# NatStar

Version 5.00 Edition 1

# NS-DK

Version 5.00 Edition 1

# NatWeb

Version 4.00 Edition 1

**MySQL**

Ref. n° NS500MYSQLU00033

# Contents

# About this Manual

This is the MySQL manual for Nat System‘s development tools. This manual describes the MySQL interface allowing the access to a MySQL database.

## Relationship to other manuals

☞ Before reading this manual you are expected to have read the « Overview » and « Getting started » manuals. You should not need to use this manual unless you have been advised to do so or if you are already an experienced Nat System developer. If this is the case, you can use this manual to learn in detail about the components it describes.

☞ Strictly speaking, in standard use of NatStar’s Information Modeling tool, you don’t have to program data accesses yourself. The Information Modeling engine takes care of that. In this case, you don’t need to look at the libraries described in this manual. However this manual will prove usefeul if you want to program your applications’ data accesses yourself.

## Organization of the manual

This manual contains one chapter, which describes the set of API components of the MySQL interface.

| | |
|---|---|
| **Chapter 1** | **MySQL interface**<br>Describes the functions of the NSW2MY40 library associated with the MySQL database. |

# Conventions

## Typographic conventions

| | |
|---|---|
| **Important term** | Important terms are printed in **bold**. |
| *Interface component* | The names of windows, dialog boxes, controls, buttons, menus and options are printed in *italics*. |
| [F9] | Function key names appear in square brackets. |
| FILENAME | Filenames are printed in UPPERCASE. |
| `syntax example` | Syntax examples are printed in a `fixed-width font.` |

## Notational conventions

| | |
|---|---|
| • | A round bullet is used for lists |
| ♦ | A diamond is used for alternatives |
| **1.** | Numbers are used to mark the steps in a procedure to be carried out in sequence |

## Operating conventions

| | |
|---|---|
| Choose *XXX* \ *YYY* | This means you need to open the *XXX* menu, then choose the *YYY* command (option) from this menu.<br>You can perform this action using the mouse or mnemonic characters on the keyboard. |
| Click the *XXX* \ *YYY* button | This means you need to display the tool bar named *XXX*, then click the *YYY* button in this tool bar (the name of each button is shown by its help bubble).<br>You can only perform this action with the mouse. |
| Choose the *XXX* button | This means you need to choose the *XXX* button in a dialog box.<br>You can perform this action using the mouse or mnemonic characters on the keyboard. |

## Icon codes

| | |
|---|---|
| ☞ | **Comment,** note, etc. |
| | **Reference** to another part of the documentation |
| | **Danger**: precaution to be taken, irreversible action, etc. |
| | **Suggestion**: helpful hints, etc. |
| | **To go a step further**: level of detail or expertise greater than the average level of the document |

| | |
|---|---|
| | Indicates specific information on using the software under DOS- Windows (all versions) |
| | Indicates specific information on using the software under DOS- Windows 32 bits |
| 32 bits | Indicates specific information on using the software under OS/2- PM 2.x or DOS- Windows 32 bits |
| UNIX | Indicates specific information on using the software under Unix systems |

# Chapter 1 MySQL interface

The NSW2MY40 library allows your applications built with Nat System development tools to interface with client versions of MySQL.

***This chapter explains***

- How to install this library
- The components in this library, arranged in functional categories
- The reference of the components in this library
- The reference of the NS_FUNCTION extensions in the library

# Contents

# Introduction

The NSW2MY40 library allows your applications built with Nat System's development tools to interface with client versions of MySQL.

The version of MySQL supported by Nat System is 4.0.15. This version does not support stored procedures, RECORD/REEXECUTE mode and the insertion of images/texts from files.

## Correspondence between drivers and MySQL versions

The name of the driver is used with THINGS_DB_INIT instruction for NatStar and SQL_INIT instruction for NatStar, NatWeb and NS-DK.

The following table presents MySQL's versions and the corresponding drivers.

| **MySQL's version** | **Driver** |
|---|---|
| **MySQL 4.0.15** | NSW2MY40.DLL |

# Installation

Copy the file NSW2MY40.DLL into the directory that contains the DLLs for your Nat System environment (C:\NATSTAR\BIN, C:\NATWEB\BIN, and so on.)

The SQL libraries supplied with your Nat System development tools interface with the DLLs supplied by the DBMS manufacturer. In some cases, a utility also needs to be run. Check your configuration using the manuals supplied by your MySQL vendor.

# Implicit Output Data Conversions

Refer to MySQL User Manual to see the allowed conversions

Nevertheless, for certain data, use the following conversions :

| MySQL | NCL |
|---|---|
| TINYINT | INT(1) |
| BIT | INT(1) |
| BOOL | INT(1) |
| SMALLINT | INT(2) |
| MEDIUMINT | INT(4) |
| INT | INT(4) |
| INTEGER | INT(4) |
| BIGINT | INT(4) |
| FLOAT | NUM(4) |
| DOUBLE | NUM(8) |
| REAL | NUM(8) |
| DECIMAL | NUM(4) |
| DEC | NUM(4) |
| NUMERIC | NUM(4) |
| DATE * | CSTRING(30) |
| DATETIME | CSTRING(30) |
| TIMESTAMP | CSTRING(30) |
| TIME | CSTRING(10) |
| YEAR | INT(4) |
| CHAR | CSTRING |
| VARCHAR ** | CSTRING |
| ENUM *** | CSTRING |
| SET *** | CSTRING |
| TINYBLOB | CHAR/SQL_IMAGE |
| TINYTEXT | CHAR |
| BLOB | CHAR/SQL_IMAGE |
| TEXT | CHAR |
| MEDIUMBLOB | CHAR/SQL_IMAGE |
| MEDIUMTEXT | CHAR/FILE |

| | |
|---|---|
| LONGBLOB | CHAR/SQL_IMAGE |
| LONGTEXT | CHAR/FILE |

(*) Size equal to 30, to cover all formats.

(**) Size limited to 255, otherwise use the Blob types.

(***)Size equal to the longer string of the list.

# Functional categories in the NSW2MY40 library

Here is a list, arranged by functional category, of the instructions, functions and constants in the NSW2MY40 library.

## Initializing and stopping application use of the DBMS



## Opening and closing a database



## Managing errors



## Executing a SQL command: SELECT, INSERT, UPDATE, CREATE TABLE …



## Managing the cursor



## Managing blobs

## Managing the DBMS behavior



## Managing the current database



## Managing the MySQL server

# NSW2MY40 library reference

# SQL_INIT instruction

Initializes the communication's environment of MySQL Application/Base.

**Syntax** **SQL_INIT** *DLL_name*

**Parameter**

| | | | |
|---|---|---|---|
| *DLL_name* | CSTRING | I | name of the driver to load |

**Notes**

1. This must be the first SQL_ instruction called by any application that wants to use MySQL with NCL: it is responsible for loading the library.
2. The *DLL-name* parameter should contain the name of the DLL used to access the MySQL database: 'NS02MY40'.

**Example**

```
SQL_INIT 'ns02my40'
IF (sql_error%)
      Message 'error at init'&&sql_error%, sql_errmsg$(sql_error%)
      Exit
ENDIF
...
SQL_STOP
```

**See also** SQL_STOP, SQL_INITMULTIPLE%, SQL_STOPMULTIPLE, SQL_STOPALL, SQL_ERROR% , SQL_ERRMSG$

# SQL_STOP instruction

Unloads the current MySQL driver and closes simultaneously all opened databases and cursors.

**Syntax** **SQL_STOP**

**Example**

Refer to the example of SQL_INIT instruction.

