Dynamic SQL Support

This section describes the dynamic SQL support provided by Natural SQL Gateway. Natural SQL Gateway does not support static SQL.

- SQL Support - General Information
- Internal Handling of Dynamic Statements
- NSB - Statements and System Variables
- NSB - Natural DML Statements
- Natural SQL Statements

SQL Support - General Information

The SQL support of Natural SQL Gatewayprovides the flexibility of dynamic 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.

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.

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. 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` respectively.

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.

Since the statement table is contained in the SQL buffer area, the `DB2SIZE` parameter (see *Natural Parameter Modification for Natural SQL Gateway* in *Installing Natural SQL Gateway*) may not be sufficient and may need to be increased.

NSB - Statements and System Variables

This section contains special considerations concerning Natural DML statements, Natural SQL statements, and Natural system variables when used with SQL.

It mainly consists of information also contained in the Natural documentation set where each Natural statement and variable is described in detail.

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.

This section covers the following topics:

- NSB Special Register Consideration

NSB Special Register Consideration

Natural SQL Gateway supports the following special registers, which can be set via the `PROCESS SQL` statement:

- `SCHEMA`

The `SCHEMA` special register determines the implicitly first level qualifier of table names, that is, the schema or creator name of the table, if the first qualifier is not explicitly specified. The `SCHEMA` special register could be set by `PROCESS SQL` *`ddm-name`* `<< SET SCHEMA = :`*`hv`*`>>`, where *ddm-name* denotes the DDM whose DBID is mapped to type CNX and `:`*`hv`* denotes an alphanumeric variable containing the first level qualifier.

The `SCHEMA` special register cannot be retrieved or interrogated by SQL statements.

- `CATALOG`

The `CATALOG` special register determines the implicitly second level qualifer of table names, that is, the location or database name of the table, if the second level qualifier is not explicitly specified. The `CATALOG` special register could be set by `PROCESS SQL` *`ddm-name`* `<< SET CATALOG = :`*`hv`*`>>`, where *`ddm-name`* denotes DDM whose DBID is mapped to type `CNX` and `:`*`hv`* denotes a alphanumeric variable containing the second level qualifier.

The `CATALOG` special register could not be retrieved or interrogated by SQL statements.

- `RCI_VERSION`

The `RCI_VERSION` is an alphanumeric character string containing the version of the remote client interface used to communicate with the CONNX JDBC server. The `RCI_VERSION` is a read-only special register which could be retrieved by `PROCESS SQL` *`ddm-name`* `<<GET :`*`hv`*` = RCI_VERSION>>`, where *`ddm-name`* denotes a DDM whose DBID is mapped to type `CNX` and :*hv* denotes a alphanumeric variable. The `RCI_VERSION` string has the following format:

```
RCI: 4.1.1 CONNX 10.5 SP2 (build 7294)
```

NSB - Natural DML Statements

This section summarizes particular points you have to consider when using Natural DML statements with SQL. Any Natural statement *not* mentioned in this section can be used with SQL without restriction.

- BACKOUT TRANSACTION
- DELETE
- END TRANSACTION
- FIND
- GET

- HISTOGRAM
- READ
- STORE
- UPDATE

BACKOUT TRANSACTION

This statement undoes all database modifications made since the beginning of the last logical transaction. Logical transactions can start either after the beginning of a session or after the last `SYNCPOINT`, `END TRANSACTION`, or `BACKOUT TRANSACTION` statement.

How the statement is translated and which command is actually issued depends on the TP-monitor environment:

- In batch mode and under TSO, the `BACKOUT TRANSACTION` statement is translated into an SQL `ROLLBACK` command.

As all cursors are closed when a logical unit of work ends, a `BACKOUT TRANSACTION` statement must not be placed within a database loop; instead, it has to be placed outside such a loop or after the outermost loop of nested loops.

If an external program written in another standard programming language is called from a Natural program, this external program must not contain its own `ROLLBACK` command if the Natural program issues database calls, too. The calling Natural program must issue the `BACKOUT TRANSACTION` statement for the external program.

If a program tries to backout updates which have already been committed, for example by a terminal I/O, a corresponding Natural error message (NAT3711) is returned.

