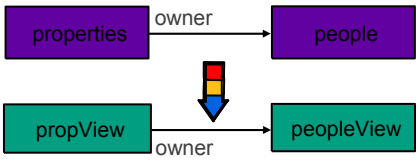
Chapter 2 – Object-Relational Views and Collection Types

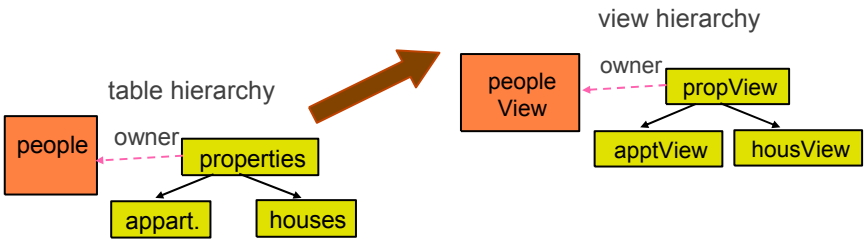
Prof. Dr.-Ing. Stefan Deßloch
AG Heterogene Informationssysteme
Geb. 36, Raum 329
Tel. 0631/205 3275
dessloch@informatik.uni-kl.de

Inhalt

Object Views

- Views have been extended to support
  - Typed views
  - View hierarchies
  - References on base tables can be mapped to references on views





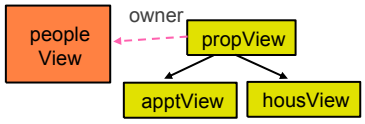
Object Views – Design Points

- Support the creation of a "closed" set of related object views that reference each other
- Mutually recursive references among object views
- Object ids (REF values in self-referencing columns) must be unique and long-lived (just like for typed tables)
- Structured types as the foundation for object views
  - same type can be used for typed tables, column types, object views
- Types used for defining object views don't have to related to type of underlying typed base tables
  - different attributes, behavior
- Object views are like "virtual typed tables"
  - associated type, self-referencing column, scoped references
  - view hierarchies

Object Views: Example

```
CREATE TYPE propViewType AS
      (owner          REF (person),
      location        address)
      REF USING integer NOT FINAL
CREATE TYPE apptViewType UNDER
      propVIewType ...
CREATE TYPE housViewType UNDER
      propViewType ...
CREATE VIEW propView OF propVIewType
      REF IS propID USER GENERATED
      (owner WITH OPTIONS SCOPE peopleView)
      AS (SELECT CAST (INTEGER(oid) AS
      REF(propViewType)), owner, location
      FROM ONLY (properties) )
CREATE VIEW housView OF housViewType
      UNDER propView
      AS (SELECT owner, location FROM ONLY
      (houses) )
CREATE VIEW apptView OF apptViewType
      UNDER propView
      AS (SELECT owner, location FROM ONLY
      (appartments) )
```

- Self-referencing column has to be defined for the root view
  - if USER GENERATED is used, then the view body has to include the oid column
  - only USER GENERATED and DERIVED are supported
- OIDs/references need to be cast to compatible ref types in the view body
- Values in self-referencing columns of view hierarchies need to be unique within the hierarchy
  - a view hierarchy can only be defined over a single table hierarchy
    - multiple hierarchies, multiple untyped base tables not supported
  - the FROM clause in the view body must reference a single table, and must specify ONLY for typed table reference
    - super/subviews must reference corresponding proper super/subtables



Enhanced Object View Support

- Limitations in SQL 1999 Object Views
  - restrictions in the view body
  - cannot define view hierarchies over one or more untyped base tables
- DB vendors have developed extensions to address these limitations
  - Oracle, IBM
- DB2 Object Views
  - less restrictions in view body
  - view hierarchies over single or multiple "legacy" tables
  - algorithm for static disjointness checking for subviews
    - guarantee uniqueness of oids in view hierarchies
  - UNCHECKED option for oid uniqueness
    - if multiple legacy tables are involved

M.Carey, S.Rielau, B.Vance: *Object View Hierarchies in DB2 UDB*, Proc. EDBT 2000

View Hierarchy Over a Single Legacy Table

- Example

```
CREATE VIEW vdept of Vdept_t
    (REF IS oid USER GENERATED)
    AS SELECT Vdept_t(dno), name,
    Vempt_t(mgrno) FROM dept
CREATE VIEW vperson of Vperson_t
    (REF IS oid USER GENERATED)
    AS SELECT Vperson_t(eno), name
    FROM emp
    WHERE salary IS NULL
CREATE VIEW vemp OF Vempt_t UNDER
    vperson
    (dept WITH OPTIONS SCOPE vdept)
    AS SELECT Vemp_t(eno), name,
    Vdept_t(deptno)
    FROM emp
    WHERE salary < 100000
ALTER VIEW vdept
    ALTER COLUMN mgr
    ADD SCOPE vemp
```

- Migration path for exploiting OR capabilities over legacy databases
- Self-referencing columns derived from primary keys of legacy table
- Foreign keys are converted into scoped references
- Disjointness check for subviews in a hierarchy
  - performed by analyzing the view predicates
  - done statically at view definition time
  - conservative algorithm
- UNCHECKED option
  - additional option for suppressing the disjointness check
  - can be used if multiple legacy tables are involved
  - uniqueness is now a user responsibility!

