Dynamic and Static SQL Support

This section describes the dynamic and static SQL support provided by Natural.

The following topics are covered:

- SQL Support - General Information
- Internal Handling of Dynamic Statements
- Preparing Programs for Static Execution
- Execution of Natural in Static Mode
- Mixed Dynamic/Static Mode
- Messages and Codes

Related Documentation

For a list of error messages that may be issued during static generation, see *Static Generation Messages and Codes Issued under NDB/NSQ* in the Natural *Messages and Codes* documentation.

SQL Support - General Information

The SQL support of Natural combines the flexibility of dynamic SQL support with the high performance of static SQL support.

In contrast to static SQL support, the Natural dynamic SQL support does not require any special consideration with regard to the operation of the SQL interface. All SQL statements required to execute an application request are generated automatically and can be executed immediately with the Natural `RUN` command. Before executing a program, you can look at the generated SQL code, using the `LISTSQL` command.

Access to SQL/DS through Natural has the same form whether dynamic or static SQL support is used. Thus, with static SQL support, the same SQL statements in a Natural program can be executed in either dynamic or static mode. An SQL statement can be coded within a Natural program and, for testing purposes, it can be executed using dynamic SQL. If the test is successful, the SQL statement remains unchanged and static SQL for this program can be generated.

Thus, during application development, the programmer works in dynamic mode and all SQL statements are executed dynamically, whereas static SQL is only created for applications that have been transferred to production status.

Internal Handling of Dynamic Statements

Natural automatically provides for the preparation and execution of each SQL statement and handles the opening and closing of cursors used for scanning a table.

The following topics are covered:

- I/O Module NDBIOMO for Dynamic SQL Statement Execution
- Statement Table
- Processing of SQL Statements Issued by Natural

I/O Module NDBIOMO for Dynamic SQL Statement Execution

As each dynamic execution of an SQL statement requires a statically defined `DECLARE STATEMENT` and `DECLARE CURSOR` statement, a special I/O module named `NDBIOMO` is provided which contains a fixed number of these statements and cursors. This number is specified during the generation of the `NDBIOMO` module in the course of the Natural for DB2 installation process.

Statement Table

If possible, an SQL statement is only prepared once and can then be executed several times if required. For this purpose, Natural internally maintains a table of all SQL statements that have been prepared and assigns each of these statements to a `DECLAREd STATEMENT` in the module `NDBIOMO`. In addition, this table maintains the cursors used by the SQL statements `SELECT`, `FETCH`, `UPDATE` (positioned), and `DELETE` (positioned).

Each SQL statement is uniquely identified by:

- the name of the Natural program that contains this SQL statement,
- the line number of the SQL statement in this program,
- the name of the Natural library into which this program was stowed,
- the time stamp when this program was stowed.

Once a statement has been prepared, it can be executed several times with different variable values, using the dynamic SQL statement `EXECUTE USING DESCRIPTOR` or `OPEN CURSOR USING DESCRIPTOR`.

When the full capacity of the statement table is reached, the entry for the next prepared statement overwrites the entry for a free statement whose latest execution is the least recent one.

When a new `SELECT` statement is requested, a free entry in the statement table with the corresponding cursor is assigned to it and all subsequent `FETCH`, `UPDATE`, and `DELETE` statements referring to this `SELECT` statement will use this cursor. Upon completion of the sequential scanning of the table, the cursor is released and free for another assignment. While the cursor is open, the entry in the statement table is marked as used and cannot be reused by another statement.

If the number of nested `FIND` (`SELECT`) statements reaches the number of entries available in the statement table, any further SQL statement is rejected at execution time and a Natural error message is returned.

The size of the statement table depends on the size specified for the module `NDBIOMO`. Since the statement table is contained in the SQL/DS buffer area, the setting of Natural profile parameter `DB2SIZE` (see also *Natural Parameter Modification for SQL/DS*) may not be sufficient and may need to be increased.