**See also** SQL_INIT, SQL_INITMULTIPLE%, SQL_STOPMULTIPLE, SQL_STOPALL, SQL_ERROR%, SQL_ERRMSG$

# SQL_OPEN instruction

Connects the application with the database.

**Syntax** **SQL_OPEN** *logical-DBname, password-string*

**Parameters**

| | | | |
|---|---|---|---|
| *logical-DBname* | CSTRING | I | logical name of the database to open |
| *password-string* | CSTRING | I | password to connect to a server |

**Notes**

1. MySQL could open several databases or the same several times through the network.
2. The *logical-DBname* specified the internal alias of the application to design the connection. It could also correspond to the physical name of the MySQL database.
3. The *password-string* parameter specifies the command string used to connect to a local or remote database, as follows :

   `"USERID[/PASSWORD][@<CONNECTOR>]`

   where :

   `[USERID]` name of the user account on the server

   `[PASSWORD]` password for the user account

   ☞ If the password is not defined, the connection string may be just user@connector.

   `[@<CONNECTOR>]` used to specify which open database will receive the query.

   A set of options to define and/or configure the connection to establish as *host*, *base*, *port*, *timeout* … , separated by the character '**;**'. When an option is optional, do not fill it.

   Example :

   ```
   "host=MyHost;base=MyBase;port=MyPort;socket=MySocket;…"
   ```

   Below the list of possible options available now for the parameter `[@<CONNECTOR>]`:

   **Host=MyHost** : name of the server where MySQL database is installed. Optional if it is a local database.

   **Base=MyBase** : name of the database with which one wants to establish the connection. Optional if the logical name already contains the physical name of the database.

**Port=MyPort** : the port number on which the MySQL server is listening (3306 by default), optional.

**Socket=MySocket** : if the connection is established by a socket file (especially for Unix platforms).

**Timeout=120** : timeout in seconds for a query to return. If the waiting exceeds the delay, a timeout error is raised.

**File=path-of-file.ini** : reads the options in this file instead of the default file %Windir%\my.ini.

**Group=named-group** : reads the options of the group 'named-group' in the my.ini file.

**Compress=N** : for a better performance of the network's transfer, the data are compacted before the transfer. Position this option to N (NO) to deactivate.

**Example 1**

*Logical name = physical name of the database. In this case, it is useless to specify it in the connector.*

```
SQL_OPEN 'mysql', 'root@host=pollux;port=3306;timeout=240;'
; SQL processing
SQL_CLOSE 'mysql'
```

**Example 2**

*Logical name different from physical name, local server, no timeout, no compression*

```
SQL_OPEN 'db1', 'root@port=3306;base=mysql;compress=N'
; SQL processing
SQL_CLOSE 'db1'
```

**See also** SQL_CLOSE, NS_FUNCTION CHANGEDBCNTX, SQL_ERROR%, SQL_ERRMSG$

# SQL_CLOSE instruction

Closes a connection of a database.

**Syntax** **SQL_CLOSE** *logical-DBname*

**Parameter**

| | | | |
|---|---|---|---|
| *logical-DBname* | CSTRING | I | logical name of the database to close |

**Notes**

1. Although that you recommend that you close each of the databases opened by an application, an SQL_CLOSE instruction is automatically generated for each open database when an application is closed.

**Example**

Refer to the example of the SQL_OPEN instruction.

**See also** SQL_OPEN

# SQL_EXEC instruction

Executes an SQL command: SELECT, INSERT, UPDATE, CREATE TABLE …

**Syntax** **SQL_EXEC** [**AT** *logical-DBname*] *SQL-command* [**USING** *cursor-handle*]

**Parameters**

| | | | |
|---|---|---|---|
| *logical-DBname* | CSTRING | I | logical name of database |
| *SQL-command* | CSTRING | I | SQL command to execute |
| *cursor-handle* | INT(4) | I | cursor value |

**Notes**

1. The SQL command is passed directly without quotes. It can correspond to any Oracle SQL command, whether it's a data definition command (CREATE TABLE, CREATE INDEX, ....) or a data manipulation command (SELECT, INSERT, UPDATE, ...).
2. The query is sent to the database specified after the AT command (without quotes and *case-sensitive*). If the AT command isn't specified, the SQL_EXEC executes on the current database.
3. If USING *cursor-handle* is specified, it indicates which cursor previously opened by SQL_OPENCURSOR% must be used to execute the SQL command. If no cursor has been opened, the cursor's value is that of DEFAULT_CURSOR: -1.
4. The SQL command can return values in NCL variables. For this, just pass these variables in parameters.
5. It is possible to pass a segment's field as a data-receiving variable in an SQL query.
6. The commands SQL_EXEC, SQL_EXECSTR and SQL_EXEC_LONGSTR depend on the SQL language accepted by the DBMS in use (Refer to the *MySQL* documentation ou http://www.mysql.com).
7. For SQL commands that are too long, it is possible to use the special continuation character "\":
```
SQL_EXEC UPDATE SAMPLE SET SOCIETE = :A$\
                       WHERE VILLE = :C$\
AND \
                                PAYS  = :D$
```
8. The types of variables recognized by the interface are:
   - INT(1), INT(2), and INT(4),
   - NUM(8), NUM(4),
   - STRING,
   - CSTRING,
   - CHAR.

**9.** The INTO clause is used by the SELECT and FETCH commands. It defines a list of host variables. Its syntax is:

```
INTO :var1 [:indic1] [, :var2 [:indic2] \
[, ... ] ]
```

**10.** We suggest using INTO in a SELECT to improve performance because during a FETCH, in each loop, the driver has to analyze the variables of the INTO clause. Using the INTO clause in a FETCH should be restricted to doing things like be entering elements into an array.

**11.** Always put a ":" before the name of a variable or a flag.

**12.** A flag is an NCL integer variable which can have the following values:

- NULL_VALUE_INDICATOR (i.e. -1) indicates that the associated NCL variable which precedes it has a NULL value.
- Any other value indicates that the associated NCL variable which precedes it has a NOT NULL value, and can therefore be used.

**13.** In SQL, NULL does not mean 0 or an empty string ("""). However, to make it possible to assign a value in all cases, when a column contains a NULL value, a numeric target NCL variable will be assigned at 0 and a string target NCL variable will be assigned an empty string ("""). MySQL differentiates the empty strings ("""") from NULL strings (not typed).

**14.** MySQL doesn't check the coherence of data's type to insert with the type of the received column. An implicit conversion is done if necessary.
Examples :
A non numerical CHAR inserted in a INTEGER column is inserted 0. Indeed, the conversion stopped to the first non numerical CHAR.
'3 little pigs' inserted in an INTEGER column gives '3'.
A FLOAT inserted in a INTEGER column is truncated to its integer part (3.25 → 3).
A non valid date inserted in a DATE column is inserted 0.
(0000-00-00 US format, by default).

**15.** MySQL extract all blanks from character strings (Trim). So the blanks at the end of the string are never inserted, nor taken into consideration in the equality between two strings.

**16.** For the Images/texts, the insertion is made from memory buffers of CHAR NCL type or SQL_IMAGE NCL which the sze is limited to 64kb.

**17.** The insertion from files (TYPE_SQL_INSERT_BLOB%) is not possible for the moment. Because of requests's preparation containing huge data in 4.0.15 version (wait for the 4.1). However the extraction is already possible.

**18.** Each MySQL data could be stored in whatever NCL type. But it is strongly recommended to use the correct NCL type defined above with the correct size to improve precision during the conversion.