DELETE

The `DELETE` statement is used to delete a row from an SQL table which has been read with a preceding `FIND`, `READ`, or `SELECT` statement. It corresponds to the SQL statement `DELETE WHERE CURRENT OF` *`cursor-name`*, which means that only the row which was read last can be deleted.

Example:

```
FIND EMPLOYEES WITH NAME = 'SMITH'
    AND FIRST_NAME = 'ROGER'
DELETE
```

Natural would translate the above Natural statements into SQL and assign a cursor name (for example, `CURSOR1`) as follows:

```
DECLARE CURSOR1 CURSOR FOR
SELECT FROM EMPLOYEES
 WHERE NAME = 'SMITH' AND FIRST_NAME = 'ROGER'
DELETE FROM EMPLOYEES
 WHERE CURRENT OF CURSOR1
```

Both the SELECT and the DELETE statement refer to the same cursor.

Natural translates a DML DELETE statement into an SQL DELETE statement in the same way it translates a FIND statement into an SQL SELECT statement.

A row read with a FIND SORTED BY cannot be deleted due to SQL restrictions explained with the FIND statement. A row read with a READ LOGICAL cannot be deleted either.

DELETE when using the File Server

If a row rolled out to the file server is to be deleted, Natural rereads automatically the original row from the database to compare it with its image stored in the file server. If the original row has not been modified in the meantime, the DELETE operation is performed. With the next terminal I/O, the transaction is terminated, and the row is deleted from the actual database.

If the DELETE operates on a scrollable cursor, the row on the file server is marked as DELETE hole and is deleted from the base table.

However, if any modification is detected, the row will not be deleted and Natural issues the NAT3703 error message for non-scrollable cursors.

Since a DELETE statement requires that Natural rereads a single row, a unique index must be available for the respective table. All columns which comprise the unique index must be part of the corresponding Natural view.

END TRANSACTION

This statement indicates the end of a logical transaction and releases all SQL data locked during the transaction. All data modifications are committed and made permanent.

How the statement is translated and which command is actually issued depends on the TP-monitor environment:

- In batch mode and under TSO, the END TRANSACTION statement is translated into an SQL COMMIT WORK command.

An END TRANSACTION statement must not be placed within a database loop, since all cursors are closed when a logical unit of work ends. Instead, it has to be placed outside such a loop or after the outermost loop of nested loops.

If an external program written in another standard programming language is called from a Natural program, this external program must not contain its own COMMIT command if the Natural program issues database calls, too. The calling Natural program must issue the END TRANSACTION statement for the external program.

Note:
Transaction data cannot be written to SQL databases.

FIND

The FIND statement corresponds to the SQL SELECT statement.

Example:

Natural statements:

```
FIND EMPLOYEES WITH NAME = 'BLACKMORE'
   AND AGE EQ 20 THRU 40
OBTAIN PERSONNEL_ID NAME AGE
```

Equivalent SQL statements:

```
SELECT PERSONNEL_ID, NAME, AGE
  FROM EMPLOYEES
    WHERE NAME = 'BLACKMORE'
      AND AGE BETWEEN 20 AND 40
```

Natural internally translates a FIND statement into an SQL SELECT statement as described in *Processing of SQL Statements Issued by Natural* in the section *Internal Handling of Dynamic Statements*. The SELECT statement is executed by an OPEN CURSOR statement followed by a FETCH command. The FETCH command is executed repeatedly until either all records have been read or the program flow exits the FIND processing loop. A CLOSE CURSOR command ends the SELECT processing.

The WITH clause of a FIND statement is converted to the WHERE clause of the SELECT statement. The basic search criterion for an SQL table can be specified in the same way as for an Adabas file. This implies that only database fields which are defined as descriptors can be used to construct basic search criteria and that descriptors cannot be compared with other fields of the Natural view (that is, database fields) but only with program variables or constants.

Note:
As each database field (column) of an SQL table can be used for searching, any database field can be defined as a descriptor in a Natural DDM.

The WHERE clause of the FIND statement is evaluated by Natural after the rows have been selected via the WITH clause. Within the WHERE clause, non-descriptors can be used and database fields can be compared with other database fields.