Processing of SQL Statements Issued by Natural

The embedded SQL uses cursor logic to handle `SELECT` statements. The preparation and execution of a `SELECT` statement is done as follows:

1. The typical `SELECT` statement is prepared by a program flow which contains the following embedded SQL statements (note that *`X`* and *`SQLOBJ`* are SQL variables, not program labels):

   ```
   DECLARE SQLOBJ STATEMENT
   DECLARE X CURSOR FOR SQLOBJ
   INCLUDE SQLDA (copy SQL control block)
   ```

   Then, the following statement is moved into `SQLSOURCE`:

   ```
   SELECT PERSONNEL_ID, NAME, AGE
    FROM EMPLOYEES
    WHERE NAME IN (?, ?)
       AND AGE BETWEEN ? AND ?
   ```

   Note:
   The question marks (?) above are parameter markers which indicate where values are to be inserted at execution time.

```
PREPARE SQLOBJ FROM SQLSOURCE
```

2. Then, the SELECT statement is executed as follows:

   ```
   OPEN X USING DESCRIPTOR SQLDA
   FETCH X USING DESCRIPTOR SQLDA
   ```

   The descriptor SQLDA is used to indicate a variable list of program areas. When the OPEN statement is executed, it contains the address, length, and type of each value which replaces a parameter marker in the WHERE clause of the SELECT statement. When the FETCH statement is executed, it contains the address, length, and type of all program areas which receive fields read from the table.

   When the FETCH statement is executed for the first time, it sets the Natural system variable *NUMBER to a non-zero value if at least one record is found that meets the search criteria. Then, all records satisfying the search criteria are read by repeated execution of the FETCH statement.

3. Once all records have been read, the cursor is released by executing the following statement:

   ```
   CLOSE X
   ```

Preparing Programs for Static Execution

This section describes how to prepare Natural programs for static execution.

The following topics are covered:

- Basic Principles
- Generation Procedure: CMD CREATE Command
- Modification Procedure: CMD MODIFY Command

For an explanation of the symbols used in this section to describe the syntax of Natural statements, see *Syntax Symbols* in the Natural *Statements* documentation.

Basic Principles

Static SQL is generated in Natural batch mode for one or more Natural applications which can consist of one or more Natural object programs. The number of programs that can be modified for static execution in one run of the generation procedure is limited to 999.

During the generation procedure, the database access statements contained in the specified Natural objects are extracted, written to work files, and transformed into a temporary Assembler program. If no Natural program is found that contains SQL access or if any error occurs during static SQL generation, batch Natural terminates and condition code 40 is returned, which means that all further JCL steps must no longer be executed.

The temporary Assembler program is written to a temporary file (the Natural work file CMWKF06) and precompiled. The size of the workfile is proportional to the maximum number of programs, the number of SQL statements and the number of variables used in the SQL statements. During the precompilation step, a static SQL/DS module (access module) is created, and after the precompilation step, the precompiler output is extracted from the Assembler program and written to the corresponding Natural objects, which means that the Natural objects are modified (prepared) for static execution. The temporary Assembler

program is no longer used and deleted.

Note:
Since the Assembler precompiler of SQL/DS does not support `GRAPHIC` field types, you cannot generate a static Assembler program if your Natural program(s) contain any references to `GRAPHIC`-type columns.

The Natural subprogram `NDBDBRM` can be used to check whether a Natural program contains an SQL access and whether it has been modified for static execution.

Generation Procedure: CMD CREATE Command

The following topics are covered:

- Generating Static SQL for Natural Programs
- Static Name
- USING-Clause

Generating Static SQL for Natural Programs

To generate static SQL for Natural programs

1. Logon to the Natural system library `SYSSQL`.

   Since a new `SYSSQL` library has been created when installing Natural for SQL/DS, ensure that it contains all Predict interface programs necessary to run the static SQL generation. These programs are loaded into `SYSSQL` at Predict installation time (see the relevant *Predict* product documentation).

