Manuel d'utilisation
Interface commune d'accès aux bases de données

Table des matières

RESUME

Ce chapitre décrit les APIs communes à tous les drivers regroupés par catégories d'utilisation (chargement driver, gestion des erreurs, ...), les détails particuliers à un SGBD sont décrits dans leur chapitre respectif.

Ainsi, par exemple le mécanisme de gestion des erreurs est décrit de manière générale dans le premier chapitre et les messages d'erreurs sont décrits pour chaque driver SGBD dans leur chapitre respectif.

CATEGORIES FONCTIONNELLES

Accès à une base de données

Instruction SQL_OPEN

Ouvre une base de données.

Syntaxe	**SQL_OPEN** *nom-logique-base, chaîne-connexion*			
Paramètres	nom-logique-base	CSTRING	I	nom logique de la base de données à ouvrir
	chaîne-connexion	CSTRING	I	chaîne de connexion à une base de données

1. Le paramètre nom-logique-base correspond au nom logique de la base de données à ouvrir.
2. Le paramètre chaîne-connexion correspond à la chaîne de connexion à une base de données locale ou distante. Sa syntaxe est spécifique à chaque moteur, reportez vous au chapitre attaché au moteur que vous utilisez.

Reportez-vous au chapitre qui correspond au driver que vous utilisez pour les codes d'erreur récupérés par SQL_ERROR%.

```
; ---- Exemple ODBC2
; ---- DATASOURCE DSSYBASE connect
SQL_OPEN "BASE1", "usr1/pswd1@DSSYBASE"
IF SQL_ERROR% <> 0
MESSAGE "Erreur Base1", SQL_ERRMSG$(SQL_ERROR%)
ENDIF
...
; ---- du DATASOURCE DSSYBASE disconnect
SQL_CLOSE "BASE1"
```

***Voir aussi** SQL_CLOSE, AT, NS_FUNCTION CHANGEDBCNTX, SQL_ERROR%, SQL_ERRMSG$*

Instruction SQL_CLOSE

Ferme une base de données.

Syntaxe	**SQL_CLOSE** *nom-logique-base*			
Paramètres	nom-logique-base	CSTRING	I	nom logique de la base de données à fermer

Même s'il est recommandé de fermer la ou les bases ouvertes au sein de l'application, toute fermeture d'application par l'instruction SQL_STOP fait automatiquement les SQL_CLOSE des bases ouvertes.

```
; ---- Exemple ODBC2
; ---- DATASOURCE DSSYBASE connect
SQL_OPEN "BASE1", "usr1/pswd1@DSSYBASE"
IF SQL_ERROR% <> 0
MESSAGE "Erreur Base1", SQL_ERRMSG$(SQL_ERROR%)
ENDIF
...
; ---- du DATASOURCE DSSYBASE disconnect
SQL_CLOSE "BASE1"
```

***Voir aussi** SQL_OPEN, AT, NS_FUNCTION CHANGEDBCNTX, SQL_ERROR%, SQL_ERRMSG$*

Commande AT

Spécifie le nom de la base logique sur laquelle porte l'ordre SQL qui lui succède.

Syntaxe	**AT** *nom-logique-base ordre-SQL*			
Paramètres	nom-logique-base	CSTRING	I	nom logique de la base de données
	ordre-SQL	CSTRING	I	ordre SQL à exécuter

1. nom-logique-base a été précisée par le premier paramètre d'ouverture de la base : SQL_OPEN.
2. Dans le cas de l'ouverture simultanée de plusieurs bases, la première ouverte est celle considérée par défaut.
3. Pour passer d'une base de données à une autre, nous recommandons d'utiliser la commande NS_FUNCTION_CHANGEDBCNTX car la commande AT risque de ne plus être supportée dans les versions futures.

```
; ---- Exemple pour RDB
SQL_OPEN "BASE1" , "USR1/PWD1@NODENAME1#DECNET"
SQL_OPEN "CLASS2" , "USR1/PWD1@NODENAME1#DECNET!SERVERCLASS"
SQL_OPEN "BASE3" , "USR1/PWD1@NODENAME1"

SQL_EXEC SELECT... ; SELECT sur BASE3

SQL_EXEC AT CLASS2 SELECT... ; SELECT sur CLASS2
SQL_EXEC AT CLASS2 FETCH... ; FETCH sur CLASS2

SQL_EXEC FETCH... ; FETCH sur BASE3
```