Note:
SQL tables do not have sub-, super-, or phonetic descriptors.

A FIND NUMBER statement is translated into a SELECT statement containing a COUNT(*) clause. The number of rows found is returned in the Natural system variable *NUMBER as described in the Natural *System Variables* documentation.

The FIND UNIQUE statement can be used to ensure that only one record is selected for processing. If the FIND UNIQUE statement is referenced by an UPDATE statement, a non-cursor (searched) UPDATE operation is generated instead of a cursor-oriented (positioned) UPDATE operation. Therefore, it can be used if you want to update an SQL primary key. It is, however, recommended to use Natural SQL Searched UPDATE statement to update a primary key.

In static mode, the FIND NUMBER and FIND UNIQUE statements are translated into a SELECT SINGLE statement as described in the section *Natural SQL Statements*.

The FIND FIRST statement cannot be used. The PASSWORD, CIPHER, COUPLED and RETAIN clauses cannot be used either.

The SORTED BY clause of a FIND statement is translated into the SQL SELECT ... ORDER BY clause, which follows the search criterion. Because this produces a read-only result table, a row read with a FIND statement that contains a SORTED BY clause cannot be updated or deleted.

A limit on the depth of nested database loops can be specified at installation time. If this limit is exceeded, a Natural error message is returned.

FIND when Using the File Server

As far as the file server is concerned, there are no programming restrictions with selection statements. It is, however, recommended to make yourself familiar with its functionality considering performance and file server space requirements.

GET

This statement is ISN-based and therefore cannot be used with SQL tables.

HISTOGRAM

The HISTOGRAM statement returns the number of rows in a table which have the same value in a specific column. The number of rows is returned in the Natural system variable *NUMBER as described in Natural *System Variables* documentation.

Example:

Natural statements:

```
HISTOGRAM EMPLOYEES FOR AGE
OBTAIN AGE
```

Equivalent SQL statements:

```
SELECT COUNT(*), AGE FROM EMPLOYEES
  WHERE AGE > -999
  GROUP BY AGE
  ORDER BY AGE
```

Natural translates the HISTOGRAM statement into an SQL SELECT statement, which means that the control flow is similar to the flow explained for the FIND statement.

READ

The READ statement can also be used to access SQL tables. Natural translates a READ statement into an SQL SELECT statement.

READ PHYSICAL and READ LOGICAL can be used; READ BY ISN, however, cannot be used, as there is no SQL equivalent to Adabas ISNs. The PASSWORD and CIPHER clauses cannot be used either.

Since a READ LOGICAL statement is translated into a SELECT ... ORDER BY statement - which produces a read-only table -, a row read with a READ LOGICAL statement cannot be updated or deleted (see Example 1). The start value can only be a constant or program variable; any other field of the Natural view (that is, any database field) cannot be used.

A READ PHYSICAL statement is translated into a SELECT statement without an ORDER BY clause and can therefore be updated or deleted (see Example 2).

Example 1:

Natural statements:

```
READ PERSONNEL BY NAME
OBTAIN NAME FIRSTNAME DATEOFBIRTH
```

Equivalent SQL statements:

```
SELECT NAME, FIRSTNAME, DATEOFBIRTH FROM PERSONNEL
  WHERE NAME >= ' '
    ORDER BY NAME
```

Example 2:

Natural statements:

```
READ PERSONNEL PHYSICAL
OBTAIN NAME
```

Equivalent SQL statements:

```
SELECT NAME FROM PERSONNEL
```

If the READ statement contains a WHERE clause, this clause is evaluated by the Natural processor after the rows have been selected according to the descriptor value(s) specified in the search criterion.

READ when using the File Server

As far as the file server is concerned there are no programming restrictions with selection statements. It is, however, recommended to make yourself familiar with its functionality considering performance and file server space requirements.

STORE

The STORE statement is used to add a row to an SQL table. The STORE statement corresponds to the SQL statement INSERT.

