

NDB - **Natural** User-defined Functions

A function is an operation denoted by a function name followed by zero or more operands that are enclosed in parentheses. A **function** represents a relationship between a set of input values and a set of result values. If a function **has been** implemented by a user-written program, DB2 **refers to it as a** user-defined function (UDF).

NDB supports the writing and executing of **Natural** user-defined functions (**Natural** UDFs). There are two types of **Natural (???)** UDF:

- scalar UDF
- table UDF

The scalar UDF accepts several input arguments and returns one output value. It can be invoked by any SQL statement like a DB2 built-in-function.

The table UDF accepts several input arguments and returns a set of output values comprising one table row **during** each invocation.

You invoke a table UDF with a SELECT statement by specifying the table-function name in the FROM clause. A table UDF performs as a DB2 table and is invoked for each **FETCH** operation for the table-function specified in the SELECT statement.

Below is information on:

- PARAMETER STYLE DB2SQL
- Writing **Natural UDFs**
- Security
- Example UDF NAT.DEM2UDFN

Related Topic: Natural Stored Procedures

PARAMETER STYLE DB2SQL

A Natural UDF receives (passes^{???}) parameters similar to **PARAMETER STYLE** DB2SQL for **Natural** stored procedures as described in the relevant section. The input parameters are **passed** in the sequence in which they are defined in the CREATE FUNCTION statements. **Following the input parameters**, the result parameter(s) are passed. **(The result parameter(s) are passed after the input parameters^{???})** For each parameter, a **NULL** indicator is passed. In addition, the following parameters are **passed**:

- The SQLSTATE to be returned to DB2,
- The qualified name of the **Natural (???)** UDF,
- The specific name of the **Natural (???)** UDF,
- The SQL diagnostic string to be returned to DB2.

The parameters are defined in the Natural parameter data area (PDA) DB2SQL_P supplied in the Natural system library SYSDB2.

If the feature SCRATCHPAD *nmn* is specified additionally in the CREATE FUNCTION statement, **the SCRATCHPAD storage parameter is passed to the Natural (???) UDF.**

Use the following definitions:

```

01 scratchpad A(4+nnn)
01 redefines scratchpad
02 scratchpad_length (I4)
02

```

Redefine the SCRATCHPAD in the Natural (???) UDF according to your requirements.

The first four bytes of the **scratchpad (gross???)** area contain a length (???) field. Before initially invoking the UDF with an SQL statement, DB2 resets the **scratchpad (gross???)** area to x'00' and puts into the first four bytes the specified size *nnn* of the SCRATCHPAD as integer value.

Thereafter, DB2 does not reinitialize the SCRATCHPAD between the invocations of the UDF **for the invoking SQL statement.???** Instead, after returning from the UDF, the contents of the SCRATCHPAD is **preserved** and restored at the next invocation of the UDF.

If the FINAL CALL option is specified for a scalar UDF, or if the UDF is a table UDF, the CALL TYPE parameter is passed. The CALL TYPE parameter is an integer indicating the type of call DB2 performs on the UDF. See the DB2 SQL GUIDE for details on the **parameter (???)** values provided.

If the option DBINFO is used additionally, the DBINFO data (???) structure is passed to the **Natural (???)** stored procedure. The DBINFO structure is described by the Natural PDA DBINFO_P that is supplied in the Natural system library SYSDB2.

Defining UDF Parameters

Define the UDF parameters in Natural as shown in the example program below:

```

DEFINE DATA PARAMETER
01 PI1    /* first input parameter
01 PI2

01 PIn ... /* last input parameter
01 RS1 /* first result parameter

01 RSn /* last result parameter
01 N_PI1 (I2) /* first NULL indicator
01 N_PI2 (I2)

01 N_PIn (I2)
01 N_RS1 (I2)

01 N_RSn (I2) /* last NULL indicator
PARAMETER USING DB2SQL_P /* function, specific, sqlstate, diagnose
PARAMETER
01 SCRATCHPAD A(4+nnn) /* only if SCRATCHPAD nnn is specified
    01 REDEFINES SCRATCHPAD
02 SCRATCHPAD_LENGTH (I4)
02
01 CALL_TYPE (I4) /* --- only if FINAL CALL specified or table UDF

PARAMETER USING DBINFO_P /* ---- only if DBINFO is specified
LOCAL

END-DEFINE

```

The **NDB server stub** that executes the **Natural UDF** determines the name of the subprogram and the library to be invoked from the qualified and specific names of the UDF. The SCHEMA name is used as library name and the **function** name is used as subprogram name.

The subprogram **ROUTINEN** (supplied in the Natural system library SYSDDB2) is used to access the DB2 catalog **in order to** determine the format of the user parameters of the **Natural UDF**. After the formats have been determined, they are **stored** in the Natural buffer pool. During subsequent invocations of the **Natural UDF**, the formats are retrieved from the Natural buffer pool. This requires that at least READS SQL DATA is specified for a **Natural UDF**.

If a Natural runtime error occurs during the execution of a Natural **UDF** with **PARAMETER STYLE DB2SQL**, SQLSTATE is set to 38N99 and a diagnostic string contains the text of the Natural error message.

If an error occurs in the **NDB server stub** during the execution of a **Natural UDF** using **PARAMETER STYLE DB2SQL**, the SQLSTATE is set to 38S99 and a diagnostic string **contains** the text of the error message.