**19.** Seeing that at insertion, NULL are different from an empty string ("""), so these two following requests are totally different :
SELECT * from MyTable where MyCol is NULL;

SELECT * from MyTable where MyCol='';

**20.** For images/texts, you can use TYPE_SQL_SELECT_BLOB% to extract them in files form. Images/texts less than 64 kb can be extract directly in memory buffers with NCL type CHAR or SQL_IMAGE NCL.

**Example 1**

```
local dataptr%, size%, nbread%(2), file%, fname$
local sql_image localimage

sql_exec drop table test_image
sql_exec create table test_image (id integer, comment varchar(255), colimage
blob)
sql_exec ns_function imageon

fname$ = "d:\Documents\Images\tintin.bmp"
size% = fgetsize%(fname$)
new size%, dataptr%
file% = f_open% (1, fname$)
f_blockread file%, dataptr%, size%, nbread%
; test the f_error% error
localimage.realsize = size%
localimage.length% = size%
localimage.ptr% = dataptr%

sql_exec insert into test_image values (1, "tintin sans milou", :localimage)
; test the sql_error% error

f_close file%
dispose dataptr%
sql_image ns_function imageoff
```

**Example 2**

```
local fname$,file%,nbread%(2),hbmp%
local sql_image localimage
local opt$
local val%

sql_exec ns_function imageon

new 65535, localimage.ptr%
move 65535 to localimage.realsize

sql_exec select colimage into :localimage from test_image
; test the sql_error% error

fname$="e:\temp\tinitin.bmp"
file%=f_create%(1,fname$)

f_blockwrite file%, localimage.ptr%, localimage.realsize, nbread%
; test the f_error% error
f_close file%

dispose localimage.ptr%
sql_exec ns_function imageoff
```

# SQL_EXECSTR instruction

Executes an SQL command: SELECT, INSERT, UPDATE, CREATE TABLE …

**Syntax** **SQL_EXECSTR** *SQL-command* [, *variable* [, *variable* [, ....]]]
[**USING** *handle-name*]

**Parameters**

| | | | |
|---|---|---|---|
| *SQL-command* | CSTRING | I | SQL order to execute |
| *variable* | | I | NCL variable list |
| *handle_name* | INT(4) | I | cursor value |

**Notes**

1. *SQL-command* is either a string *host* variable or a character string containing the SQL command to execute in quotation marks
1. When you use the SQL_EXEC instruction, you write the names of the *host* variables directly in the text of the SQL query. When you use the SQL_EXECSTR instruction, the *host* variables are parameters of the instruction.
2. When you use the SQL_EXECSTR instruction, each *host* variable is represented in the text of the query by a ":" character. The first ":" corresponds to the first *host* variable passed as a parameter, and so on.
3. The other functions of the SQL_EXECSTR command are the same as SQL_EXEC.

**Example**

```
LOCAL REQ$, TABLE$, FATHER$, SON$
LOCAL AGE%, IND1%, IND2%, CURS1%
TABLE$   = "FAMILY"
AGE%     = 20
REQ$ = "SELECT NAME, AGE, CHILDNAME INTO : :,:,: : FROM '" &\
            TABLE$ & "' WHERE AGE > :"
; ---- Open a cursor
CURS1%=SQL_OPENCURSOR%
; ---- Select persons older than 20 from
;      the FAMILY table
SQL_EXECSTR :REQ$, :FATHER$,:IND1%,:AGE%,:SON$,:IND2%,:AGE%, USING CURS1%

WHILE SQL_ERROR% = 0
  IF IND2% = NULL_VALUE_INDICATOR
     INSERT AT END FATHER$ & " does not have a son" TO LBOX
  ELSE
     INSERT AT END FATHER$ & "'s son" &  "is" & SON$ TO LBOX
  ENDIF
  SQL_EXEC FETCH USING CURS1%
ENDWHILE
; ---- Close the cursor
SQL_CLOSECURSOR
```

**See also** SQL_EXEC, SQL_EXEC_LONGSTR, SQL_OPENCURSOR%, SQL_CLOSECURSOR, SQL_ERROR%, SQL_ERRMSG$

# SQL_EXEC_LONGSTR instruction

Executes a very long SQL command: SELECT, INSERT, UPDATE, CREATE TABLE …

**Syntax** **SQL_EXEC_LONGSTR** *sql-string-address, var-array-address, cursor-num*

**Parameters**

| | | | |
|---|---|---|---|
| *sql-string-address* | INT(4) | I | address of the character string containing the SQL statement to execute |
| *var-array-address* | INT(4) | I | address of the array containing the host variables (or indicators) |
| *cursor-num* | INT(2) | I | cursor value |

**Notes**

1. The executed statement can contain any SQL command in the host language (DML or DDL). The size of the string depends on the RDBMS used; it is unlimited for certain database engines and limited to 4096 characters for others.
2. *sql-string-address* is the address of the string which contains the SQL command to execute.
3. *var-array-address* is an array of NCLVAR segments which describe the NCL host variables. If your SQL statement does not use any variables, pass 0 in *var-array-address*.
4. When you use the SQL_EXEC_LONGSTR instruction, each *host* variable is represented in the text of the query by a ":" character. The first ":" corresponds to the first *host* variable in the array of *host* variables, and so on.
5. The NCLVAR segment and any constants used are declared in the NSDBMS library as follows:

```
SEGMENT NCLVAR
  INT     PTR_VAR(4)
  INT     TYPE_VAR(2)
  INTEGER SIZE_VAR
  INT     RESERVED(4)
ENDSEGMENT

CONST TYPE_SQL_INT%     0
CONST TYPE_SQL_STRING%  1
CONST TYPE_SQL_CSTRING% 2
CONST TYPE_SQL_NUM%     3
CONST TYPE_SQL_SEGMENT% 10
CONST TYPE_SQL_IMAGE%   11
CONST TYPE_SQL_SELECT_BLOB%    12
; CONST TYPE_SQL_INSERT_BLOB%  13 (doesn't work with MySQL 4.0)
```

6. This array of segments should have an **index that is greater than** the number of variables used (the last element contains 0). You are therefore advised to fill this array initially (using the NCL FILL verb) to ensure that element 0 actually exists, since the end of the scan is determined by this element.
7. If no cursors have been opened, the cursor value must be set to that of the DEFAULT CURSOR: -1.
8. SQL_EXEC_LONGSTR replaces SQL_EXECLONGSTR%. To use this instruction, you will still find the code you need in the notes of NSDBMS.NCL.
9. The other function of SQL_EXEC_LONGSTR instruction are the same as SQL_EXEC.

**Example**

```
LOCAL NCLVAR VARLIST[3]      ; for 2 variables
LOCAL SQL_STR$               ; string to pass
LOCAL VAR1%, VAR2$           ; host variables
LOCAL CONDITION%             ; input variable

; ---- Set the array to 0
FILL @VARLIST, SIZEOF VARLIST, 0

SQL_STR$            = "SELECT VCHAR, VINT " & "FROM TAB1 " &\ "WHERE VINT >= :"

VARLIST[0].PTR_VAR  = @CONDITION%
varlist[0].TYPE_VAR = TYPE_SQL_INT%
VARLIST[0].SIZE_VAR = SIZEOF @CONDITION%

SQL_EXEC_LONGSTR @SQL_STR$, @VARLIST, DEFAULT_CURSOR
FILL @VARLIST, SIZEOF VARLIST, 0
SQL_STR$ = "FETCH INTO :, :"

VARLIST[0].PTR_VAR  = @var2$
VARLIST[0].TYPE_VAR = TYPE_SQL_CSTRING%
VARLIST[0].SIZE_VAR = SIZEOF var2$
VARLIST[1].PTR_VAR  = @var1%
VARLIST[1].TYPE_VAR = TYPE_SQL_INT%
VARLIST[1].SIZE_VAR = SIZEOF var1%

WHILE SQL_ERROR% = 0

  SQL_EXEC_LONGSTR @SQL_STR$, @VARLIST, DEFAULT_CURSOR
  IF SQL_ERROR% = 0
     MESSAGE "SELECT", VAR1% && VAR2$
  ENDIF
ENDWHILE
```

**See also** FILL (NCL), NSDBMS.NCL, SQL_EXEC, SQL_EXECSTR, SQL_ERROR%, SQL_ERRMSG$

# SQL_OPENCURSOR% function

Opens a cursor and returns its handle.