Voir aussi ***SQL_OPEN***, ***SQL_CLOSE***, ***NS_FUNCTION_CHANGEDBCNTX***, ***SQL_ERROR%***, ***SQL_ERRMSG$***, ***SQL_EXEC***

NS_FUNCTION CHANGEDBCNTX

Permet de changer de base de données courante.

Syntaxe	**NS_FUNCTION CHANGEDBCNTX** *:nom-logique-base*			
Paramètre	nom-logique-base	CSTRING	I	nom logique de la base de données qui devient la base de données courante

1. La base de données spécifiée nom-logique-base devient la base de données courante.
2. Si la base de données précisée est invalide, on reste positionné sur la base en cours.
3. Si la commande SQL_OPENCURSOR% est appelée après NS_FUNCTION CHANGEDBCNTX, le curseur ouvert sera rattaché à la base choisie comme argument.

```
; ---- Exemple avec SYBASE 11
LOCAL LOGICALDBNAME$
; ---- Ouverture de la base logique BASE1 associée à une connexion sur ; le
serveur logique SERV1 avec la base physique B1
SQL_OPEN "BASE1","USR1/PSWD1@SERV1"
SQL_EXEC USE B1 ; Ordre SQL de SYBASE car BASE1 n'est pas
; le nom d'une base physique de SERV1
SQL_EXEC ... ; BASE1 est la base courante

; ---- Ouverture de la base logique pubs2
; associé à une connexion sur le serveur logique SERV1
; avec la base physique pubs2
SQL_OPEN "PUBS2/NOCURSOR","USR1/PSWD1@SERV1"
SQL_EXEC ... ; pubs2 est la base courante

LOGICALDBNAME$ = "BASE1"
SQL_EXEC NS_FUNCTION CHANGEDBCNTX :LOGICALDBNAME$
SQL_EXEC ... ; BASE1 est la base courante

SQL_CLOSE "BASE1"
SQL_EXEC ... ; pubs2 est la base courante

SQL_CLOSE "pubs2"
```

***Voir aussi** SQL_OPEN, SQL_CLOSE, AT, SQL_ERROR%, SQL_ERRMSG$, SQL_EXEC*

Initialiser un SGBD, terminer l'utilisation des SGBD

Fonction SQL_INITMULTIPLE%

Définit un SGBD et l'initialise.

Syntaxe	**SQL_INITMULTIPLE%** *(nom-SGBD)*			
Paramètre	nom-SGBD	CSTRING	I	nom de la librairie qui correspond au SGBD à utiliser.
Valeur retournée	INT(2)			

1. Elle doit être appelée en premier par toute application voulant utiliser un SGBD (elle provoque le chargement de la librairie).
2. Elle a le même effet que SQL_INIT mais peut être appelée plusieurs fois avec des noms de SGBD différents. Elle permet de travailler simultanément avec plusieurs SGBD.
3. Elle retourne un handle qui identifie de manière unique le SGDB.