2. Specify the `CMD CREATE` command and the Natural input necessary for the static SQL generation process; the `CMD CREATE` command has the following syntax:

   ```
   CMD CREATE DBRM static-name  USING using-clause
   {application-name,object-name,excluded-object}
   :
   :
   ```

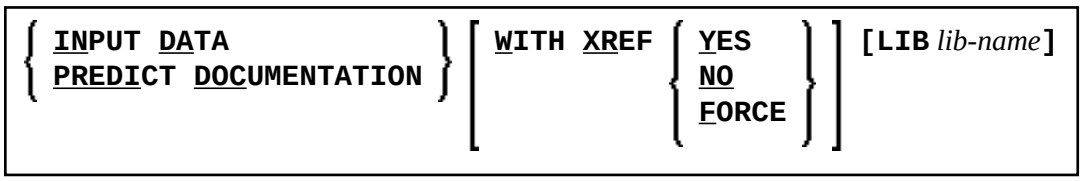
   The generation procedure reads but does not modify the specified Natural objects. If one of the specified programs was not found or had no SQL access, return code 4 is returned at the end of the generation step.

Static Name

If the `PREDICT DOCUMENTATION` option is to be used, a corresponding Predict static SQL entry must be available and the *`static-name`* must correspond to the name of this entry. In addition, the *`static-name`* must correspond to the name of the static SQL/DS package to be created during precompilation. The *`static-name`* can be up to 8 characters long and must conform to Assembler naming conventions.

USING-Clause

The *using-clause* specifies the Natural objects to be contained in the the static SQL/DS package. These objects can either be specified explicitly as `INPUT DATA` in the JCL or obtained as `PREDICT DOCUMENTATION` from Predict.



If the parameters to be specified do not fit in one line, specify the command identifier (`CMD`) and the various parameters in separate lines and use both the input delimiter (as specified with the Natural profile/session parameter `ID` - default is a comma (,) - and the continuation character indicator - as specified with the Natural profile/session parameter `CF`; default is a percent (%) - as shown in the following example:

Example:

```
CMD
CREATE,DBRM,static,USING,PREDICT,DOCUMENTATION,WITH,XREF,NO,%
LIB,library
```

Alternatively, you can also use abbreviations as shown in the following example:

Example:

```
CMD CRE DBRM static US IN DA W XR Y LIB library
```

The sequence of the parameters `USING`, `WITH`, and `LIB` is optional.

INPUT DATA

As input data, the applications and names of the Natural objects to be included in the static SQL/DS package must be specified in the subsequent lines of the job stream (*application-name,object-name*). A subset of these objects can also be excluded again (*excluded-objects*). Objects in libraries whose names begin with `SYS` can be used for static generation, too.

The applications and names of Natural objects must be separated by the input delimiter - as specified with the Natural profile parameter `ID`; default is a comma (,). If you wish to specify all objects whose names begin with a specific string of characters, use an *object-name* or *excluded-objects* name that ends with asterisk notation (*). To specify all objects in an application, use asterisk notation only.

Example:

```
LIB1,ABC*
LIB2,A*,AB*
LIB2,*
:
.
```

The specification of applications/objects must be terminated by a line that contains a period (.) only.

PREDICT DOCUMENTATION

Since Predict supports static SQL for SQL/DS, you can also have Predict supply the input data for creating static SQL by using already existing PREDICT DOCUMENTATION.

WITH XREF Option

Since Predict Active References supports static SQL for SQL/DS, the generated static SQL/DS package can be documented in Predict, and the documentation can be used and updated with Natural.

WITH XREF is the option which enables you to store cross-reference data for a static SQL entry in Predict each time a static SQL/DS package is created (YES). You can instead specify that no cross-reference data are stored (NO) or that a check is made to determine whether a Predict static SQL entry for this static DBRM already exists (FORCE). If so, cross-reference data are stored; if not, the creation of the static DBRM is not allowed. For more detailed information on Predict Active References, refer to the relevant Predict documentation.

When WITH XREF (YES/FORCE) is specified, XREF data are written for both the Predict static SQL entry (if defined in Predict) and each generated static Natural program. However, static generation with WITH XREF (YES/FORCE) is possible only if the corresponding Natural programs have been cataloged with XREF ON.