**Syntax** **SQL_OPENCURSOR%**

**Returned value** INT(4)

**Notes**

1. After opening the cursor, it can be used with the following instructions
```
SQL_EXEC SELECT ... USING cursor-handle
SQL_EXEC FETCH ... USING cursor-handle
```
2. A cursor is an internal resource managed by the NSW2MY40 and is used, for example, to store the current table row position for the next SQL call.
3. When the system is opened, only one cursor is defined, known as the DEFAULT_CURSOR.
4. If no cursors have been opened, this DEFAULT_CURSOR will be used to execute all SQL statements that maintain current positions within the database, including SELECT and FETCH statements.
5. A problem occurs if an SQL statement other than FETCH (for example UPDATE or INSERT) is embedded in a scanning sequence; the current position is lost and the FETCH statement that follows the embedded statement will terminate with the error.

   SQL_OPENCURSOR% solves this problem by executing all SELECT and FETCH commands with the new cursor.
6. Generally speaking, a new cursor should be opened each time you wish to perform a SELECT FETCH scan while another similar scan is still in progress with the last cursor opened.
7. The NSW2MY40 DLL specifically designed for the DBMS stores cursors in a LIFO (Last In First Out) stack: SQL_OPENCURSOR% stacks and SQL_CLOSECURSOR unstacks.
8. The following rules apply when executing a statement with a cursor:
   - Statements are always executed with the specified cursor.
   - If with SQL_EXEC, the USING clause isn't specified, the commands are executed with the DEFAULT_CURSOR.
9. When several databases are opened simultaneously, the cursor opened by SQL_OPENCURSOR% is immediately associated with the current database.
10. If you want to open a cursor in a database other than the current one, you must execute the SQL_EXEC CHANGEDBCNTX:*otherbase$* command to change databases before you execute SQL_OPENCURSOR%.

**Example**

See the example of the SQL_CLOSETHECURSOR instruction.

**See also** SQL_CLOSECURSOR, SQL_OPENTHECURSOR%, SQL_CLOSETHECURSOR, SQL_ERROR%, SQL_ERRMSG$

# SQL_CLOSECURSOR instruction

Closes the last cursor opened and the last occupied by SQL_OPENCURSOR%.

**Syntax** **SQL_CLOSECURSOR**

**Notes**

1. SQL_CLOSECURSOR closes the last cursor opened, situated at the top of the LIFO (Last In First Out) cursor stack.
2. SQL_CLOSECURSOR must only close cursors opened with SQL_OPENCURSOR%.
3. The error codes returned by SQL_ERROR% for this instruction are: -32003 or -32005.
4. The SQL_CLOSECURSOR instruction must not be used with the IM module of NatStar.
5. Nat System recommends you to use SQL_CLOSETHECURSOR instead of SQL_CLOSECURSOR.

**Example**

See the example of the SQL_CLOSETHECURSOR instruction.

**See also** SQL_OPENCURSOR%, SQL_OPENTHECURSOR%, SQL_CLOSETHECURSOR, SQL_ERROR%, SQL_ERRMSG$

# SQL_OPENTHECURSOR% function

Opens a cursor and returns its handle.

**Syntax** **SQL_OPENTHECURSOR%**

**Returned value** INT(2)

**Notes**

1. After opening the cursor, it can be used with the following instructions:
```
SQL_EXEC SELECT ... USING cursor-handle
SQL_EXEC FETCH  ... USING cursor-handle
```
2. A cursor is an internal resource managed by the NSW2MY40 DLL and is used, for example, to store the current table row position for the next SQL call.
3. When the system is opened, only one cursor is defined, known as the DEFAULT_CURSOR.
4. If no cursors have been opened, this DEFAULT_CURSOR will be used to execute all SQL statements that maintain current positions within the database, including SELECT and FETCH statements.
5. A problem occurs if an SQL statement other than FETCH (for example UPDATE or INSERT) is embedded in a scanning sequence; the current position is lost and the FETCH statement that follows the embedded statement will terminate with the error.

   SQL_OPENCURSOR% solves this problem by executing all SELECT and FETCH commands with the new cursor.
6. Generally speaking, a new cursor should be opened each time you wish to perform a SELECT FETCH scan while another similar scan is still in progress with the last cursor opened.
7. The following rules apply when executing a statement with a cursor:
   - Statements are always executed with the specified cursor.
   - If with SQL_EXEC, the USING clause isn't specified, the commands are executed with the DEFAULT_CURSOR.
8. When opening several databases at the same time, the cursor opened by SQL_OPENTHECURSOR% is immediately associated with the current database.
9. If you want to open a cursor in a database other than the current one, you must execute the SQL_EXEC CHANGEDBCNTX:*otherbase$* command to change databases before you execute SQL_OPENCURSOR%.

**Example**

Refer to the example of the SQL_CLOSETHECURSOR instruction.

**See also** SQL_OPENCURSOR%, SQL_CLOSECURSOR, SQL_CLOSETHECURSOR, SQL_ERROR%, SQL_ERRMSG$

# SQL_CLOSETHECURSOR instruction

Closes the cursor associated with the given handle.

**Syntax** **SQL_CLOSETHECURSOR** *cursor-handle*

| **Parameter** | | | | |
|---|---|---|---|---|
| | *cursor-handle* | INT(4) | I | handle of cursor to close |

**Note**

1. SQL_CLOSETHECURSOR can only close cursors opened with SQL_OPENTHECURSOR%.

**Example**

```
; ---- Example showing the two different types of
;      cursors (for clarity, we have not
;      included error test code)

SQL_EXEC ....                    ; uses the default cursor
C1% = SQL_OPENCURSOR%            ; opens the C1% cursor
SQL_EXEC UPDATE ...              ; uses the default cursor
SQL_EXEC SELECT ...              ; uses the default cursor
SQL_CLOSETHECURSOR C1%           ; => error
C2% = SQL_OPENTHECURSOR%         ; opens the C2% cursor
SQL_EXEC UPDATE ...              ; uses the default cursor
SQL_EXEC UPDATE ... USING C1%    ; uses the C1% cursor
SQL_EXEC SELECT ... USING C2%    ; uses the C2% cursor
SQL_EXEC SELECT ... USING C1%    ; uses the C1% cursor
SQL_CLOSECURSOR                  ; closes the C1% cursor
SQL_EXEC UPDATE ....             ; uses the default cursor
SQL_EXEC SELECT .... USING C2%   ; uses the C2% cursor
SQL_CLOSECURSOR%                 ; => error
SQL_CLOSETHECURSOR C2%           ; closes the C2% cursor
SQL_EXEC ....                    ; uses the default cursor
```

**See also** SQL_OPENCURSOR%, SQL_CLOSECURSOR, SQL_OPENTHECURSOR%, SQL_ERROR%, SQL_ERRMSG$

# SQL_ERROR% function

Returns the error code of the last SQL_ instruction executed.