Elle ne peut pas être utilisée dans un programme parallèlement à SQL_INIT. Il faut choisir le mode dans lequel vous désirez travailler (en mode multi-SGDB ou non), avant de démarrer l'écriture d'un programme. De la même manière, les fonctions SQL_STOPMULTIPLE, SQL_USE et SQL_GETUSED% ne peuvent être utilisées que dans un contexte multi-SGBD.

```
;Exemple
LOCAL HOR8%, HS11%

; ---- Chargement de la librairie d'accès à ORACLE 8
HOR8% = SQL_INITMULTIPLE% ("NSw2OR8")
IF SQL_ERROR% <> 0
MESSAGE "Erreur de chargement à ORACLE 8", SQL_ERRMSG$ (SQL_ERROR%)
ENDIF

LOCAL HOR8%, HS11%

; ---- Chargement de la librairie d'accès à ORACLE 8
HOR8% = SQL_INITMULTIPLE% ("NSw2OR8")
IF SQL_ERROR% <> 0
MESSAGE "Erreur de chargement à ORACLE 8", SQL_ERRMSG$ (SQL_ERROR%)
ENDIF

; ---- Chargement de la librairie d'accès à SYBASE 11(devient le driver
courant)
HS11% = SQL_INITMULTIPLE% ("NSw2S11")
IF SQL_ERROR% <> 0
MESSAGE "Erreur de chargement à SYBASE système 11", SQL_ERRMSG$
(SQL_ERROR%)
ENDIF
; ---- Connexion sur la base SYBAXE 11 de nom pubs2
```

```
SQL_OPEN "PUBS2","USR1/PWSD1@SERV1"

; ---- Changement de driver courant
SQL_USE HOR8%

; ---- Connexion à une base ORACLE 8 par le service COMPTA1
SQL_OPEN "BASE1", "SCOTT/TIGER@COMPTA1"

; ---- Décharge toutes les librairies chargées
SQL_STOPALL
```

Voir aussi SQL_INIT, SQL_STOP, SQL_STOPMULTIPLE, SQL_ERROR%, SQL_ERRMSG$

Instruction SQL_STOPALL

Arrête tous les SGBD initialisés.

Syntaxe	SQL_STOPALL

```
;Exemple
LOCAL HOR8%, HS11%
HOR8% = SQL_INITMULTIPLE% ("NSw2OR8")
...
HS11% = SQL_INITMULTIPLE% ("NSw2S11")
...
SQL_STOPALL
```

Voir aussi ***SQL_STOPMULTIPLE***, ***SQL_STOP***, ***SQL_ERROR%***, ***SQL_ERRMSG$***

Instruction SQL_STOPMULTIPLE

Termine l'utilisation d'un SGBD.

Syntaxe	**SQL_STOPMULTIPLE** *handle*			
Paramètre	handle	INT(2)	I	handle du SGBD à arrêter

Elle ne peut être utilisée que par des SGBD initialisés avec SQL_INITMULTIPLE%.

```
;Exemple
LOCAL HOR8%,HS11%
HOR8% = SQL_INITMULTIPLE% ('NSw2OR8')
HS11% = SQL_INITMULTIPLE% ('NSw2S11')
...
SQL_STOPMULTIPLE HOR8%
IF SQL_ERROR% <> 0
MESSAGE "Erreur de déchargement de la librairie ", SQL_ERRMSG$ (SQL_ERROR%)
ENDIF

SQL_STOPMULTIPLE HS11%
IF SQL_ERROR% <> 0
MESSAGE "Erreur de déchargement de la librairie ", SQL_ERRMSG$ (SQL_ERROR%)
ENDIF
```

***Voir aussi** SQL_INITMULTIPLE%, SQL_STOP, SQL_INIT, SQL_ERROR%, SQL_ERRMSG$*

Sélection du driver SGBD utilisé

Instruction SQL_INIT

Charge le driver permettant d'utiliser une version d'un SGBD donné.

Syntaxe	**SQL_INIT** *nom_DLL*			
Paramètre	nom_DLL	CSTRING	I	nom du driver à charger

1. Cette fonction doit être la première instruction SQL_ appelée par toute application voulant utiliser un moteur SGBD donné avec NCL.
2. Le paramètre nom-DLL doit contenir le nom de la DLL d'accès au SGBD donné.
3. Pour connaître le nom-DLL propre à la version du SGBD que vous utilisez, reportez au chapitre spécifique associé à ce moteur.

```
; ---- Exemple ORACLE 8.0
SQL_INIT "NSw2OR8" ; Charge le driver pour ORACLE 8.0
IF SQL_ERROR% <> 0
MESSAGE "Problème de chargement de DLL",SQL_ERROR% &&
SQL_ERRMSG$(SQL_ERROR%)
RETURN
ENDIF
...
SQL_STOP ; Décharge le driver
```

Voir aussi ***SQL_STOP***, ***SQL_INITMULTIPLE%***, ***SQL_STOPMULTIPLE***, ***SQL_STOPALL***, ***SQL_USE***, ***SQL_ERROR%***, ***SQL_ERRMSG$***

Instruction SQL_STOP

Décharge le driver SGBD courant en fermant toutes les connexions et les curseurs ouverts.