Collection Types: Arrays

- Array characteristics
  - Maximal length instead of actual length
    - like CHARACTER VARYING
    - has become optional in SQL 2003
  - Any element type admissible
  - "Arrays anywhere"
- Array operations
  - Element access by ordinal number
  - Cardinality
  - Comparison
  - Constructors
  - Assignment
  - Concatenation
  - CAST
  - Declarative selection facilities over arrays

Arrays (cont.)

- Tables with array-valued columns

```
CREATE TABLE reports
(id          INTEGER,
 authors     VARCHAR(15) ARRAY[20],
 title       VARCHAR(100),
 abstract    FullText)
```

- Appropriate DML operations

```
INSERT INTO reports(id, authors, title)
 VALUES (10, ARRAY ['Date', 'Darwen'], 'A Guide to the SQL Standard')
```


Access to array elements

- By ordinal position
- Declarative (i.e. query) facility
  - Implicitly transforms array into table
  - Selection by element content and/or position
  - Unnesting
- Examples:

```
SELECT id, authors[1] AS name FROM reports

SELECT r.id, a.name
FROM   reports AS r, UNNEST (r.authors) AS a (name)

SELECT r.id, a.name, a.position
FROM   reports AS r,
       UNNEST (r.authors) WITH ORDINALITY AS a (name, position)
```

Collection Types: MULTISET

- Complements the (unbound) ARRAY collection type
- Varying-length, unordered collections of element having specified type
- No (specified) maximum cardinality
- Usage examples:
  - COL1 INTEGER MULTISET
  - COL2 Address MULTISET
  - CREATE FUNCTION FOO (BAR CHAR(6))
    RETURNS CHAR(6) MULTISET
    ...

MULTISET Value Constructors

- By enumeration:
  - MULTISET[2, 3, 5, 7]
- Empty specification:
  - MULTISET[]
- By query:
  - MULTISET(SELECT COL1
    FROM TBL1
    WHERE COL2 > 10)
  - Result is the **multiset of resulting col1-values**, not the multiset of result rows
    - degree of the subquery must be 1
  - To obtain a multiset of rows, use the ROW constructor
    - MULTISET(SELECT **ROW(COL1, COL2)**
      FROM TBL1
      WHERE COL2 > 10)

MULTISET Operators

- Element reference (returns the only element in the multiset):
  - **ELEMENT**(MVE)
    - returns NULL iff
      - MVE is null
      - MVE has no elements
      - MVE has one element NULL
- Set function (converts a multiset into a set; i.e., duplicates are eliminated):
  - **SET**(MVE)
- Cardinality expression (returns the number of elements in the multiset):
  - **CARDINALITY**(MVE)
- UNION, EXCEPT, and INTERSECT:
  - *MVE1* MULTISET UNION [DISTINCT | ALL] *MVE2*
  - *MVE1* MULTISET EXCEPT [DISTINCT | ALL] *MVE2*
  - *MVE1* MULTISET INTERSECT [DISTINCT | ALL] *MVE2*
  - Similar to ordinary set operations, except ALL is the default

Using MULTISETs as Table References

- UNNEST operation:
  - **UNNEST**(MVE) AS correlation_name
- Example 1:
  - **UNNEST** MULTISET (2, 3, 5, 7) AS P

  produces the following table P:

  7
  5
  3
  2
- Example 2:
  - SELECT T.K, SUM (M.E)
    FROM T, **UNNEST** (T.M) AS M(E)
    GROUP BY T.K

MULTISET Predicates

- Comparison predicate (only equality and inequality)
  - Equal means
    - same number of elements
    - possible to match up the elements in pairs
- DISTINCT predicate
- MEMBER predicate
  - test for membership
- SUBMULTISET predicate
  - test whether multiset is a sub-multiset of another
- IS A SET predicate
  - test whether multiset contains anz duplicates

MULTISET Aggregates

- COLLECT
  - Transform the values in a group into a multiset.

```
SELECT Dept, COLLECT (Name)
   FROM PERS
   GROUP BY Dept
```

- FUSION
  - Form a union of the multisets in a group.
  - Number of duplicates of a given value in the result is the sum of the number of duplicates in the multisets in the rows of the group.
- INTERSECTION
  - Form an intersection of the multisets in a group.
  - Number of duplicates of a given value in the result is the minimum of the number of duplicates in the multisets in the rows of the group.