**Syntax** **SQL_ERROR%**

**Returned value** INT(4)

**Notes**

1. SQL_ERROR% complies with SQL conventions. The function returns:
   - 0 if no errors occurred.
   - A positive number for non-fatal errors (the instruction was executed but issued a warning).
   - A negative number for fatal errors (the instruction could not be executed).
2. This function applies to all MySQL driver functions.
3. Two types of return code are sent by the interface :

   - Proprietary DBMS SQL error codes which are described in the editor's manuals.

   - Internal Nat System error codes. They correspond to errors not handles by the host DBMS. These error messages are numbered and have the format "32XXX".

   **Example :**
   ```
   -32004 "NSSQLE004 ** NO MORE CURSORS AVAILABLE"
   ```
4. List of internal Nat System errors common to MySQL :

   **0** "NSSQLE000 ** UNUSED NATSYS ERROR CODE"

   **Cause :** No error occurred, the execution was carried out correctly.

   **+100** "NSSQLW100 ** NO ROW WAS FOUND OR LAST ROW REACHED"

   **Cause :** There are no rows or the last row was reached after a FETCH or SELECT.

   **-32001** "NSSQLE001 ** HEAP ALLOCATION ERROR"

   **Cause :** Internal memory allocation/deallocation error

   **-32002** "NSSQLE002 ** DYNAMIC ALLOCATION ERROR"

   **Cause :** Internal memory allocation/deallocation error

   **-32003** "NSSQLE003 ** DYNAMIC FREE STORAGE ERROR"

   **Cause :** Internal memory allocation/deallocation error

**-32004** "NSSQLE004 ** NO MORE CURSORS AVAILABLE"

**Cause :** Too many cursors opened simultaneously.

**-32005** "NSSQLE005 ** NO MORE CURSORS OR TRYING TO DEALLOCATE ONLY CURSOR"

**-32006** "NSSQLE006 ** INVALID INTO CLAUSE in FETCH/SELECT"

**Cause :** Syntax error in the INTO clause of a SELECT or a FETCH statement

**-32007** "NSSQLE007 ** TOO MANY VARIABLES IN INTO CLAUSE"

**Cause :** More than 200 in the INTO clause

**-32008** "NSSQLE008 ** MISSING HOST VARIABLE AFTER ','"

**Cause :** Syntax error in the INTO clause. Variable missing after a continuation comma.

**+32009** "NSSQLW009 ** INTO CLAUSE : NOT ENOUGH VARIABLES"

**Cause** : A SELECT statement contains an INTO clause with fewer variables than the number of variables returned by the query.

**Warning** : The system will still fill the host variables supplied to it.

**+32010** "NSSQLW010 ** AN OPENED CURSOR WAS CLOSED BY SYSTEM"

**Cause** : Following the arrival of a new SQL command for a cursor, the system forced the closure of a cursor containing an active query.

**-32011** "NSSQLE011 ** WHERE/VALUE CLAUSE : NOT ENOUGH VARIABLES"

**Cause :** Not enough host variables received in the table to be able to substitute the variables specified in the WHERE clause.

**-32012** "NSSQLE012 ** INVALID INPUT VARIABLE DATA TYPE"

**Cause :** Invalid data type in a WHERE clause.

**-32014** "NSSQLE014 ** NO OUTPUT VARIABLES DEFINED FOR FETCH"

**Cause :** The FETCH and the prior SELECT have not defined any output variables (INTO clause).

**-32015** "NSSQLE015 ** CURSOR NOT READY (MISSING SELECT) "

**Cause :** FETCH attempted without a prior SELECT or cursor closed by the system between the SELECT and the FETCH statements.

**-32016** "NSSQLE016 ** INVALID SQL DATA TYPE"

**Cause :** Data type invalid for output.

**-32017** "NSSQLE017 ** INVALID DATA CONVERSION REQUESTED"

**Cause :** Type conversion invalid for output.

STRING -> NUM

NUM -> STRING

REAL -> INTEGER

INTEGER -> REAL

**-32018** "NSSQLE018 ** NUMERIC DATA TYPE : INVALID LENGTH"

**Cause :** Invalid length for the data type (for example, real number with a length of 48)

**-32019** "NSSQLE019 ** INVALID DECIMAL PACKED FORMAT"

**Cause :** Unable to convert data to packed decimal format.

**+32020** "NSSQLW020 ** STRING DATA TRUNCATED"

**Cause :** The string passed as a variable is shorter than the field received from the DBMS. The received field has been truncated.