Example:

Natural statements:

```
STORE RECORD IN EMPLOYEES
  WITH PERSONNEL_ID = '2112'
       NAME         = 'LIFESON'
       FIRST_NAME   = 'ALEX'
```

Equivalent SQL statements:

```
INSERT INTO EMPLOYEES (PERSONNEL_ID, NAME, FIRST_NAME)
  VALUES ('2112', 'LIFESON', 'ALEX')
```

The PASSWORD, CIPHER and USING/GIVING NUMBER clauses cannot be used.

UPDATE

The Natural DML UPDATE statement updates a row in an SQL table which has been read with a preceding FIND, READ, or SELECT statement. It corresponds to the SQL statement UPDATE WHERE CURRENT OF *cursor-name* (positioned UPDATE), which means that only the row which was read last can be updated.

UPDATE when using the File Server

If a row rolled out to the file server is to be updated, Natural automatically rereads the original row from the database to compare it with its image stored in the file server. If the original row has not been modified in the meantime, the UPDATE operation is performed. With the next terminal I/O, the transaction is terminated and the row is definitely updated on the database.

If the UPDATE operates on a scrollable cursor, the row on the file server and the row in the base table are updated. If the row no longer qualifies for the search criteria of the related SELECT statement after the update, the row is marked as UPDATE hole on the file server.

However, if any modification is detected, the row will not be updated and Natural issues the NAT3703 error message.

Since an UPDATE statement requires rereading a single row by Natural, a unique index must be available for this table. All columns which comprise the unique index must be part of the corresponding Natural view.

UPDATE with FIND/READ

As explained with the FIND statement, Natural translates a FIND statement into an SQL SELECT statement. When a Natural program contains a DML UPDATE statement, this statement is translated into an SQL UPDATE statement and a FOR UPDATE OF clause is added to the SELECT statement.

Example:

```
FIND EMPLOYEES WITH SALARY < 5000
  ASSIGN SALARY = 6000
  UPDATE
```

Natural would translate the above Natural statements into SQL and assign a cursor name (for example, CURSOR1) as follows:

```
DECLARE CURSOR1 CURSOR FOR
SELECT SALARY FROM EMPLOYEES WHERE SALARY < 5000
  FOR UPDATE OF SALARY
UPDATE EMPLOYEES SET SALARY = 6000
  WHERE CURRENT OF CURSOR1
```

Both the SELECT and the UPDATE statement refer to the same cursor.

Due to SQL logic, a column (field) can only be updated if it is contained in the FOR UPDATE OF clause; otherwise updating this column (field) is rejected. Natural includes automatically all columns (fields) into the FOR UPDATE OF clause which have been modified anywhere in the Natural program or which are input fields as part of a Natural map.

However, an SQL column is not updated if the column (field) is marked as "not updateable" in the Natural DDM. Such columns (fields) are removed from the FOR UPDATE OF list without any warning or error message. The columns (fields) contained in the FOR UPDATE OF list can be checked with the LISTSQL command.

The Adabas short name in the Natural DDM determines whether a column (field) can be updated.

The following table shows the ranges that apply:

Short-Name Range	Type of Field
AA - N9	non-key field that can be updated.
Aa - Nz	non-key field that can be updated.
OA - O9	primary key field.
PA - P9	ascending key field that can be updated.
QA - Q9	descending key field that can be updated.
RA - X9	non-key field that cannot be updated.
Ra - Xz	non-key field that cannot be updated.
YA - Y9	ascending key field that cannot be updated.
ZA - Z9	descending key field that cannot be updated.
1A - 9Z	non-key field that cannot be updated.
1a - 9z	non-key field that cannot be updated.

Be aware that a primary key field is never part of a `FOR UPDATE OF` list. A primary key field can only be updated by using a non-cursor `UPDATE` operation (see also `UPDATE` in the section *Natural SQL Statements*).

A row read with a `FIND` statement that contains a `SORTED BY` clause cannot be updated (due to SQL limitations as explained with the `FIND` statement). A row read with a `READ LOGICAL` cannot be updated either (as explained with the `READ` statement).