Syntaxe	**SQL_STOP**

Cette fonction doit être impérativement exécutée à la fin de l'application.

Reportez vous à l'exemple de l'instruction SQL_INIT.

Voir aussi ***SQL_INIT*****,** ***SQL_INITMULTIPLE%*** **,** ***SQL_STOPMULTIPLE*** **,** ***SQL_STOPALL*** **,** ***SQL_USE*****,** ***SQL_ERROR%*****,** ***SQL_ERRMSG$***

SGBD à utiliser : choisir, paramétrer et récupérer les caractéristiques

Fonction SQL_GETNAME$

Retourne le nom de la librairie correspondant au SGBD en cours d'utilisation.

Syntaxe	**SQL_GETNAME$**
Valeur retournée	CSTRING

```
;Exemple
LOCAL HOR8%, HS11%
; --- Chargement de la librairie NSw2OR8
HOR8% = SQL_INITMULTIPLE% ("NSw2OR8")

; --- Chargement de la librairie NSw2S11
HS11% = SQL_INITMULTIPLE% ("NSw2S11")
MESSAGE "Librairie qui correspond au SGBD en cours d'utilisation",
SQL_GETNAME$
; retourne NSw2S11

SQL_USE HINF%
; --- Changement de driver
MESSAGE "Librairie correspondant au SGBD en cours d'utilisation",
SQL_GETNAME$
; retourne NSw2OR8
```

Voir aussi ***SQL_PRODUCT$***, ***SQL_VERSION$***, ***SQL_USE***, ***SQL_ERROR%***, ***SQL_ERRMSG$***

Fonction SQL_GETUSED%

Retourne le handle du SGBD en cours d'utilisation.

Syntaxe	**SQL_GETUSED%**
Valeur retournée	INT(2)

```
;Exemple
LOCAL HOR8%, HS11%, H%
HOR8% = SQL_INITMULTIPLE% ("NSw2OR8")
HS11% = SQL_INITMULTIPLE% ("NSw2S11")

H% = SQL_GETUSED%
;la valeur de h% est la même que hS11%
SQL_STOPMULTIPLE H%
;Fermeture de NSw2S11
```

Voir aussi ***SQL_USE*****,** ***SQL_ERROR%*****,** ***SQL_ERRMSG$***

Fonction SQL_PRODUCT$

Retourne le nom complet du SGDB associé à la librairie en cours d'utilisation.

Syntaxe	**SQL_PRODUCT$**
Valeur retournée	CSTRING

Nous vous invitons à utiliser cette fonction et la fonction SQL_VERSION$ avant de contacter le support technique, puisque ces deux fonctions permettent d'identifier très précisément la version et le driver que vous utilisez.

```
;Exemple
LOCAL HOR8%, HS11%
HOR8% = SQL_INITMULTIPLE% ("NSw2OR8")
HS11% = SQL_INITMULTIPLE% ("NSw2S11")

MESSAGE "SGBD en cours d'utilisation", SQL_PRODUCT$
; retourne le nom de produit de NSw2S11

SQL_USE HOR8%
MESSAGE "SGBD en cours d'utilisation", SQL_PRODUCT$
; retourne le nom de produit de NSw2OR8
```

Voir aussi SQL_GETNAME$, SQL_VERSION$, SQL_ERROR%, SQL_ERRMSG$

Instruction SQL_USE

Permet de choisir le SGBD qui va être utilisé dans les ordres SQL qui suivent.

Syntaxe	**SQL_USE** *handle*			
Paramètre	handle%	INT(2)	I	handle d'un SGBD précédemment initialisé avec SQL_INITMULTIPLE%.