WITH XREF FORCE only applies to the USING INPUT DATA option.

Note:
If you do not use Predict, the XREF option must be omitted or set to NO and the module NATXRF2 need not be linked to the Natural nucleus.

LIB Option

With the LIB (library) option, a Predict library other than the default library (*SYSSTA*) can be specified to contain the Predict static SQL entry and XREF data. The name of the library can be up to eight characters long.

Modification Procedure: CMD MODIFY Command

The modification procedure modifies the Natural objects involved by writing precompiler information into the object and by marking the object header with the *static-name* as specified with the CMD CREATE command.

In addition, any existing copies of these objects in the Natural global buffer pool (if available) are deleted and XREF data are written to Predict (if specified during the generation procedure).

To perform the modification procedure

1. Logon to the Natural system library SYSSQL.
2. Specify the CMD MODIFY command which has the following syntax:

```
CMD MODIFY [ XREF ]
```

The input for the modify step is the precompiler output which must reside on a dataset defined as the Natural work file CMWKF01.

The output consists of precompiler information which is written to the corresponding Natural objects. In addition, a message is returned telling you whether it was the first time an object was modified for static execution (modified) or whether it had been modified before (re-modified).

If the XREF option is specified, the Natural work file CMWKF02 must be defined to contain the resulting list of cross-reference information concerning the statically generated SQL statements (see also *Assembler/Natural Cross-References*).

Assembler/Natural Cross-References

If you specify the XREF option of the MODIFY command, an output listing is created on the work file CMWKF02, which contains the static SQL/DS package name and the Assembler statement number of each statically generated SQL statement together with the corresponding Natural source code line number, program name, library name, database ID and file number.

Example:

```
------------------------------------------------------------------------
 DBRMNAME STMTNO      LINE NATPROG  NATLIB   DB    FNR   COMMENT
------------------------------------------------------------------------
 DEM2S    000087      0170 DEM2SUPD HGK      00010 00032 SELECT
          000111      0230                               UPD/DEL
 DEM2S    000121      0370 DEM2SINS HGK      00010 00032 INSERT
 DEM2S    000131      0150 DEM2SDEL HGK      00010 00032 SELECT
          000155      0170                               UPD/DEL
 DEM2S    000165      0040 DEM2SDL2 HGK      00010 00032 UPD/DEL
```

Column	Explanation
DBRMNAME	Name of the static SQL/DS package which contains the static SQL statement.
STMTNO	Assembler statement number of the static SQL statement.
LINE	Corresponding Natural source code line number.
NATPROG	Name of the Natural program that contains the static SQL statement.
NATLIB	Name of the Natural library that contains the Natural program.
DB / FNR	Natural database ID and file number.
COMMENT	Type of SQL statement.

Execution of Natural in Static Mode

To be able to execute Natural in static mode, all users of Natural must have the SQL/DS EXECUTE PLAN/PACKAGE privilege for the plan created in the precompilation step.

To execute static SQL, start Natural and execute the corresponding Natural program. Internally, the Natural runtime interface evaluates the precompiler data written to the Natural object and then performs the static accesses.

To the user there is no difference between dynamic and static execution.

Mixed Dynamic/Static Mode

It is possible to operate Natural in a mixed static and dynamic mode where for some programs static SQL is generated and for some not.

The mode in which a program is run is determined by the Natural object program itself. If a static SQL/DS package is referenced in the executing program, all statements in this program are executed in static mode.

Note:
Natural programs which return a runtime error do not automatically execute in dynamic mode. Instead, either the error must be corrected or, as a temporary solution, the Natural program must be recataloged to be able to execute in dynamic mode.

Within the same Natural session, static and dynamic programs can be mixed without any further specifications. The decision which mode to use is made by each individual Natural program.

Messages and Codes

For a list of error messages that may be issued during static generation, refer to *Static Generation Messages and Codes Issued under NDB/NSQ* in the Natural *Messages and Codes* documentation.