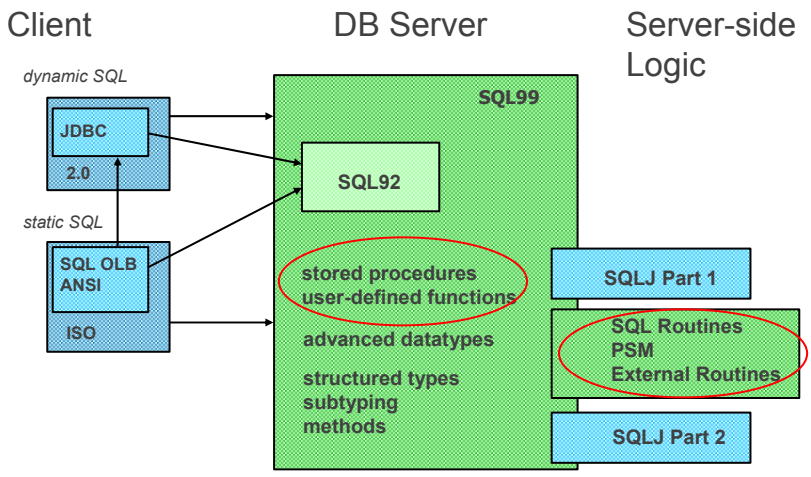
Chapter 3 – User-defined Routines and Object Behavior

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

Inhalt

The "Big Picture"



SQL-invoked Routines

- Named persistent code to be invoked from SQL
  - SQL-invoked **procedures**
  - SQL-invoked **functions**
    - SQL-invoked **methods**
- Created directly in a schema or in a SQL-server module
  - schema-level routines
  - module-level routines
- Have schema-qualified names
- Supported DDL
  - CREATE and DROP statements
  - ALTER statement -- still limited in functionality
  - EXECUTE privilege controlled through GRANT and REVOKE statements
- Described by corresponding information schema views

SQL-invoked Routines (cont.)

- Have a header and a body
  - Header consists of a name and a (possibly empty) list of parameters.
- Parameters of procedures may specify parameter mode
  - IN
  - OUT
  - INOUT
- Parameters of functions are always IN
- Functions return a single value
  - Header must specify data type of return value via RETURNS clause
- SQL routines
  - Both header and body specified in SQL
- External routines
  - Header specified in SQL
  - Bodies written in a host programming language
    - May contain SQL by embedding SQL statements in host language programs or using CLI

SQL Procedural Language Extensions

Compound statement	BEGIN ... END;
SQL variable declaration	DECLARE var CHAR (6);
If statement	IF subject (var <> 'urgent') THEN ... ELSE ...;
Case statement	CASE subject (var) WHEN 'SQL' THEN ... WHEN ...;
Loop statement	LOOP < SQL statement list> END LOOP;
While statement	WHILE i<100 DO .... END WHILE;
Repeat statement	REPEAT ... UNTIL i>=100 END REPEAT;
For statement	FOR result AS ... DO ... END FOR;
Leave statement	LEAVE ...;
Return statement	RETURN 'urgent';
Call statement	CALL procedure_x (1,3,5);
Assignment statement	SET x = 'abc';
Signal/resignal statement	SIGNAL division_by_zero

External Routines

- Parameters
  - Names are optional
  - Cannot be of any SQL data type
  - Permissible data types depend on the host language of the body
- LANGUAGE clause
  - Identifies the host language in which the body is written
- NAME clause
  - Identifies the host language code, e.g., file path in Unix
  - If unspecified, it corresponds to the routine name

```
CREATE PROCEDURE get_balance (IN acct_id INT, OUT bal DECIMAL(15,2))
LANGUAGE C
EXTERNAL NAME 'bank\balance_proc'

CREATE FUNCTION get_balance( IN INTEGER) RETURNS DECIMAL(15,2))
LANGUAGE C
EXTERNAL NAME 'usr/McKnight/banking/balance'
```


External Routines (cont.)