Les ordres SQL qui suivent, s'appliquent jusqu'à la prochaine instruction SQL_USE, au SGBD choisi.

```
;Exemple
LOCAL HOR8%, HS11%
HOR8% = SQL_INITMULTIPLE% ("NSw2OR8")
...
HS11% = SQL_INITMULTIPLE% ("NSw2S11")
SQL_USE HOR8%
SQL_OPEN "BASE1", ""
...
SQL_USE HS11%
SQL_OPEN "BASE2", ""
```

***Voir aussi** SQL_INITMULTIPLE%, SQL_GETUSED%, SQL_ERROR%, SQL_ERRMSG$*

Fonction SQL_VERSION$

Retourne la version de la librairie correspondant au SGBD en cours d'utilisation.

Syntaxe	**SQL_VERSION$**
Valeur retournée	CSTRING

Nous vous invitons à utiliser cette fonction et la fonction SQL_PRODUCT$ avant de contacter le support technique, puisque ces deux fonctions permettent d'identifier très précisément la version et le driver que vous utilisez.

```
;Exemple
LOCAL HOR8%
HOR8% = SQL_INITMULTIPLE% ("NSW2OR8")
MESSAGE "Version", SQL_VERSION$
; Retourne "ORACLE WINDOWS NT / DLL 2.0 / ORA8 PROD.9 / Sep 20 1996"
...
```

***Voir aussi** SQL_PRODUCT$, SQL_GETNAME$, SQL_ERROR%, SQL_ERRMSG$*

NS_FUNCTION ANSIOFF, ANSION

Dans le mode ANSIOFF, lorsqu'un appel à UPDATE ou DELETE n'affecte aucun enregistrement, aucune erreur n'est renvoyée.
Dans le mode ANSION, lorsqu'un appel à UPDATE ou DELETE n'affecte aucun enregistrement, une erreur (warning) de code 100 est renvoyée.

Syntaxe	**NS_FUNCTION** **ANSIOFF** et **NS_FUNCTION ANSION**

1. ANSIOFF est le mode par défaut.
2. SQL_ERROR% permet de récupérer le warning renvoyé.

```
;Exemple
; ---- Mode ANSIOFF par défaut
SQL_EXEC DELETE ... WHERE ...
; ---- Ici même si aucun enregistrement n'a été effacé SQL_ERROR% vaut
zéro.

; ---- Mode ANSION
SQL_EXEC NS_FUNCTION ANSION
SQL_EXEC UPDATE ... WHERE ...
IF SQL_ERROR% = 100
MESSAGE "Aucun enregistrement mis à jour",
SQL_ERROR% && SQL_ERRMSG$(SQL_ERROR%)
ENDIF

; ---- Retour au mode par défaut
SQL_EXEC NS_FUNCTION ANSIOFF
```

Voir aussi SQL_ERROR%, SQL_ERRMSG$

NS_FUNCTION TRIMCHAROFF, TRIMCHARON

Dans le mode TRIMCHARON, lors d'un SELECT les blancs en fin de chaîne sont supprimés. TRIMCHARON est applicable uniquement aux variables hôtes de type CSTRING ou STRING, mais pas aux variables de type CHAR ou VARCHAR2.
Le mode TRIMCHARON permet de limiter la taille du buffer sur le réseau.
Dans le mode TRIMCHAROFF les blancs en fin de chaîne sont conservés.

