Chapter 4 – Application Programs and Object-Relational Capabilities

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

Inhalt

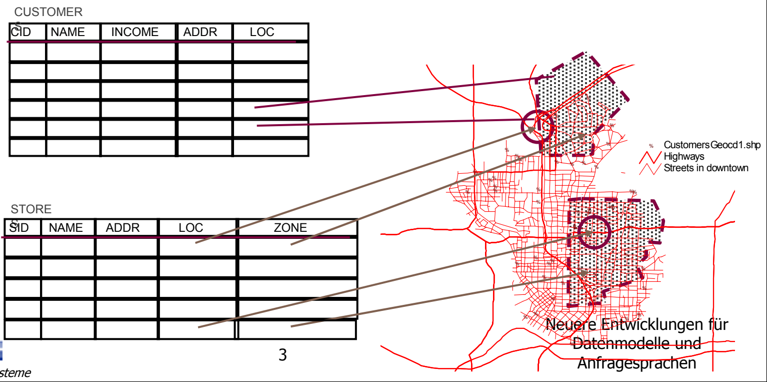
Structured Types and External Programs

- Instance of a structured type has to be made available in an external programming language environment

```
SELECT c.name, c.loc INTO :name :location
FROM  store s, customers c
WHERE  within(s.loc, :CA)=1 AND
    (within(c.loc, s.zone)=1 OR distance(c.loc, s.loc)<100)
```

client program

external routine



Transforms

- Transforms are user-defined functions or methods that get invoked automatically whenever UDT values are exchanged between SQL and external programs.
- Each UDT is associated with a collection of transform groups; each transform group is associated with:
  - A from_sql function that maps a UDT value into a value of predefined type.
  - A to_sql function that maps a value of a predefined type into a UDT value.

CREATE TRANSFORM

- CREATE TRANSFORM statement specifies a transform for a given UDT

```
CREATE TRANSFORM FOR point
    group1(     FROM SQL WITH FUNCTION from_point1(point),
                TO SQL WITH FUNCTION to_point1(char(27))
    group2(     FROM SQL WITH FUNCTION from_point2(point),
                TO SQL WITH FUNCTION to_point2(char(50));
```

- A transform group with a given name can be specified for only one type within a type hierarchy.
- An implicit transform is created for every distinct type on its creation, based on its cast functions.

Methods as Transform Functions

- Both from_sql and to_sql functions can be specified as methods:

```
CREATE TRANSFORM FOR point
    group1(     FROM SQL WITH METHOD from_point1() FOR point,
                TO SQL WITH METHOD to_point1(char(27) FOR point)
    group2(     FROM SQL WITH METHOD from_point2() FOR point,
                TO SQL WITH METHOD to_point2(char(50) FOR point);
```

- Both from_sql and to_sql methods can be overridden to define subtype-specific transform methods.
  - dynamic binding rules apply, i.e., if there is an overriding method available, that method is picked for execution.
- If there is no transform available for a UDT with a given group name, then a transform defined for one of its supertypes is picked.

Transforms in Embedded Programs

- An embedded program can specify transform groups for use during the execution of program:

```
TRANSFORM GROUP group1
TRANSFORM GROUP group2 FOR TYPE point
```

- A host variable whose data type is a UDT must specify a predefined type; must be same as the return type of from_sql function of the transform group specified for the UDT:

```
SQL TYPE IS point AS CHAR(50) pointvar
```

- from_sql function or method is automatically invoked on the UDT value and the result is passed to the host variable:

```
EXEC SQL SELECT center INTO :pointvar FROM circles WHERE ...
```

- to_sql function or method is automatically invoked on the host variable value and the result is passed to SQL:

```
EXEC SQL
   UPDATE circles
   SET center = :pointvar
   WHERE ...
```

Transforms in Dynamic SQL

- SET TRANSFORM GROUP statement sets the transform group for one or more UDTs for use during execution of dynamic SQL statements:

```
SET DEFAULT TRANSFORM GROUP group1;
SET TRANSFORM GROUP FOR TYPE point group2;
```

- Two special registers are provided to inquire about the session defaults:

```
CURRENT_DEFAULT_TRANSFORM_GROUP;
CURRENT_TRANSFORM_GROUP_FOR_TYPE point;
```

Transforms in External Routines

- An external routine can specify transform groups for use during the execution of routine:

```
CREATE FUNCTION foo(p1 point)
RETURNS INTEGER
EXTERNAL
TRANSFORM GROUP group1;
```

- The parameter in the external program corresponding to 'p1' must specify a host language type that corresponds to CHAR(27).
- Transform functions for UDT parameters are picked during the creation of external routines; once selected, the transform functions are frozen.
- Type-preserving functions/methods
  - If a to-sql **method** is defined, then a new instance of the most-specific type of the respective UDT parameter (e.g., SELF) is created, and the to-sql method is invoked on that instance

Dropping Transforms

- DROP TRANSFORM statement can be used to drop either a transform group or all transform groups attached to a UDT:

```
DROP TRANSFORM group1 FOR point RESTRICT;
DROP TRANSFORM ALL FOR point CASCADE;
```

- Dependencies between a transform group and the external routines that depend on that transform group are taken into account during dropping of transforms.

UDT, Array and Multiset Locators

- Similar to large object locators.
- A host variable can be specified as a locator variable for a UDT or an array/multiset type:

```
SQL TYPE IS point AS LOCATOR pointvar;
SQL TYPE IS INTEGER ARRAY[10] AS LOCATOR avar;
```

- An unique implementation-dependent 4-octet integer locator value is generated and passed to the host variable:

```
EXEC SQL
SELECT center INTO :pointvar
FROM circles WHERE ...
```

Using Locators in Assignment Statements

- When locators are used in assignment statements, the UDT or the array/multiset value corresponding to the given locator value is first found, and the result is then used in the assignment:

```
EXEC SQL
UPDATE circles
SET center = :pointvar
WHERE ...
```

Locators and External Routines

- A parameter of an external routine can be specified as locator parameter if its data type is either a UDT or an array or multiset type, or the returns type of an external function can specify AS LOCATOR if it is either a UDT or an array or multiset type:

```
CREATE FUNCTION foo(p1 emp AS LOCATOR)
RETURNS emp AS LOCATOR
EXTERNAL ...
```

- When the routine is invoked, an unique implementation-dependent 4-octet integer locator value is generated for each input locator parameter and passed as the argument value.
- After the routine finishes execution, for each output locator parameter or function result, the UDT or the array value corresponding to the locator value is first found, and the result is then returned to the caller.