If the application wants to raise an error condition during the execution of a Natural UDF, the SQLSTATE parameter must be set to a value other than 00000. See the DB2 SQL Guide for specifications of user errors in the SQLSTATE parameter.

Additionally, a text for analysing errors can be placed in the DIAGNOSE parameter.

If a table **(???) Natural UDF** wants to signal to DB2 that it has found no row to return, '02000' (Kommata erforderlich???) should be returned in the SQLSTATE parameter.

For a **Natural UDF** that contains the attributes DISALLOW PARALLEL and FINAL CALL, the **NDB server stub retains** the Natural session allocated earlier. This Natural session will then be reused by all subsequent UDF invocations until Natural encounters the final call.

Writing a **Natural UDF**

Below is a general guideline of how to write a **Natural UDF**.

To write a **Natural UDF**

1. - 8. See Steps 1 to 8 in Writing Natural Stored Procedures.
9. Determine the format and attributes of the parameters, which are passed between the caller and the stored procedure.
10. Create a **Natural parameter data area (PDA)**.
11. Append the **parameter definitions from the (???) PDA to the Natural UDF** by defining NULL indicators (one for each parameter) and include the **PDA DB2SQL_P**.
12. If **required**, code a **scratchpad (gross???)** area in the parameter list
13. If **required**, code a call-type parameter.
If you have specified DBINFO, include the **PDA DBINFO_P**. See also the relevant DB2 literature by IBM.
14. Code your UDF as a Natural subprogram **and consider the following:**

Attention when accessing other databases

You can access other databases (for example, Adabas) within a **Natural stored procedure**. You should keep in mind that your access to other databases is synchronized neither with the updates done by the caller of the stored procedure, nor with the updates done against DB2 within the stored procedure.

NDB handling of COMMIT and ROLLBACK statements

DB2 does not allow a stored procedure to issue COMMIT or ROLLBACK statements; the execution of these statements **would put (force the caller into a must-rollback situation???)** the caller in a must-rollback **situation ???**. If a Natural stored procedure issues a COMMIT or ROLLBACK, the NDB runtime processes these statements as follows:

COMMIT against DB2 is skipped.

This allows the stored procedure to commit Adabas updates without getting a must-rollback **condition (state passt nicht!!!???)** by DB2.

ROLLBACK against DB2 is skipped if it comes from a Natural source.

ROLLBACK against DB2 is executed if it is user-programmed.

Thus, after a Natural error, the caller receives a corresponding Natural error message text but not an unqualified must-rollback **condition ???**. Additionally, this **behavior** ensures that every database transaction the stored procedure performs is backed out if the user program backs out the transaction.

15. Issue a CREATE FUNCTION statement that defines your UDF, for example:

```
CREATE FUNCTION <FUNCTION>
  ( [ PARM1 ]      <FORMAT> ,
    [ PARM2 ]      <FORMAT> ,
    [ PARM3 ]      <FORMAT>

  )
  RETURNS <FORMAT>

  EXTERNAL NAME <LOADMOD>
  LANGUAGE ASSEMBLE
  PROGRAM TYPE <PGM TYPE>
  PARAMETER STYLE DB2SQL
.
.
. ;
```

In the example above, the variable data are enclosed in angle brackets (< >) and refer to the keywords preceding the brackets. Specify a valid value, for example:

LOADMOD denotes the **Natural (???)** stored procedure/UDF server stub module, for example, NDB41SRV. PARM1 - PARM3 and FORMAT relate to the call parameter list of the UDF. See also Example UDF NAT.DEM2UDFN below.

16. Code a **Natural** program containing SQL **statements** that invoke the UDF.

Specify the parameters in **(for???)** the **Natural** UDF invocation **(???)** in the same sequence as specified in the **Natural** UDF definition. The format of the parameters in the calling program must be compatible with the format defined in the **Natural** UDF.

Security

DB2 provides an authorization ID **for executing a Natural UDF and controlling** access to NON-SQL resources with an external security product like RACF. **The NDB server stub uses this authorization ID to perform an implicit LOGON within the Natural session created for the Natural UDF.** So, if the **Natural** UDF is to be executed in a Natural Security environment, ensure that the authorization ID used has been defined in the FSEC file.

The authorization ID **is the authorization ID of**

- The address space in which the **Natural** UDF is running **if SECURITY DB2 has been specified**, or
- The process that invokes the **Natural** UDF, **if SECURITY USER has been specified**, or
- The owner of the procedure **if SECURITY DEFINER has been specified.**

Example UDF NAT.DEM2UDFN

This section describes the example UDF NAT.DEM2UDFN, a Natural subprogram **used to calculate the product of two numbers (???)**.

The example UDF NAT.DEM2UDF comprises the following members that are supplied in the Natural system library SYSDB2:

Member	Explanation
DEM2CUDF	Contains SQL statements used to create DEM2UDFN (see below).
DEM2UDFP	The client (Natural) program that <ul style="list-style-type: none">● Fetches rows from the NAT.DEMO table,● Invokes the UDF NAT.DEM2UDFN (see below) in the WHERE clause, and● Displays the rows fetched.
DEM2UDFN	The UDF that builds the product of two numbers. DEM2UDFN has to be copied to the Natural library NAT on the FUSER in the executing environment.