- RETURNS clause may specify **CAST FROM** clause

```
CREATE FUNCTION get_balance( IN INT)
RETURNS DECIMAL(15,2)) CAST FROM REAL
LANGUAGE C
```

  - C program returns a REAL value, which is then cast to DECIMAL(15,2) before returning to the caller.
- Special provisions to handle **null indicators** and the **status** of execution (SQLSTATE)
  - PARAMETER STYLE SQL (is the default)
  - PARAMETER STYLE GENERAL

PARAMETER STYLE SQL

- Additional parameters necessary for null indicators, returning function results, and returning SQLSTATE value
- External language program (i.e., the body) has 2n+4 parameters for procedures and 2n+6 parameters for functions where n is the number of parameters of the external routine

```
CREATE FUNCTION get_balance( IN INTEGER)
RETURNS DECIMAL(15,2)) CAST FROM REAL
LANGUAGE C
EXTERNAL NAME 'bank\balance'
PARAMETER STYLE SQL
```

```
void balance (int* acct_id,
float* rtn_val,
int* acct_id_ind,
int* rtn_ind,
char* sqlstate[6],
char* rtn_name [512],
char* spc_name [512],
char* msg_text[512])
{
...
}
```


PARAMETER STYLE GENERAL

- No additional parameters
- External language program (i.e., the body) must have exactly the same number of parameters
- Cannot handle null values
  - Exception is raised if any of the arguments evaluate to null
- Value returned in an implementation-dependent manner

```
CREATE FUNCTION get_balance( IN INTEGER)
RETURNS DECIMAL(15,2)) CAST FROM REAL
LANGUAGE C
EXTERNAL NAME  'bank\balance'
PARAMETER STYLE GENERAL
```

```
float* balance (int* acct_id)
{
...
}
```

Routine Characteristics

- DETERMINISTIC or NOT DETERMINISTIC
  - DETERMINISTIC (default)
    - Routine is expected to return the same result/output values for a given list of input values. (However, no checks are done at run time.)
  - NOT DETERMINISTIC routines not allowed in
    - Constraint definitions
    - Assertions
    - In the condition part of CASE expressions
    - CASE statements
- RETURNS NULL ON NULL INPUT or CALLED ON NULL INPUT (default)
  - RETURNS NULL ON NULL INPUT
    - An invocation returns null result/output value if any of the input values is null without executing the routine body
- DYNAMIC RESULT SETS <unsigned integer>
  - Valid on procedures only (SQL or external)
  - Defined number of result sets that the procedure is allowed to return
  - If unspecified, DYNAMIC RESULT SETS 0 is implicit

Routine Characteristics (cont.)

- CONTAINS SQL, READS SQL DATA, or MODIFIES SQL DATA
  - External routines may in addition specify NO SQL
  - Implementation-defined default
  - For SQL routines -- check may be done at routine creation time
  - For both SQL and external routines -- exception raised if a routine attempts to perform actions that violate the specified characteristic
  - Routines with MODIFIES SQL DATA not allowed in
    - Constraint definitions
    - Assertions
    - Query expressions other than table value constructors in an INSERT statement
    - Triggered actions of BEFORE triggers
    - Condition part of CASE expressions
    - CASE statements
    - searched delete statements
    - search condition of searched update statements (are allowed in SET clause)

Privilege Requirements

- SQL routine
  - Creator must have all the privileges required for execution of the routine body
  - Creator gets the EXECUTE privilege on the routine automatically
    - GRANT OPTION on EXECUTE privilege given if creator has GRANT OPTION on all the privileges required for execution of the routine body
    - Creator loses the GRANT OPTION if at any time he/she loses any of the privileges required for successful execution of the routine body
  - Routine is dropped if at any time the creator loses any of the privileges required for execution of the routine body (in CASCADE mode)
- External routine
  - Creator gets the EXECUTE privilege with GRANT OPTION on the routine automatically

Routine Overloading

- Overloading -- multiple routines with the same unqualified name

  S1.F (p1 INT, p2 REAL)

  S1.F (p1 REAL, p2 INT)

  S2.F (p1 INT, p2 REAL)
- Within the same schema
  - Every overloaded routine must have a unique signature, i.e., different number of parameters or different types for the same parameters

  S1.F (p1 INT, p2 REAL)

  S1.F (p1 REAL, p2 INT)
- Across schemas
  - Overloaded routines may have the same signature

  S1.F (p1 INT, p2 REAL)

  S2.F (p1 INT, p2 REAL)
- Functions can be overloaded by type. Procedures can only be overloaded based on number of parameters.