If a column is to be updated which is redefined as an array, it is strongly recommended to update the whole column and not individual occurrences; otherwise, results are not predictable. To do so, in reporting mode you can use the `OBTAIN` statement (as described in the Natural Statements documentation), which must be applied to all field occurrences in the column to be updated. In structured mode, however, all these occurrences must be defined in the corresponding Natural view.

The data locked by an `UPDATE` statement are released when an `END TRANSACTION` (`COMMIT WORK`) or `BACKOUT TRANSACTION` (`ROLLBACK WORK`) statement is executed by the program.

Note:
If a length indicator field or `NULL` indicator field is updated in a Natural program without updating the field (column) it refers to, the update of the column is not generated for SQL and thus no updating takes place.

UPDATE with SELECT

In general, the DML `UPDATE` statement can be used in both structured and reporting mode. However, after a `SELECT` statement, only the syntax defined for Natural structured mode is allowed:

```
UPDATE [ RECORD ]   [ IN ] [ STATEMENT ] [( r )]
```

This is due to the fact that in combination with the `SELECT` statement, the DML `UPDATE` statement is only allowed in the special case of:

```
   ...
SELECT ...
   INTO VIEW view-name
   ...
```

Thus, only a whole Natural view can be updated; individual columns (fields) cannot.

Example:

```
DEFINE DATA LOCAL
01 PERS VIEW OF SQL-PERSONNEL
  02 NAME
  02 AGE
END-DEFINE

SELECT *
  INTO VIEW PERS
  FROM SQL-PERSONNEL
  WHERE NAME LIKE 'S%'

    IF NAME = 'SMITH'
      ADD 1 TO AGE
    UPDATE
    END-IF

END-SELECT
    ...
```

In combination with the DML `UPDATE` statement, any other form of the `SELECT` statement is rejected and an error message is returned.

In all other respects, the DML `UPDATE` statement can be used with the `SELECT` statement in the same way as with the Natural `FIND` statement described earlier in this section and in the *Natural Statements* documentation.

Natural SQL Statements

This section covers points you have to consider when using Natural SQL statements with Natural SQL Gateway. These SQL specific points mainly consists in syntax restrictions or enhancements which belong to the Extended Set of Natural SQL syntax. The Extended Set is provided in addition to the Common Set to support database specific features.

This section covers the following topics:

- Syntactical Items
- COMMIT
- DELETE
- INSERT

- PROCESS SQL
- ROLLBACK
- SELECT
- Natural System Variables
- Error Handling

Syntactical Items

The following common syntactical items are either Natural SQL Gateway (NSB) specific and do not conform to the standard SQL syntax definitions (that is, to the Common Set of Natural SQL syntax) or impose restrictions when used with Natural SQL Gateway (see also *SQL Statements* in the Natural *Statements* documentation).

This section covers the following topics:

- atom
- factor
- scalar-function
- column-function
- scalar-operator
- special-register
- case-expression

atom

An atom can be either a parameter (that is, a Natural program variable or host variable) or a constant.

factor

The following factors are specific to Natural SQL Gateway and belong to the Natural Extended Set:

```
special-register
scalar-function (scalar-expression, ...)
case-expression
```

scalar-function

A scalar function is a built-in function that can be used in the construction of scalar computational expressions. Scalar functions are specific to Natural SQL Gateway and belong to the Natural Extended Set.

See the *CONNX Users Guide* for available scalar functions.

Each scalar function is followed by one or more scalar expressions in parentheses. The number of scalar expressions depends upon the scalar function. Multiple scalar expressions must be separated from one another by commas.

Example:

```
SELECT NAME
  INTO NAME
  FROM SQL-PERSONNEL
  WHERE SUBSTR ( NAME, 1, 3 ) = 'Fri'
       ...
```

column-function

A column function returns a single-value result for the argument it receives. The argument is a set of like values, such as the values of a column. Column functions are also called aggregating functions.

The following column functions conform to standard SQL.

```
AVG
COUNT
MAX
MIN
SUM
```

scalar-operator

The concatenation operator (CONCAT or "||") does not conform to standard SQL and belongs to the Natural Extended Set.

