AG Heterogene Informationssysteme
Prof . Dr.-Ing. Stefan Deßloch
Fachbereich Informatik
Technische Universität Kaiserslautern

Recent Developments for Data Models – Solution to Exercise 2

Monday, May 21, 2012 – 15:30 to 17:00 – Room 36-336

1) User-defined Ordering

Reconsider the typed table hierarchy introduced by the first exercise sheet. Say, the following instances appear in the hierarchy tables (some attributes omitted).

Article

pubkey	title	pdate	source_title
4	aaa	2000	bbb
6	aaa	2002	TechReport

Book

pubkey	title	pdate
2	aaa	2000
5	bbb aaa	2000

TechReport

pubkey	title	pdate
1	aaa	2002
3	aaa	2002

a. Assume that the following user-defined ordering is specified.

```
CREATE ORDERING FOR ArticleT EQUALS ONLY BY STATE;
CREATE ORDERING FOR BookT EQUALS ONLY BY STATE;
CREATE ORDERING FOR TechReportT EQUALS ONLY BY STATE;
```

Perform a pairwise comparison of the tuples! Which tuples are considered to be equal?

Only the tuples with OID 1 and 3 are considered to be equal. Note that the implicitly generated STATE function compares not only the attribute values but also the most-specific types.

b. Assume that the following user-defined ordering is specified.

```
CREATE FUNCTION publicationMap(pub PublicationT)
RETURNS VARCHAR(255)
RETURN pub.title || CAST(YEAR(pub.pdate) AS CHAR(4))

CREATE FUNCTION containedPublMap (pub ContainedPublT)
RETURNS VARCHAR(255)
RETURN pub.source_title || ' ' || pub.title ||
CAST(YEAR(pub.pdate) AS CHAR(4));

CREATE FUNCTION techReportMap (pub TechReportT)
RETURNS VARCHAR(255)
RETURN 'TechReport ' || pub.title  ||
CAST(YEAR(pub.pdate) AS CHAR(4));

CREATE ORDERING FOR PublicationT ORDER FULL BY MAP WITH
FUNCTION publicationMap(PublicationT);

CREATE ORDERING FOR ContainedPublT ORDER FULL BY MAP WITH
FUNCTION containedPublMap(ContainedPublT);

CREATE ORDERING FOR TechReportT ORDER FULL BY MAP WITH
FUNCTION techReportMap(TechReportT);
```

Perform a pairwise comparison of the tuples! Which tuples are considered to be equal?

The tuples with the following OIDs are considered to be equal: 1 and 3, 1 and 6, 3 and 6, 4 and 5.

c. Specify an ordering function using the RELATIVE ordering category that mimics the user-defined ordering specified in b) above.

```
CREATE FUNCTION helper(p PublicationT)
RETURNS VARCHAR(255)
BEGIN ATOMIC
  IF p IS OF (TechReportT) THEN
    RETURN 'TechReport' || p..title ||
           CAST(YEAR(p..pdate) AS CHAR(4));
  ELSEIF p IS OF (ContainedPublT) THEN
    RETURN TREAT(p AS ContainedPublT)..source_title || ' ' ||
           p..title || CAST(YEAR(p..pdate) AS CHAR(4));
  ELSE
    RETURN p..title || CAST(YEAR(p..pdate) AS CHAR(4));
  END IF;
END@

CREATE FUNCTION publicationRelative(a PublicationT, b Publica-
tionT)
RETURNS INTEGER
BEGIN ATOMIC
    IF helper(a) < helper(b) THEN RETURN -1;
    ELSEIF helper(a) > helper(b) THEN RETURN 1;
    ELSE RETURN 0;
    END IF;
END@

CREATE ORDERING FOR PublicationT ORDER FULL BY RELATIVE WITH
FUNCTION publicationRelative(PublicationT, PublicationT);
```

2) Object-relational Views

Reconsider the publication database's schema as introduced by the first exercise sheet. Create views to provide the following information.

a. Create a (non-typed) view to provide access statistics of publications. Include the authors' self-referencing column, the total number of accesses (local and remote) to papers of the respective author, and the average number of accesses (local and remote) per paper!