Routine Invocation

- Procedure -- invoked by a CALL statement:

  CALL get_balance (100, bal);
- Function -- invoked as part of an expression:

  SELECT account_id, get_balance (account_id)

  FROM accounts
- Requires the invoker to have EXECUTE privilege on the routine -- otherwise no routine will be found for the invocation

  It is not an authorization violation!!!

Subject Routine Determination

- Decides the function to invoke for a given invocation based on the
  - Compile-time data types of all arguments
  - Type precedence list of the data types of the arguments
  - SQL path
- Always succeeds in finding a unique subject function, if one exists.
- Type precedence list is a list of data type names
  - Predefined types -- defined by the standard based on increasing precision/length

    SMALLINT: SMALLINT, INTEGER, DECIMAL, NUMERIC, REAL, FLOAT, DOUBLE

    CHAR: CHAR, VARCHAR, CLOB
  - User-defined types -- determined by the subtype-supertype relationship
    - if B is a subtype of A and C is a subtype of B, then the type precedence list for C is (C, B, A).

Subject Routine Determination - Path

- Path is a list of schema names.
  - Can be specified during the creation of a schema, SQL-client module, or a SQL-server module

```
CREATE SCHEMA schema5
PATH schema1,schema3
...;
```

  - Every session has a default path, which can be changed using the SET statement.

```
SET PATH 'schema1, schema2'
```


Subject Routine Determination Algorithm

1. Determine the set of candidate functions for a given function invocation, F(a1, a2, ..., an):
   - Every function contained in S1 that has name F and has n parameters if the function name is fully qualified, i.e., the function invocation is of the form S1.F(a1, a2, ..., an), where S1 is a schema name.
   - Every function in every schema of the applicable path that has name F and has n parameters if the function name is not fully qualified.
2. Eliminate unsuitable candidate functions
   - The invoker has no EXECUTE privilege
   - The data type of i-th parameter of the function is not in the type precedence list of the static type of the i-th argument (for parameter)
3. Select the best match from the remaining functions
   - Examine the type of the 1st parameter of each function and keep only those functions such that the type of their 1st parameter matches best the static type of the 1st argument (i.e., occurs earliest in the type precedence list of the static type of the argument), and eliminate the rest.
   - Repeat Step b for the 2nd and subsequent parameters. Stop whenever there is only one function remaining or all parameters are considered.
4. Select the "subject function"
   - From the remaing functions take the one whose schema appears first in the applicable path (if there is only one function, then it is the "subject function")

Subject Routine Determination - Example

- Assume Y is a subtype of X. Assume the following three functions (with specific names F1, F2, and F3):
  - F1: F(p1 X, p2 Y)
  - F2: F (p1 Y, p2 Y)
  - F3: F(p1 X, p2 REAL)
- The subject function for F(y,y) where the static type of y is Y is F2.
- Now, assume the following three functions (with specific names F4, F5, and F6):
  - F4: F(p1 X, p2 Y)
  - F5: F(p1 X, p2 X)
  - F6: F(p1 X, p2 REAL)
- The subject function for F(y,y) where the static type of y is Y is F4.

Specific Names

- Uniquely identifies each routine in the database
  - If unspecified, an implementation-dependent name is generated.

```
CREATE FUNCTION get_balance( acct_id INTEGER)
RETURNS DECIMAL(15,2))
SPECIFIC func1
BEGIN
 ...
RETURN ...;
END
```

- Can only be used to identify the routine in ALTER, DROP, GRANT, and REVOKE statements

```
DROP SPECIFIC FUNCTION func1 RESTRICT;
```

- DDL statements can also identify a routine by providing the name and the list of parameter types

```
DROP FUNCTION get_balance(INTEGER) CASCADE;
```

- Cannot be used to invoke a routine

Altering Routines

- Routines can be altered with ALTER statement.
- Allowed only for external routines and for the following routine characteristics:
  - Language
  - Parameter style
  - SQL data access indication
  - Null behavior
  - Dynamic result set specification
  - NAME clause

```
ALTER FUNCTION get_balance (INTEGER)
READS SQL DATA
RESTRICT
```

- RESTRICT is the only allowed option, i.e., a routine cannot be altered if there are any dependent objects.