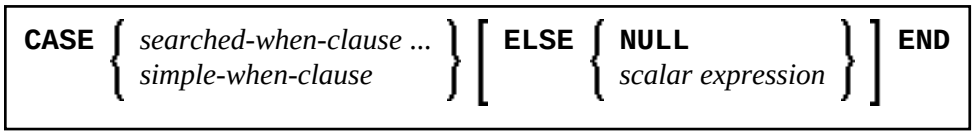
special-register

The following special registers do not conform to standard SQL and belong to the Natural Extended Set:

USER

A reference to a special register returns a scalar value.

case-expression



Case-expressions do not conform to standard SQL and are therefore supported by the Natural SQL Extended Set only.

Example:

```
   DEFINE DATA LOCAL
   01 #EMP
   02 #EMPNO (A10)
   02 #FIRSTNME (A15)
   02 #MIDINIT  (A5)
   02 #LASTNAME (A15)
   02 #EDLEVEL   (A13)
   02 #INCOME (P7)
   END-DEFINE
  SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME,
         (CASE WHEN EDLEVEL < 15 THEN 'SECONDARY'
               WHEN EDLEVEL < 19 THEN 'COLLEGE'
```

```
            ELSE             'POST GRADUATE'
         END ) AS EDUCATION, SALARY + COMM AS INCOME
        INTO
        #EMPNO, #FIRSTNME, #MIDINIT, #LASTNAME,
        #EDLEVEL, #INCOME
          FROM DSN8510-EMP
          WHERE (CASE WHEN SALARY = 0 THEN NULL
                                      ELSE SALARY / COMM
                                      END ) > 0.25
DISPLAY #EMP
END-SELECT
END
```

COMMIT

The SQL `COMMIT` statement indicates the end of a logical transaction and releases all SQL data locked during the transaction. All data modifications are made permanent.

`COMMIT` is a synonym for the Natural `END TRANSACTION` statement as described in the section *Natural DML Statements*.

No transaction data can be provided with the `COMMIT` statement.

If the file server is used, an implicit end-of-transaction is issued after each terminal I/O.

If an external program written in another standard programming language is called from a Natural program, this external program must not contain its own `COMMIT` command if the Natural program issues database calls, too. The calling Natural program must issue the `COMMIT` statement for the external program.

Further details and syntax: `COMMIT` in *SQL Statements* in the Natural *Statements* documentation.

DELETE

Both the cursor-oriented or positioned `DELETE`, and the non-cursor or searched `DELETE` SQL statements are supported as part of Natural SQL Gateway; the functionality of the positioned `DELETE` statement corresponds to that of the Natural DML `DELETE` statement.

With Natural SQL Gateway, a table name in the `FROM` clause of a searched `DELETE` statement can be assigned a correlation-name. This does not correspond to the standard SQL syntax definition and therefore belongs to the Natural Extended Set.

The searched `DELETE` statement must be used, for example, to delete a row from a self-referencing table, since with self-referencing tables a positioned `DELETE` is not allowed by Natural SQL Gateway.

Further details and syntax: `DELETE` in *SQL Statements* in the Natural *Statements* documentation.

INSERT

The `INSERT` statement is used to add one or more new rows to a table.

Since the INSERT statement can contain a select expression, all the NSB specific syntactical items described above apply.

Further details and syntax: INSERT in *SQL Statements* in the Natural *Statements* documentation.

PROCESS SQL

The PROCESS SQL statement is used to issue SQL statements to the underlying database. The statements are specified in a statement-string, which can also include constants and parameters.

The set of statements which can be issued is also referred to as Flexible SQL and comprises those statements which can be issued with the SQL statement EXECUTE.

In addition, Flexible SQL includes the following Natural SQL Gateway specific statements:

```
CONNECT
SET CATALOG
SET SCHEMA
GET host-variable = RCI_VERSION
```

CONNECT

The CONNECT statement establishes a connection to the CONNX JDBC server. It has to be executed before any SQL statement is issued against the CONNX JDBC server.