**-32021** "NSSQLE021 ** RESET STORAGE ERROR"

**Cause :** Deallocation error of the internl heap

**+32022** "NSSQLW022 ** FUNCTION NOT SUPPORTED IN MYSQL DATABASE"

**Cause :** The executed instruction is not available.

**-32023** "NSSQLE023 ** TOO MANY OPENED DATABASES"

**Cause :** More than 5 databases opened simultaneously.

**+32024** "NSSQLW024 ** DB ALREADY OPENED"

**Cause :** The database used with SQL_OPEN has already been opened.

**-32025** "NSSQLE025 ** DB NOT PREVIOUSLY OPENED"

**Cause :** Attempt to close a database that has not been happened.

**-32026** "NSSQLE026 ** INVALID DATABASE NAME REF"

**Cause :** Unknown database name used in the AT clause of the SQL_EXEC statement (database not opened)

**-32028** "NSSQLE028 ** UNABLE TO GET MYSQL LOGIN"

**Cause :** Failed to connect to DB2 (e.g. server name error).

**-32029** "NSSQLE029 ** MYSQL VARIABLE INPUT BIND FAILED"

**Cause :** Type mismatch between a variable and a database field.

**-32030** "NSSQLE030 ** MYSQL VARIABLE OUTPUT BIND FAILED"

**Cause :** Type mismatch between a variable and a database field.

**-32031** "NSSQLE031 ** MYSQL BUFFER FILL ERROR"

**Cause :** Buffer overflow (due to data conversion,etc.)

**-32032** "NSSQLE032 ** RPC PARAMETER NAME EXPECTED"

**Cause :** Remote procedure call : procedure name missing.

**-32033** "NSSQLE033 ** TOO MANY RPC PARAMETERS"

**Cause :** Remote procedure call : procedure name missing.

**-32034** "NSSQLE034 ** RPC PROCEDURE NAME EXPECTED"

**Cause :** Remote procedure call : procedure name missing.

**-32035** "NSSQLE035 ** NOT ENOUGH PARAMETERS FOR RPC CALLS"

**Cause :** Remote procedure call : too many parameters specified

**-32036** "NSSQLE036 ** INVALID RPC PARAMETERS SUPPLIED"

**Cause :** Remote procedure call : invalid parameters specified

**-32037** "NSSQLE037 ** INVALID RPC PROCEDURE INITIALIZATION"

**-32038** "NSSQLE038 ** RPC PROCEDURE EXECUTION FAILED"

**-32039** "NSSQLE039 ** MEMORY CONSISTENT ERROR"

**-32040** "NSSQLE040 ** INVALID TYPE FOR INDICATOR"

**-32041** "NSSQLE041 ** CONTEXT NOT CREATED"

**-32042** "NSSQLE042 ** CONTEXT NOT FOUND"

**-32044** "NSSQLE044 ** NO SET LOGIN TIME"

**-32045** "NSSQLE045 ** NO SET TIME"

**-32046** "NSSQLE046 ** SET MAXPROCS FAILED"

**-32047** "NSSQLE047 ** DB OPEN FAILED"

**-32048** "NSSQLE048 ** DB NOT OPENED"

**-32049** "NSSQLE049 ** LOGIN RECORD NOT ALLOCATED"

**-32050** "NSSQLE050 ** MEMORY DEALLOCATION ERROR"

**-32051** "NSSQLE051 ** CURSOR NOT FOUND"

**-32052** "NSSQLE052 ** MUST EXECUTE SELECT BEFORE THE FETCH COMMAND"

**-32053** "NSSQLE053 ** ERROR IN CLOSING DATABASE"

**-32054** "NSSQLE054 ** ERROR IN EXECUTING SQL STATEMENT"

**-32055** "NSSQLE055 ** ERROR IN EXECUTING FETCH COMMAND"

**-32056** "NSSQLE056 ** INDICATOR'S SIZE TOO SMALL TO HOLD VALUE"

**-32057** "NSSQLE057 ** UNKNOWN NCL VARIABLE TYPE PASSED"

**-32058** "NSSQLE058 ** RPC : INIT ERROR"

**-32059** "NSSQLE059 ** RPC : PARAMETERS FOUND BUT NO VALUES CLAUSE"

**-32060** "NSSQLE060 ** RPC : PARAMETER TYPE MISMATCH"

**-32061** "NSSQLE061 ** RPC : PROCEDURE NAME MISSING"

**-32062** "NSSQLE062 ** RPC : INDICATORS MAY ONLY BE ON OUT VARIABLES"

**-32063** "NSSQLE063 ** RPC : MYSQL ERROR DURING RPC PREPARATION"

**-32064** "NSSQLE064 ** RPC : MYSQL ERROR DURING RPC EXECUTION"

**-32065** "NSSQLE065 ** RPC : MYSQL ERROR DURING RPC EXEC CHECK"

**-32066** "NSSQLE066 ** RPC : PROCEDURE NOT PREPARED"

**-32067** "NSSQLE067 ** LOGGER : CAN'T OPEN FILE"

**-32068** "NSSQLE068 ** PARSER : TOKEN TABLE FULL"

**-32069** "NSSQLE069 ** EXEC : INCOMPATIBLE CURSOR MODE"

**-32070** "NSSQLE070 ** EXEC : MYSQL ERROR DURING SIZE BUFFERING EXECUTION"

**-32071** "NSSQLE071 ** EXEC : MYSQL ERROR DURING SIZE BUFFERING DELETION"

**-32072** "NSSQLE072 ** EXEC : INVALID CURSOR MODE"

**-32073** "NSSQLE073 ** EXEC : THAT ROW IS NOT IN BUFFER"

**-32074** "NSSQLE074 ** EXEC : INCORRECT SYNTAX FOR THIS CURSOR MODE"

**-32075** "NSSQLE075 ** EXEC : MISSING INTO CLAUSE FOR THIS CURSOR MODE"

**-32076** "NSSQLE076 ** EXEC : INVALID SIZE FOR ROW BUFFERING"

**-32077** "NSSQLE077 ** EXEC : INVALID ROW NUMBER"

**-32078** "NSSQLE078 ** EXEC : MEMORY DEALLOCATION ERROR FOR SCROLL STATUS"

**-32079** "NSSQLE079 ** EXEC : MYSQL: ROW IS MISSING IN SCROLL BUFFER"

**-32080** "NSSQLE080 ** NO STATEMENT IN PROGRESS"

**-32081** "NSSQLE081 ** DATA NOT READY TO RESULT PROCESSING"

**-32082** "NSSQLE082 ** INVALID WINDOW HANDLE"

**-32083** "NSSQLE083 ** USER MESSAGE MUST BE RANGE IN 0 AND 15"

**-32084** "NSSQLE084 ** INVALID STATEMENT SEND TO DLL"

**-32085** "NSSQLE085 ** NO MORE RESULT TO FETCH"

**-32086** "NSSQLE086 ** INVALID PARAMETER TO CHANGE OPTION"

**-32087** "NSSQLE087 ** INVALID PARAMETER TO CHANGE OPTION VALUE"

**-32088** "NSSQLE088 ** LOGIN TIME CHANGE FAILED"

**-32089** "NSSQLE089 ** TIMEOUT CHANGE FAILED"

**-32090** "NSSQLE090 ** INVALID NS_FUNCTION STATEMENT"

**-32091** "NSSQLE091 ** INVALID DATABASE NAME"

**-32092** "NSSQLE092 ** INVALID INTO CLAUSE WHEN ASYNCHRONOUS MODE"

**-32093** "NSSQLE093 ** INVALID LENGTH FOR DATABASE NAME"

**-32095** "NSSQLE095 ** INVALID LENGTH FOR USER NAME"

**-32096** "NSSQLE096 ** INVALID LENGTH FOR PASSWORD"

**-32097** "NSSQLE097 ** INVALID LENGTH FOR SERVER NAME"

**-32098** "NSSQLE098 ** INVALID LENGTH FOR SERVER NAME"

**-32099** "NSSQLE099 ** KEYWORD AT IS NOT SUPPORTED"

**-32101** "NSSQLE101 ** UNABLE TO OPEN FILE"

**-32102** "NSSQLE102 ** NO MEMORY AVAILABLE"

**-32103** "NSSQLE103 ** NO CONNECTION AVAILABLE TO UPDATE IMAGE/TEXT"

**-32200** "NSSQLE200 ** COMPUTE RESULT IN PROGRESS"

**See Also** ***NSnn_SQL Library Error* messages**

SQL_ERRMSG$, NS_FUNCTION ERRORCOUNT, NS_FUNCTION GETERROR, NS_FUNCTION CALLBACK

# SQL_ERRMSG$ function

Returns the error message (character string) for the last SQL_ instruction executed.

**Syntax** **SQL_ERRMSG$** *(error-code)*

**Parameter**

| | | | |
|---|---|---|---|
| *error-code* | INT(4) | I | error code |

**Returned value** CSTRING

**Notes**

1. SQL_ERRMSG$ returns the last message stored in a work area in the DLL when the error occurred.
2. See SQL_ERROR% for a detailed list of error codes and messages.

**Example**

See the example of the SQL_ERROR% instruction.

**See also** ***NSnn_SQL Library Error* messages**

# RECORD, REEXECUTE commands

The RECORD command records an SQL sequence so that it can be re-executed using the REEXECUTE command. When you call REEXECUTE, you only supply new values.

**Syntax** **RECORD** *SQL-statement*

and

**REEXECUTE**

| **Parameter** | | | |
|---|---|---|---|
| | *SQL-statement* | CSTRING | I | SQL sequence to record |

**Notes**

1. The parameters in the SQL sequence must still be accessible when the command 'SQL_EXEC REEXECUTE' is issued.
2. After RECORD, any other SQL statement other than REEXECUTE cancels the current RECORD operation.
3. Using RECORD and REEXECUTE allows you to combine the adaptability of dynamic SQL with the speed of static SQL. In fact, a dynamic SQL order is used when the RECORD command is executed. When the REEXECUTE command is executed, as the analysis of query has already been done by the motor, only the values of the host variables are set.

**Example**

```
LOCAL CODE%
LOCAL NAME$(25)

SQL_EXEC CREATE TABLE EMP(EMPNO INTEGER,ENAME CHAR(25))

CODE = 1
NOM$ = "NAME1"
SQL_EXEC RECORD INSERT INTO EMP VALUES (:CODE%,:NAME$)
FOR I= 2 TO 100
    CODE = I
    NAME$ = "NAME" & I
    SQL_EXEC REEXECUTE
ENDFOR

; ---- The EMP table now contains
; (  1, "NAME1")
; (  2, "NAME2")
; (  3, "NAME3")
; ...
; ( 99, "NAME99")
; (100, "NAME100")
```

# TYPE_SQL_SELECT_BLOB% type for blobs

Enables management of binary large objects, larger than 32K but whose size remains limited by the DBMS.

