (2)

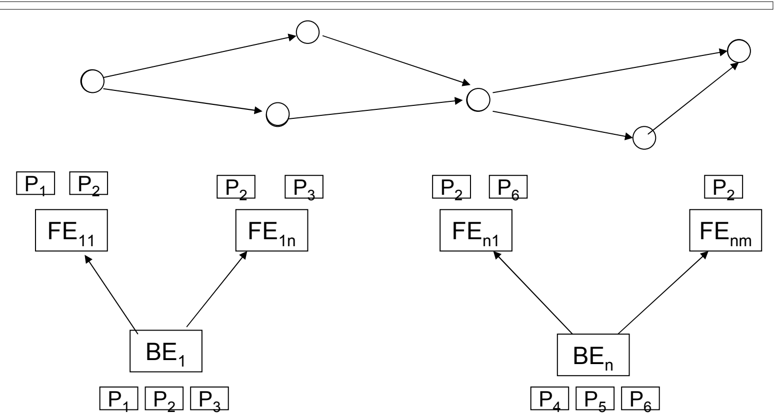
- Most simple case
  - isolated BE table
  - all modifications in the related FE table
  - primary key: Cno
    cache keys: Cid (U), CType (NU)
- Update
  - in the cache: no column NU-DC
    - Cno = 2: CLoc = NY
  - in the cache: in a column NU-DC
    - Cno = 3: CType = silver
  - data in BE: no column NU-DC
    - Cno = 4: CLoc = NY
  - data in BE: in a column NU-DC
    - Cno = 5: CType = gold

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

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

Web caching
Approaches to DB caching
DBProxy
DBCache
Adaptive and constraint-based
Cost considerations
Implementation of ACCache
Update models for caches

DB Caching – Vision



- Database management architectures
  - DDBMS
  - FDBMS
  - Grid or P2P architectures
    (peer-based data management is a natural extension of data integration, A. Halevy)

Web caching
Approaches to DB caching
DBProxy
DBCache
Adaptive and constraint-based
Cost considerations
Implementation of ACCache
Update models for caches

Trends in DBS

Web caching

Approaches to DB caching

DBProxy

DBCache

Adaptive and constraint-based

Cost considerations

Implementation of ACCache

Update models for caches

Conclusions & Outlook

- Constraint-based DB caching pushes PSJ query evaluation to the edge of the Internet
- Idea of predicate completeness can be applied to XML data models, too
- Together with native XML DBMS, it enables a seamless XML path from the data to the user

- Cache group advisor needed
  - cache group design may become complex (which tables, which RCCs, which candidate values, …)
  - only locality of reference provides gain
- Replacement (invalidation) algorithms
  - unreferenced data in the cache must be kept consistent, too
  - caching must be adaptable and workload-dependent
- Update processing in the cache
  - provision of transaction-consistent local states
  - dissemination of committed data immediately to other caches?

DBIS Datenbanken und Informationssysteme

Trends in DBS

Web caching

Approaches to DB caching

DBProxy

DBCache

Adaptive and constraint-based

Cost considerations

Implementation of ACCache

Update models for caches

Further References

- Altinel, M., Bornhövd, C., Krishnamurthy, S., Mohan, C., Pirahesh, H., Reinwald, B.: Cache tables: Paving the way for an adaptive database cache. In: Proc. VLDB Conference. (2003) 718–729
- Amiri, K., Park, S., Tewari, R., Padmanabhan, S.: DBProxy: A dynamic data cache for web applications. In: Proc. ICDE Conference. (2003) 821–831
- Bühmann, A.: Ein Schritt zurück ist kein Rückschritt, in: Informatik – Forschung und Entwicklung 20:4 (2006) 184-195
- Larson, P., Goldstein, J., Zhou, J.: MTCache: Transparent mid-tier database caching in SQL server. In: Proc. ICDE Conference, IEEE Computer Society (2004) 177–189
- Merker, C.: Konzeption und Realisierung eines Constraint-basierten Datenbank-Cache. Master's thesis, TU Kaiserslautern (2005)
- Härder, T., Bühmann, A.: Database caching – Towards a cost model for populating cache groups. In: Proc. ADBIS Conference. Volume 3255 of LNCS, Budapest, Springer (2004) 215–229

DBIS Datenbanken und Informationssysteme

© 2005 AG DBIS