Syntax

```
PROCESS SQL ddm << CONNECT TO :U:server USER :U:user PASSWORD :U:password >>
```

Parameter	Format/Length	Explanation
ddm	Constant 1-32 characters	Specifies the name of a DDM whose DBID is mapped by NTDBID to type SQL and mapped by NTDBID to type CNX.
server	A1 to A128	Specifies a string addressing the CONNX JDBC server , the port number the server listens to and the CDD to be used to access the RDBMS. The string has to have the following format: GATEWAY=*location-name*;PORT=*number*;DD=*cdd-registered-name* *location-name* denotes the the TCP/IP name of the location where the CONNX JDBC server resides. *number* denotes the port number the CONNX JDBC server listens to. Default port number is 7500. *cdd-registered-name* denotes the CDD to be used for this connection. It is a registry name entry, which is mapped to file name in the registry.
user	A1 to A32	Denotes the user ID to logon to the CONNX JDBC server or RDBMS.
password	A1 to A32	Denotes the password to logon to the CONNX JDBC server or RDBMS.

SET CATALOG

Syntax

```
PROCESS SQL ddm << SET CATALOG :U:catalog >>
```

The SET CATALOG statement sets the default catalog to the catalog identified by catalog. The default catalog will be used to identify the database system to be accessed, if the database system is not explicitly specified as first qualifier of a table name in the SQL syntax and if the CDD contains definitions of more than one database system.

Parameter	Format/Length	Explanation
ddm	Constant 1-32 characters	Specifies the name of a DDM whose DBID is mapped by NTDBID to type SQL and mapped by NTDBID to type CNX.
catalog	A1 to A32	Denotes the catalog name to be used as default catalog.

SET SCHEMA

Syntax

```
PROCESS SQL ddm << SET SCHEMA :U:schema >>
```

The SET SCHEMA statement sets the default schema to the schema identified by schema. The default schema will be used to identify the schema to be accessed, if the schema is not explicitly specified as qualifier of a table name in the SQL syntax and if the CDD contains definitions of more than one schema.

Parameter	Format/Length	Explanation
ddm	Constant 1-32 characters	Specifies the name of a DDM whose DBID is mapped by NTDBID to type SQL and mapped by NTDBID to type CNX.
schema	A1 to A32	Denotes the schema name to be used as default schema.

GET *host-variable* = RCI_VERSION

Syntax

```
PROCESS SQL ddm << GET:G:version = RCI_VERSION >>
```

The GET RCI_VERSION statement retrieves the version of the CONNX client software used in the actual session. It could be executed before any connection is established.

Parameter	Format/Length	Explanation
ddm	Constant 1-32 characters	Specifies the name of a DDM whose DBID is mapped by NTDBID to type SQL and mapped by NTDBID to type CNX.
version	A1 to A128	Receives the version string of the CONNX client software. It looks like the following: RCI: 4.1.1 CONNX 10.5 SP3 (build 8003).

To avoid transaction synchronization problems between the Natural environment and SQL, the COMMIT and ROLLBACK statements must not be used within PROCESS SQL.

Further details and syntax:PROCESS SQL in *Natural SQL Statements* in the Natural *Statements* documentation.

ROLLBACK

The SQL ROLLBACK statement undoes all database modifications made since the beginning of the last logical transaction. Logical transactions can start either after the beginning of a session or after the last COMMIT/END TRANSACTION or ROLLBACK/BACKOUT TRANSACTION statement. All records held during the transaction are released.

ROLLBACK is a synonym for the Natural statement BACKOUT TRANSACTION as described in the section *Natural DML Statements*.

However, if the file server is used, only changes made to the database since the last terminal I/O are undone.

As all cursors are closed when a logical unit of work ends, a ROLLBACK statement must not be placed within a database loop; instead, it has to be placed outside such a loop or after the outermost loop of nested loops.

If an external program written in another standard programming language is called from a Natural program, this external program must not contain its own ROLLBACK command if the Natural program issues database calls, too. The calling Natural program must issue the ROLLBACK statement for the external program.

Further details and syntax: ROLLBACK in *Natural SQL Statements* in the Natural Statements documentation.

SELECT

This section covers the following topics:

- Function
- Syntax Description
- SELECT - Cursor-Oriented
- SELECT SINGLE - Non-Cursor-Oriented
- NSB - UPDATE

See also the following sections in the *Database Management System Interfaces* documentation:

- *SELECT SINGLE - Non-Cursor-Oriented* in the *Natural for DB2* part.
- *SELECT* in the *Natural for SQL/DS* part.