**Notes**

1. One new NCL data type has been added to NSDBMS.NCL and is to be declared in the Type_Var field of the NCLVAR structure:**TYPE_SQL_SELECT_BLOB%**. It must be used for retrieving a binary file from the database
2. Selection of images with TYPE_SQL_SELECT_BLOB% is limited by default to 1 Mb. However, this size may be changed using the NS_FUNCTION CHANGEOPTION TEXTSIZE instruction.
3. The cursor mode must be set on 3.

**Example**

```
LOCAL NCLVAR HL[2]
LOCAL INT IMAGNO
LOCAL DESCRIP$
LOCAL FIMAGE$
LOCAL INT J
LOCAL SQL$
LOCAL BMP%
LOCAL CURSORMODE%
LOCAL Opt$
LOCAL Val%, i%
LOCAL CSTRING Req$(2000)

CURSORMODE% = DB_ODBC_CURSOR_BINDING ;3
SQL_EXEC NS_FUNCTION SETCURSORMODE :CURSORMODE%
SQL_EXEC DROP TABLE BIGIMAGE
IF SQL_ERROR% <> 0
    MESSAGE "Error ",SQL_ERRMSG$(SQL_ERROR%)
 ENDIF
SQL_EXEC NS_FUNCTION Statement INTO :Req$
IF SQL_ERROR% <> 0
    MESSAGE "Error ",SQL_ERRMSG$(SQL_ERROR%)
 ENDIF
Insert AT END Req$ TO LISTBOX1
; Création de la table
SQL_EXEC CREATE TABLE BIGIMAGE(NUMBER INTEGER, DESCRIPTION VARCHAR(255),
COLIMAGE IMAGE)
IF SQL_ERROR% <> 0
    MESSAGE "Error ",SQL_ERRMSG$(SQL_ERROR%)
 ENDIF
SQL_EXEC NS_FUNCTION Statement INTO :Req$
Insert AT END Req$ TO LISTBOX1

FILL @HL, SIZEOF HL, 0
FIMAGE$ = F_IMAGE
HL[0].PTR_VAR = @FIMAGE$
HL[0].TYPE_VAR = TYPE_SQL_INSERT_BLOB%
HL[0].SIZE_VAR = SIZEOF FIMAGE$
```

```
SQL$="INSERT INTO BIGIMAGE (NUMBER, DESCRIPTION, COLIMAGE) VALUES \
 ( 1,'This is a big picture > 32000 bytes', : )"
SQL_EXEC_LONGSTR @SQL$, @HL, DEFAULT_CURSOR
IF SQL_ERROR% <> 0
    MESSAGE "Error ",SQL$&&SQL_ERRMSG$(SQL_ERROR%)
ENDIF
SQL_EXEC NS_FUNCTION Statement INTO :Req$
Insert AT END Req$ TO LISTBOX1

; ---- SELECT with automatic writing  in file EXTRACT.BMP
; ---- One & One column only in the SELECT clause , the IMAGE !!!!!
FILL @HL, SIZEOF HL, 0
FIMAGE$ = "C:\TEMP\EXTRACT.BMP"
FERASE FIMAGE$
HL[0].PTR_VAR = @FIMAGE$
HL[0].TYPE_VAR = TYPE_SQL_SELECT_BLOB%
HL[0].SIZE_VAR = SIZEOF FIMAGE$

SQL$="SELECT COLIMAGE INTO : FROM BIGIMAGE WHERE NUMBER = 1"
SQL_EXEC_LONGSTR @SQL$, @HL, DEFAULT_CURSOR

IF SQL_ERROR% <> 0
    MESSAGE "Error ",SQL$&&SQL_ERRMSG$(SQL_ERROR%)
 ENDIF
SQL_EXEC NS_FUNCTION Statement INTO :Req$
Insert AT END Req$ TO LISTBOX1

; ---- Display of the picture
BMP% = CREATEBMP%(FIMAGE$)
MOVE BMP% TO BMPF

CURSORMODE% = DB_ODBC_CURSOR_DEFAULT  ;0
SQL_EXEC NS_FUNCTION SETCURSORMODE :CURSORMODE%
```

**See also** NSDBMS.NCL, SQL_ERROR%, SQL_ERRMSG$, NS_FUNCTION IMAGEON, NS_FUNCTION IMAGEOFF, NS_FUNCTION SETCURSORMODE

# NS_FUNCTION extensions

The NS_FUNCTIONs correspond to new functionality developed by Nat System to extend database interface options.

The new functionalities can be accessed by the NCL language.

Some commands must be preceded compulsory by the keyword NS_FUNCTION :

**Syntax** SQL_EXEC **NS_FUNCTION** *command*

**Note**

*command* is one of the commands described further on.

**See also** SQL_ERROR%, SQL_ERRMSG$

# NS_FUNCTION CHANGEDBCNTX

Sets the current database.

☞ This function has been developed to manage several databases simultaneously.

**Syntax** **NS_FUNCTION CHANGEDBCNTX** *: logical-DBname*

**Parameter**

| | | | |
|---|---|---|---|
| *logical-DBname* | CSTRING | I | logical name of the current database |

**Notes**

1. The database specified in *logical-DBname* will become the current database.
2. If the specified database is invalid, the current database will not change.
3. If the SQL_OPENCURSOR command is called after NS_FUNCTION CHANGEDBCNTX, the new cursor will be associated with the database passed as an argument to this function.

**Example**

```
LOCAL LOGICALDBNAME$
SQL_OPEN "BASE1","SCOTT/TIGER@SERVICE1”
SQL_EXEC ....    ; BASE1 is the current database

SQL_OPEN "BASE2","SCOTT/TIGER@SERVICE2”
SQL_EXEC ....    ; BASE2 is the current database

LOGICALDBNAME$ = "BASE1"
SQL_EXEC NS_FUNCTION CHANGEDBCNTX :LOGICALDBNAME$
SQL_EXEC ....    ; BASE1 is the current database
SQL_CLOSE "BASE1"
SQL_EXEC ....    ; BASE2 is the current database

SQL_CLOSE "BASE2"
```

**See also** SQL_OPEN, SQL_CLOSE

# NS_FUNCTION IMAGEOFF, IMAGEON

IMAGEON mode enables binary object management (for example bitmaps) of 32 000 bytes maximum size. This manipulation has been executed in NCL segment SQL_IMAGE defined in NSDMS.NCL.

IMAGEOFF mode inactivates this function.

**Syntax** **NS_FUNCTION IMAGEOFF**

and

**NS_FUNCTION IMAGEON**

1. IMAGEOFF is the mode by default.
2. Binary objects are manipulated using an NCL segment SQL_IMAGE:

```
SEGMENT SQL_IMAGE
 INT REALSIZE(4) ; size of buffer allocated
 INT LENGTH%(4)  ; real size
                 ; (from SELECT)
  INT PTR%(4)    ; Buffer address
ENDSEGMENT
```

3. The maximum authorized size is 32K. If you want to handle BLOBs (large images) see TYPE_SQL_SELECT_BLOB%.
4. Images are not the only type of binary objects. Any type of binary file can be stored.
5. Binary storage is not cross-platform. Therefore, if you store binary files using Windows (ANSI) and you want to retrieve it using OS/2, you will probably have problems.