1) Specify the required view definition. Choose `authorstats` as view name.

```
create view authorstats
(author, sumLocal, sumRemote, avgLocal, avgRemote) as (
    select author as author,
     sum (publication->accesscnt.localAccesses) as sumLocal,
     sum (publication->accesscnt.remoteAccesses) as sumRemote,
     avg (publication->accesscnt.localAccesses) as avgLocal,
     avg (publication->accesscnt.remoteAccesses) as avgRemote
    from publauthors
    group by author
);
```

2) Retrieve the name and access statistics for all authors from the view.

```
select author->name, sumLocal, sumRemote, avgLocal, avgRemote
from authorstats;
```

3) Retrieve the name of the author with the greatest number of accesses (local and remote).

```
select author->name, sumLocal + sumRemote AS sumTotal
from authorstats
where (sumRemote + sumLocal) = (
    select max (sumRemote + sumLocal) from authorstats
);
```

b. Create a (typed) view hierarchy based on the user-defined structured type `PublicationT` and its subtypes (cf. exercise 1) to provide publications that are available electronically. A publication is assumed to be available electronically if its URL attribute is neither null nor empty.
Specify the required view definitions for electronically available publications and books (called `EPublication` and `EBook`, respectively)!

```
create view EPublication of PublicationT
(ref is publkey user generated) as (
    select publkey, title, url, pdate,
           accessCntT(accessCnt.localAccesses,
                      accessCnt.remoteAccesses)
    from only(publication)
    where url is not null and url <> ''
);

create view EBook of BookT under EPublication as (
    select publkey, title, url, pdate,
           accessCntT(accessCnt.localAccesses,
                      accessCnt.remoteAccesses),
           publisher
    from only(book)
    where url is not null and url <> ''
);
```

c. Create a restricted (typed) view hierarchy based on the views defined in b) that does not provide access statistics for publications. Create suitable user-defined structured types (`ResPublicationT` and `ResBookT`)! Then specify the required view definitions (`ResPublication` and `ResBook`)!

```
create type ResPublicationT as (
    title varchar(150),
    url varchar(150),
    pdate date
)
ref using varchar(20);
```

```
create type ResBookT under ResPublicationT as (
    publisher ref(PublisherT)
);

create view ResPublication of ResPublicationT
 (ref is publkey user generated) as (
    select CAST (Varchar(publkey) AS REF(ResPublicationT)),
           title, url, pdate
    from only (EPublication)
);

create view ResBook of ResBookT
under ResPublication as (
    select CAST (varchar(publkey) AS REF(ResBookT)),
           title, url, pdate, publisher
    from only (EBook)
);
```

d. The SQL-Standard does not allow for typed view hierarchies over untyped tables. What is the reason for this restriction?

Instances of typed views have an identity (an OID) just like instances in typed tables. Similarly, the uniqueness of OIDs is required in typed view hierarchies. Therefore the SQL-standard places restrictions on the `FROM` clause of the view definition. First, only a single typed table or view may be specified. Second, the `ONLY` keyword must be used to prevent tuples from subtables or subviews from being returned.

e. Nevertheless, database vendors developed extensions to address these limitations. Consider the following untyped legacy table.

```
create table legacyPubl (
    publkey integer not null primary key,
    title varchar(150),
    url varchar(150),
    pdate date,
    publisher integer references legacyPublisher(pid),
    first_page integer,
    last_page integer
);
```

This table shall be exposed in a typed view hierarchy (using DB2 object view extensions) based on the user-defined types `ContainedPublT` (as root view), `ArticleT`, and `BookChapterT`. Assume that articles do not have more than 20 pages while book chapters are longer. Specify the required view definitions (`VContainedPubl`, `VArticle`, and `VBookChapter`)!

```
create view VContainedPubl of ContainedPublT
mode db2sql
(ref is publkey user generated)
as (
    select CAST (publkey AS REF(ContainedPublT)),
           title, url, pdate, AccessCntT() as accesscnt,
           CAST (NULL as VARCHAR(200)) AS source_title,
           CAST (NULL as VARCHAR(300)) AS source_url,
           first_page, last_page,
           CAST (publisher AS REF(PublisherT))
    from legacyPubl
    where 1 = 0  -- evaluates to FALSE
);
```

```
create view VArticle of ArticleT
mode db2sql
under VContainedPubl
inherit select privileges
as (
    select CAST (publkey AS REF(ArticleT)),
           title, url, pdate, AccessCntT() as accesscnt,
           CAST (NULL as VARCHAR(200)) AS source_title,
           CAST (NULL as VARCHAR(300)) AS source_url,
           first_page, last_page,
           CAST (publisher AS REF(PublisherT)),
           CAST (NULL as INTEGER) AS volume,
           CAST (NULL as INTEGER) AS number
    from legacyPubl
    where (last_page - first_page + 1) <= 20
);

create view VBookChapter of BookChapterT
mode db2sql
under VContainedPubl
inherit select privileges
as (
    select CAST (publkey AS REF(BookChapterT)),
           title, url, pdate, AccessCntT() as accesscnt,
           CAST (NULL as VARCHAR(200)) AS source_title,
           CAST (NULL as VARCHAR(300)) AS source_url,
           first_page, last_page,
           CAST (publisher AS REF(PublisherT)),
           CAST (NULL as VARCHAR(15)) AS chapter
    from legacyPubl
    where (last_page - first_page + 1) > 20
);
```

f. The DB2 database manager cannot always decide whether OIDs are distinct for each view in a hierarchy. Why not? Is it possible to define typed view hierarchies even in such situations?