Syntaxe	**NS_FUNCTION** **TRIMCHAROFF** et **NS_FUNCTION TRIMCHARON**

TRIMCHAROFF est le mode par défaut.

```
;Exemple
LOCAL C$
SQL_EXEC CREATE TABLE T_DEMO(TEST CHAR(10))
SQL_EXEC INSERT INTO T_DEMO(TEST) VALUES ("A234567890")
SQL_EXEC INSERT INTO T_DEMO(TEST) VALUES ("A2345")
SQL_EXEC INSERT INTO T_DEMO(TEST) VALUES ("A")

; A ce stade, on est dans le mode par défaut
; ----- Cette boucle affichera <A234567890>
; <A2345 >
; <A >
SQL_EXEC SELECT * FROM T_DEMO
WHILE SQL_ERROR% <> 0
SQL_EXEC FETCH INTO :C$
MESSAGE "C$=<" & C$ & ">",""
ENDWHILE

; ----- Changement de mode
SQL_EXEC NS_FUNCTION TRIMCHARON

; ----- Cette boucle affichera <A234567890>
; <A2345>
; <A>
SQL_EXEC SELECT * FROM T_DEMO
WHILE SQL_ERROR% <> 0
SQL_EXEC FETCH INTO :C$
MESSAGE "C$=<" & C$ & ">",""
ENDWHILE

; ---- Retour au mode par défaut
SQL_EXEC NS_FUNCTION TRIMCHAROFF
```

Voir aussi ***SQL_ERROR%*****,** ***SQL_ERRMSG$***

Mécanismes pour exécuter un ordre SQL

Instruction SQL_EXEC

Exécute un ordre SQL.

Syntaxe	**SQL_EXEC** *[AT nom-logique-base] ordre-SQL [USING handle-curseur]*			
Paramètres	nom-logique-base	CSTRING	I	nom logique de la base de données
	ordre-SQL	CSTRING	I	ordre SQL à exécuter
	handle_curseur	INT(4)	I	valeur du curseur

1. L'ordre SQL est passé directement sans aucun guillemet. Il peut correspondre à n'importe quelle commande SQL, que cet ordre soit un ordre de définition de données (CREATE TABLE, CREATE INDEX, ...) ou de manipulation de données (SELECT, INSERT, UPDATE, ...).
2. La commande AT n'est utilisable qu'avec les bases de données autorisant plusieurs connections simultanées. La requête est envoyée à la base de données spécifiée derrière la commande AT (sans guillemets et case sensitif). Si la commande AT n'est pas spécifiée, alors la commande SQL_EXEC s'exécute sur la base de données courante.
3. Si USING handle_curseur est spécifié, alors il indique quel curseur préalablement ouvert par SQL_OPENCURSOR%, doit être utilisé pour exécuter l'ordre SQL. Si aucun curseur n'a été ouvert, alors la valeur du curseur est celle du DEFAULT_CURSOR : -1.
4. La commande SQL peut retourner des valeurs dans des variables NCL. Pour cela, il suffit de passer ces variables en paramètres.
5. Il est possible de passer un champ de segment comme variable réceptrice de donnée dans une requête SQL.

Les commandes SQL_EXEC, SQL_EXECSTR et SQL_EXEC_LONGSTR dépendent du langage SQL accepté par le SGBD utilisé (Cf. Manuels du fournisseur du SGBD utilisé).

6. Pour des commandes SQL trop longues, il est possible d'utiliser le caractère spécial de continuation "\" :

```
SQL_EXEC UPDATE SAMPLE SET SOCIETE =:A$ \
WHERE VILLE =:C$ AND \
PAYS =:D$
```

7. Les types de variables reconnus par l'interface sont :
   a) INT(1), INT(2), et INT(4),
   b) NUM(8), NUM(4),
   c) STRING,
   d) CSTRING,

e) CHAR.

8. Les types SQL reconnus sont propres à chaque SGBD, les manuels associés à chaque SGBD décrivent les conversions entre les types NCL et les types SQL autorisées.

9. La clause INTO est utilisé par les ordres SELECT et FETCH et permet de définir une liste de variables de réception. Sa syntaxe est la suivante : INTO :var1 [:indic1] [, :var2 [:indic2] [, ...]]

10. Il est préférable d'utiliser la clause INTO dans le SELECT pour améliorer les performances, car dans le FETCH le driver est obligé, à chaque passage dans la boucle, d'analyser les variables de la clause INTO. L'utilisation de la clause INTO dans le FETCH doit être par exemple réservée au remplissage des éléments d'un tableau.

Vous devez toujours faire précéder le nom d'une variable ou d'un indicateur de ":".