Dropping Routines

- Routines can be dropped using DROP statement.

```
DROP FUNCTION get_balance(INTEGER) CASCADE;
DROP FUNCTION get_balance(INTEGER) RESTRICT;
```

- Normal RESTRICT/CASCADE semantics applies with respect to dependent objects:
  - Routines
  - Views
  - Constraints
  - Triggers

TABLE Functions

- Motivation
  - Transform non-relational data into a relational table "on-the-fly" for further SQL processing
    - semi-structured, …
    - stored as BLOB, external file, …
    - provided by external service
      - search engine, web service, …
- Function returns a "multiset of rows".
- Function is used in the FROM clause of a query.
- Creating an **external** table function:

```
CREATE FUNCTION DOCMATCH(VARCHAR(30),VARCHAR(255))
    RETURNS TABLE(DOC_ID CHAR(16))
    LANGUAGE C
    NO SQL
    DETERMINISTIC
    EXTERNAL
    PARAMETER STYLE SQL
```


TABLE Functions (continued)

- Using table functions:

```
SELECT T.AUTHOR, T.DOCTEXT
FROM DOCS AS T,
    TABLE(DOCMATCH('MATHEMATICS', 'ZORN''S LEMMA')) AS F
WHERE T.DOCID = F.DOC_ID
```

TABLE Functions (continued)

- Creating an **SQL** table function:

```
CREATE FUNCTION DEPTEMPLOYEES (DEPTNO CHAR(3))
      RETURNS TABLE ( EMPNO CHAR(6),
                      LASTNAME VARCHAR(15),
                      FIRSTNAME VARCHAR(12))
      LANGUAGE SQL
      READS SQL DATA
      DETERMINISTIC
      RETURN TABLE(SELECT EMPNO, LNAME, FNAME
                   FROM EMPLOYEE
                   WHERE EMPLOYEE.WORKDEPT
                     = DEPTEMPLOYEES.DEPTNO)
```


Methods

- What are methods?
  - SQL-invoked functions "attached" to user-defined types
- How are they different from functions?
  - Implicit SELF parameter (called subject parameter)
  - Two-step creation process: signature and body specified separately.
  - Must be created in the type's schema
  - Different style of invocation, using dot-notation (e.g., *UDT-value.method*(...))

```
CREATE TYPE employee AS
(name                          CHAR(40),
base_salary      DECIMAL(9,2),
bonus                          DECIMAL(9,2))
INSTANTIABLE NOT FINAL
METHOD salary() RETURNS DECIMAL(9,2);

CREATE METHOD salary() FOR employee
BEGIN
....
END;
```

Methods (cont.)

- Three kinds of methods: instance, constructor, static methods
- Two types of instance methods:
  - Original methods: methods attached to (super) type
  - Overriding methods: methods attached to subtypes, redefining original behavior
    - Signature must match with the signature of an original method, except for the subject parameter.

```
CREATE TYPE employee AS
(name            CHAR(40),
base_salary      DECIMAL(9,2),
bonus                    DECIMAL(9,2))
INSTANTIABLE NOT FINAL
METHOD salary() RETURNS DECIMAL(9,2);

CREATE TYPE manager UNDER employee AS
(stock_option    INTEGER)
INSTANTIABLE NOT FINAL
OVERRIDING METHOD salary() RETURNS DECIMAL(9,2),      -- overriding
METHOD vested() RETURNS INTEGER                       -- original;
```


Instance Methods

- Invoked using dot syntax (assume dept table has mgr column):
  SELECT mgr.salary() FROM dept;
- Subject routine determination picks the "best" method to invoke.
  - Same algorithm as used for regular functions
  - SQL path is temporarily set to a list with the schemas of the supertypes of the static type of the self argument.
- Dynamic dispatch executed at runtime
  - Overriding methods considered at execution time
  - Overriding method with the best match for the dynamic type of the self argument is selected.
  - Schema evolution affects the actual method that gets invoked. If there is a new overriding method defined it may be picked for execution.