When a view is defined, local uniqueness of object id is ensured by verifying that the view body mentions only one table (or view) in its from clause and selects as the view's object id a unique key (possibly with a type cast) of that table. Global uniqueness across a hierarchy is checked through a conservative static analyses of the predicates in the user's view definitions each time a create view statement is issued.
(Michael Carey, Serge Rielau, Bennet Vance: Object View Hierarchies in DB2 UDB, EDBT, 2000, 478-492)

The UNCHECKED option defines the object identifier column of the typed view definition to assume uniqueness even though the system cannot prove this uniqueness. This is intended for use with tables or views that are being defined into a typed view hierarchy where the user knows that the data conforms to this uniqueness rule but it does not comply with the rules that allow the system to prove uniqueness. UNCHECKED option is mandatory for view hierarchies that range over multiple hierarchies or legacy tables or views By specifying UNCHECKED, the user takes responsibility for ensuring that each row of the view has a unique OID. If the user fails to ensure this property, and a view contains duplicate OID values, then a path-expression or DEREF operator involving one of the non-unique OID values may result in an error.
(IBM DB2 Database for Linux, UNIX, and Windows Information Center)

3) Composite Types and Collection Types

Consider the following table definition.

```
create table publication (
    title varchar(150),
    year char(4),
    author ROW(name varchar(35), firstname varchar(25))
        ARRAY[10],
    keyword varchar(50) MULTISET
);
```

Specify SQL queries to retrieve the following information.

a. All publications (title, year) where *Jim Melton* appears as first author.

```
select title, year
from publications
where author[1].name = 'Melton' and
      author[1].firstname = 'Jim'
```

b. Triples of publications (title), the authors (name), and the authors' positions.

```
select p.title, a.name, a.position
from publication as p, UNNEST(p.author)
     WITH ORDINALITY AS a (name, firstname, position)
```

c. All keywords together with the associated publications (as MULTISET).

```
select k.topic, COLLECT(p.title)
from publication p, UNNEST(p.keyword) AS k (topic)
group by k.topic
```

d. All publications with *XQuery* as a keyword.

```
select title
from publication
where 'XQuery' MEMBER OF keyword
```

e. All publications (title) with more than two keywords.

```
select title
from publication
where CARDINALITY(keyword) > 2
```

f. All publications (title) with duplicate keywords.

```
select title
from publication
where NOT keyword IS A SET
```

g. All distinct keywords used in 2008.

```
select DISTINCT k.topic
from publication p, UNNEST(p.keyword) AS k (topic)
where p.year = '2008'

select SET(FUSION(keyword))
from publication
where p.year = '2008'
```

h. All keywords and the number of times they have been used.

```
select k.topic, count(*) as number
from publication p, UNNEST(p.keyword) AS k (topic)
group by k.topic
```

i. Keywords used by all publications of *Jim Melton*.

```
select INTERSECTION(keyword)
from publication
where ROW('Melton', 'JIM') MEMBER CAST (author)
   AS ROW(name varchar(35), firstname varchar(25)) MULTISET
```

j. Pairs of publications (title) of the same authors.

```
select a.title, b.title
from publication a, publication b
where a.author = b.author
```

k. Authors with their publications (as ARRAY) ordered by publication date.

```
select a.name, a.firstname, ARRAY(
    select p2.title
    from publication p2
         UNNEST (p2.author) AS a2 (name, firstname)
    where a.name = a2.name AND a.firstname = a2.firstname
    order by p2.year desc) as publication
from publication p UNNEST (p.author) AS a (name, firstname)
```

l. Publications (title) that have no other keywords than *Understanding SQL and Java Together*

```
select p1.title
from publications p1, publications p2
where p2.title = 'Understanding SQL and Java Together' and
      p1.keyword SUBMULTISET p2.keyword
```

m. Keywords that have not been used before 2004.

```
select DISTINCT topic
from publications p UNNEST(p.keyword) AS k (topic)
where date >= 2004
EXCEPT
select DISTINCT topic
from publications p UNNEST(p.keyword) AS k (topic)
where date < 2004
```

n. All authors and the keywords they used (as MULTISET).

```
select a.name, a.firstname, COLLECT(k.topic) as keyword
from publication p, UNNEST(p.author) AS a (name, firstname),
     UNNEST(p.keyword) AS k (topic)
group by a.name, a.firstname
```

o. Insert a publication named "Understanding SQL and Java Together" by Jim Melton and Andrew Eisenberg published in 2000 with the keywords "SQLJ" and "JDBC".

```
insert into publication values
('Understanding SQL and Java Together', '2000',
 ARRAY[ROW('Melton', 'Jim'), ROW('Eisenberg', 'Andrew')],
 MULTISET['SQLJ', 'JDBC']);
```