**Example**

```
;Create table
sql_exec create table BASE1 (ID INT, LONGSTR  VARCHAR(200) NULL,\
 COLIMAGE IMAGE NULL)
if sql_error% <> 0
  message 'error create' , sql_errmsg$(sql_error%)
endif

;Insertion

LOCAL DEST$(80),DATA%,SIZE%(4),NBREAD%(2),FILE%,NIL%,FNAME$, hbmp%
LOCAL SQL_IMAGE LOCALIMAGE
; ---- Changing mode
SQL_EXEC NS_FUNCTION IMAGEON
IF SQL_ERROR% <> 0
 MESSAGE "IMAGEON", SQL_ERROR% && SQL_ERRMSG$(SQL_ERROR%)
 RETURN 1
ENDIF
; ---- Reading the file and transferring to DATA%
FNAME$ = "(NS-BMP)\TINTIN.BMP"
HBMP%=CREATEBMP%(FNAME$)
BMPT = HBMP%
; FGETSIZE% doesn't interpret the environment variables
FNAME$ = "D:\TESTS\BMP\TINTIN.BMP"
```

```
SIZE%=FGETSIZE%(FNAME$) ; = 25000 in this example
INSERT AT END "SIZE "& SIZE% TO LISTBOX1
NEW SIZE%,DATA%
FILE%=F_OPEN%(1,FNAME$)
F_BLOCKREAD FILE%, DATA%, SIZE%, NBREAD%
IF F_ERROR%
 MESSAGE"ERROR", "Failed to load " & FNAME$ &"!"
 F_CLOSE FILE%
 DISPOSE DATA%
 RETURN 1
ENDIF
; ---- Insertion in the t_image table
LOCALIMAGE.REALSIZE = SIZE%
LOCALIMAGE.LENGTH% = SIZE%
LOCALIMAGE.PTR% = DATA%
SQL_EXEC INSERT INTO BASE1 (ID, LONGSTR, COLIMAGE) VALUES (10,'Une île entre \
le ciel et l ''eau', :LOCALIMAGE)
IF SQL_ERROR% <> 0
 MESSAGE "INSERT IMAGE", SQL_ERROR% && SQL_ERRMSG$(SQL_ERROR%)
 F_CLOSE FILE%
 DISPOSE DATA%
 RETURN 1
ENDIF
F_CLOSE FILE%
DISPOSE DATA%
;select
LOCAL DEST$(80),DATA%,SIZE%(4),NBREAD%(2),FILE%,NIL%,FNAME$, hbmp%
LOCAL SQL_IMAGE LOCALIMAGE
; ---- Changing mode
SQL_EXEC NS_FUNCTION IMAGEON
IF SQL_ERROR% <> 0
 MESSAGE "IMAGEON", SQL_ERROR% && SQL_ERRMSG$(SQL_ERROR%)
 RETURN 1
ENDIF

LOCALIMAGE.realsize = 30000
NEW LOCALIMAGE.realsize,LOCALIMAGE.PTR%
SQL_EXEC SELECT COLIMAGE INTO :LOCALIMAGE FROM BASE1 WHERE ID = 10
IF SQL_ERROR% <> 0
 MESSAGE "SELECT IMAGE",SQL_ERROR% && SQL_ERRMSG$(SQL_ERROR%)
ELSE
 ; ---- Displaying the image in the CTRLBMP bitmap control
 ; (here LOCALIMAGE.length% equals to 25000)
 FNAME$="(NS-BMP)\SOUVENIR.BMP"
 FILE%=F_CREATE%(1,FNAME$)
 INSERT AT END "FILE% "& FILE% TO LISTBOX1
 F_BLOCKWRITE FILE%, LOCALIMAGE.PTR%, LOCALIMAGE.REALSIZE, NBREAD%
 IF F_ERROR%
  MESSAGE"ERROR", "Failed to write " & FNAME$ &"!"
  F_CLOSE FILE%
  DISPOSE LOCALIMAGE.PTR%
  RETURN 1
 ENDIF
 HBMP%=CREATEBMP%(FNAME$)
 BMPF = HBMP%
 F_CLOSE FILE%
 DISPOSE LOCALIMAGE.PTR%
ENDIF
DISPOSE LOCALIMAGE.PTR%
; ---- return to the default mode
SQL_EXEC NS_FUNCTION IMAGEOFF
```

**See also** NSDBMS.NCL, SQL_ERROR%, SQL_ERRMSG$, TYPE_SQL_SELECT_BLOB%

# NS_FUNCTION PINGSERVER

Checks if the connection is still valid. If the connection is not valid anymore, the application must submit a new SQL_OPEN to reactivate it.

**Syntax** **NS_FUNCTION PINGSERVER INTO** :*png%*

| | | | | |
|---|---|---|---|---|
| **Parameter** | *png%* | INT(4) | I | status of the connection (valid or not) |

**Example**

```
local png%

SQL_EXEC NS_FUNCTION PINGSERVER INTO :png%
IF (sql_error%=0)
  IF (png%)
    message 'ping', 'server is alive'
  ELSE
    message 'ping', 'server is dead'
  ENDIF
ELSE
  message 'ping error', sql_error%&&sql_errmsg$(sql_error%)
ENDIF
```

# NS_FUNCTION GETSTATUS

Returns the MySQL status character string after the last executed request. (equivalent to what the mysqladmin executable returns).

**Syntax** **NS_FUNCTION GETSTATUS INTO** :*status$*

| | | | | |
|---|---|---|---|---|
| **Parameter** | *status$* | CSTRING | I | status of MySQL server |

**Example**

```
LOCAL status$
SQL_EXEC ns_function getstatus into :status$
IF (sql_error% = 0)
  MESSAGE 'mysql status', status$
ELSE
  MESSAGE 'status error', SQL_ERROR%&&SQL_ERRMSG$(SQL_ERROR%)
ENDIF
```

# NS_FUNCTION LISTDBS

Returns the list of existing databases on the server.

**Syntax** **NS_FUNCTION LISTDBS** [*:filter$*]

**Parameter**

| | | | |
|---|---|---|---|
| *filter$* | CSTRING | I | Optional parameter, contains a search string. Could include wild characters such as %. |

**Note**

**1.** If the *filter$* parameter is not defined, all databases are sent back.

**Example**

```
local filter$, db$

filter$ = 'My%' ;all the databasesbeginning with 'My'
SQL_EXEC ns_function listdbs :filter$
WHILE (sql_error%=0)
  SQL_EXEC fetch into :db$
  MESSAGE 'base', db$
ENDWHILE
```

# NS_FUNCTION LISTTABLES

Returns the tables's list of the current database.

**Syntax** **NS_FUNCTION LISTTABLES** [:*filter$*]

| **Parameter** | *filter$* | CSTRING | I | Optional parameter contains a search string. Could include wild characters such as %. |
|---|---|---|---|---|

**Note**

1. If the *filter$* parameter is not specified, all the tables of the database are returned.

**Example**

```
local filter$, table$

filter$ = 'T_%' ; all the tables beginning with 'T_'
SQL_EXEC ns_function listtables :filter$
WHILE (sql_error%=0)
  SQL_EXEC FETCH INTO :table$
  MESSAGE 'table', table$
ENDWHILE
```

# NS_FUNCTION LISTCOLUMNS

Returns the columns' list of the specified table.

**Syntax** **NS_FUNCTION LISTCOLUMNS** :*table$* [,:*filter$*]

**Parameters**

| | | | |
|---|---|---|---|
| *table$* | CSTRING | I | the table which columns list one wants to get. |
| *filter$* | CSTRING | I | optional parameter, contains a search string which could include wild characters. |

**Note**

1. If the *filter$* parameter is not specified, all the columns of the table are returned.

**Example**

```
local filter$, table$, col$

table$ = 'MyTable'
filter$  = 'C_%' ;all the columns beginning with 'C_'
SQL_EXEC ns_function listcolumns :table$, :filter$
WHILE (sql_error%=0)
  SQL_EXEC FETCH INTO :col$
  MESSAGE 'col', col$
ENDWHILE
```