Function

The SELECT statement supports both the cursor-oriented selection that is used to retrieve an arbitrary number of rows and the non-cursor selection (singleton SELECT) that retrieves at most one single row.

Syntax Description

The syntax description of the SQL SELECT statement could be found here: *SELECT - SQL*.

SELECT - Cursor-Oriented

Like the Natural FIND statement, the cursor-oriented SELECT statement is used to select a set of rows (records) from one or more SQL tables, based on a search criterion. Since a database loop is initiated, the loop must be closed by a LOOP (in reporting mode) or END-SELECT statement (in structured mode). With this construction, Natural uses the same loop processing as with the FIND statement.

In addition, no cursor management is required from the application program; it is automatically handled by Natural.

SELECT SINGLE - Non-Cursor-Oriented

The Natural statement SELECT SINGLE provides the functionality of a non-cursor selection (singleton SELECT); that is, a select expression that retrieves at most one row without using a cursor.

Since SQL supports the singleton SELECT command in static SQL only, in dynamic mode, the Natural SELECT SINGLE statement is executed in the same way as a set-level SELECT statement, which results in a cursor operation. However, Natural checks the number of rows returned by SQL. If more than one row is selected, a corresponding error message is returned.

Further details and syntax: See also the SELECT statement for a cursor-oriented selection of rows.

NSB - UPDATE

Both the cursor-oriented or positioned UPDATE, and the non-cursor or Searched UPDATE SQL statements are supported as part of Natural SQL. Both of them reference either a table or a Natural view.

With SQL, the name of a table or Natural view to be referenced by a searched UPDATE can be assigned a correlation-name. This does not correspond to the standard SQL syntax definition and therefore belongs to the Natural Extended Set.

The Searched UPDATE statement must be used, for example, to update a primary key field, since SQL does not allow updating of columns of a primary key by using a positioned UPDATE statement.

Note:
If you use the SET * notation, all fields of the referenced Natural view are added to the FOR UPDATE OF and SET lists. Therefore, ensure that your view contains only fields which can be updated; otherwise, a negative SQLCODE is returned by SQL.

Further details and syntax: UPDATE in *Natural SQL Statements* in the Natural *Statements* documentation.

Natural System Variables

When used with Natural SQL Gateway, there are restrictions for the following Natural system variables:

- *ISN
- *NUMBER

*ISN

As there is no SQL equivalent of Adabas ISNs, the system variable *ISN is not applicable to SQL tables.

*NUMBER

When used with a FIND NUMBER or HISTOGRAM statement, *NUMBER contains the number of rows actually found.

When applied to data from an SQL table in any other case, the system variable *NUMBER only indicates whether any rows have been found. If no rows have been found, *NUMBER is 0. Any value other than 0 indicates that at least one row has been found; however, the value contained in *NUMBER has no relation to the number of rows actually found.

The reason is that if *NUMBER were to produce a valid number, Natural would have to translate the corresponding FIND statement into an SQL SELECT statement including the special function COUNT(*); however, a SELECT containing a COUNT function would produce a read-only result table, which would not be available for updating. In other words, the option to update selected data was given priority in Natural over obtaining the number of rows that meet the search criteria.

To obtain the number of rows affected by the Natural SQL statements Searched UPDATE, Searched DELETE and INSERT, the Natural subprogram NDBNROW is provided. Alternatively, you can use the Natural system variable *ROWCOUNT as described in the Natural *System Variables* documentation.

Error Handling

In contrast to the normal Natural error handling, where either an ON ERROR statement is used to intercept execution time errors or standard error message processing is performed and program execution is terminated, the enhanced error handling of Natural SQL Gateway provides an application controlled reaction to the encountered SQL error.

Two Natural subprograms, NDBERR and NDBNOERR, are provided to disable the usual Natural error handling and to check the encountered SQL error for the returned SQL code.

For further information on Natural subprograms provided for SQL, see the section *Natural Subprograms*.