Method Reference

- References can be used to invoked methods on the corresponding structured type

```
SELECT prop.price, prop.owner->income (1998)
FROM properties.prop
```

- Invocation of methods given a reference value require select privilege on the method for the target typed table

```
GRANT SELECT (METHOD income FOR person) ON TABLE people TO PUBLIC
```

  - Allows the table owner to control who is authorized to invoked methods on the rows of his/her table

Static Methods

- Properties
  - have no subject (SELF) parameter
  - behavior associated with type, not instance
  - no overriding, dynamic dispatch
- Created using keyword STATIC

```
CREATE TYPE employee ...
STATIC METHOD totalSalary(base DECIMAL(9,2), bonus DECIMAL(9,2))
    RETURNS DECIMAL(9,2);
```

- Invocation uses structured type name, "::"
  - syntax 'borrowed' from C++

```
VALUES (employee::totalSalary(70000, 10000));
```

Initializing Instances: Constructor

- Instances are generated by the system-provided constructor function
  - Attributes are initialized with their default values
- Attributes are modified (further initialized) by invoking the mutator functions

```
BEGIN
   DECLARE re real_estate;
   SET re = real_estate();                  -- generation of a new instance
   SET re.rooms = 12;                       -- initialization of attribute rooms
   SET re.size = 2500;                      -- initialization of attribute size
END

BEGIN
   DECLARE re real_estate;
   SET re = real_estate().rooms (12).size (2500);  -- same as above
END
```


User-defined Constructor Methods

- Users can define any number of constructor methods and invoke them with NEW operator

```
CREATE TYPE real_estate AS ( ....)
CONSTRUCTOR METHOD real_estate (r INTEGER, s DECIMAL(8,2)) RETURNS real_estate

CREATE CONSTRUCTOR METHOD real_estate
      (r INTEGER, s DECIMAL(8,2)) RETURNS real_estate
BEGIN
   SET self.rooms = r;
   SET self.size = s;
   RETURN re;
END

BEGIN
   DECLARE re real_estate;
   SET re = NEW real_estate(12, 2500);                  -- same as previously
END
```

Methods That Modify Object State

- SQL functions operate on a value-based model
  - modification of object state as a side-effect of method is not possible
  - method needs to return a modified copy of the SELF object
    - return type must be the ST

```
CREATE TYPE employee AS
(...)
INSTANTIABLE NOT FINAL
METHOD salary (DECIMAL(9, 2)) RETURNS employee;

CREATE METHOD salary (newsal DECIMAL(9, 2)) FOR employee
BEGIN
    SET self.base_salary = newsal;
    RETURN self;
END;

UPDATE Employees            -- assumes the table has an emp column of type employee!
    SET emp = emp.salary(80000)
    WHERE emp.name = 'Smith';

UPDATE Employees
    SET emp.salary = 80000              -- same as above
    WHERE emp.name = 'Smith'
```


Type-Preserving Functions/Methods

- SQL-invoked function, one of whose parameters is a **result SQL parameter**.
  - The most specific type of the value returned by an invocation of a type-preserving function is identical to the most specific type of the SQL argument value substituted for the result SQL parameter
  - This can be the SELF parameter for methods
- Example:

```
CREATE TYPE employee AS
(...)
INSTANTIABLE NOT FINAL
METHOD salary (DECIMAL(9, 2)) RETURNS employee SELF AS RESULT;

UPDATE Managers         -- assumes the table has a mgr column of type manager!
    SET mgr = mgr.salary(80000)
    WHERE mgr.name = 'Smith';
```

Type-checking succeeds, although the return type of manager.salary() is **employee**!