11. Un indicateur est une variable entière NCL qui peut prendre les valeurs suivantes : NULL_VALUE_INDICATOR (-1) -> cette valeur indique que la variable NCL associée qui le précède vaut NULL.

12. Toute autre valeur indique que la variable NCL associée (qui précède l'indicateur) vaut NOT NULL, et que sa valeur peut donc être prise en considération.

13. En langage SQL, NULL ne signifie pas 0 ou une chaîne vide (""). Toutefois, pour rendre possible l'affectation de valeurs dans tous les cas, si une colonne contient NULL, une variable réceptrice NCL de type numérique recevra 0 et une variable réceptrice NCL de type chaîne recevra une chaîne vide ("").

```
;Exemple
LOCAL CODE%,I%,AGE%,IND1%,IND2%
LOCAL COUNTRY$,CITY$,A$,B$
LOCAL TCODE%[10]
LOCAL TCOUNTRY$[10]

CITY$ = "PARIS"
; ========
; 1er CAS
; ========
; ---- Sélection du sous-ensemble
SQL_EXEC SELECT CODE,PAYS FROM MONDE WHERE VILLE = :CITY$
; ---- Lecture du 1er au dernier enregistrement de la sélection
WHILE SQL_ERROR% = 0
SQL_EXEC FETCH INTO :CODE%,:COUNTRY$
IF SQL_ERROR% = 0
INSERT AT END CODE% && COUNTRY$ TO LBOX1
ENDIF
ENDWHILE

; ============================
; 2ème CAS (le plus performant)
; ============================
; ---- Sélection du sous-ensemble
; et lecture du 1er enregistrement de la sélection
```

```
SQL_EXEC SELECT CODE,PAYS FROM MONDE INTO :CODE%,:COUNTRY$ \
WHERE VILLE = :CITY$
; ---- Lecture du 2ème au dernier enregistrement de la sélection
WHILE SQL_ERROR% = 0
INSERT AT END CODE% && COUNTRY$ TO LBOX1
SQL_EXEC FETCH
ENDWHILE

; =========
; 3ème CAS
; =========
; ---- Sélection du sous-ensemble
SQL_EXEC SELECT CODE,PAYS FROM MONDE WHERE VILLE = :CITY$
; ---- Lecture du 1er au dernier enregistrement de la sélection
; en remplissant les deux tableaux TCODE% et TCOUNTRY$
I% = 0
WHILE (SQL_ERROR% = 0) AND (I% < 10)
SQL_EXEC FETCH INTO :TCODE%[I%],:TCOUNTRY$[I%]
I% = I% + 1
ENDWHILE

; ============================
; Utilisation des indicateurs
; ============================
SQL_EXEC CREATE TABLE FAMILLE(NOM VARCHAR2(10),\
AGE NUMBER, \
NOMENFANT VARCHAR2(10))
PERE$ = "PAUL"
AGE% = 35
FILS$ = "PIERRE"
IND1% = 0
IND2% = 0

; --- Insère dans la table ("PAUL",35,"PIERRE")
SQL_EXEC INSERT INTO FAMILLE VALUES (:PERE$:IND1%, :AGE%, :FILS$:IND2%)
PERE$ = "PIERRE"
AGE% = 10
IND1% = 0
IND2% = NULL_VALUE_INDICATOR
; --- Insère dans la table ("PIERRE",10,NULL)
SQL_EXEC INSERT INTO FAMILLE VALUES (:PERE$:IND1%, :AGE%, :FILS$:IND2%)

; ---- La boucle SELECT affichage dans la LISTBOX LBOX
; 'Le fils de PAUL s'appelle PIERRE'
; 'PIERRE n'a pas de fils'
SQL_EXEC SELECT NOM, AGE, NOMENFANT INTO :PERE$:IND1%, :AGE%, :FILS$:IND2%
\
FROM FAMILLE

WHILE SQL_ERROR% = 0
; ---- IND1% vaut toujours 0 ici
IF IND2% = NULL_VALUE_INDICATOR
INSERT AT END PERE$ & " n'a pas de fils" TO LBOX
ELSE
INSERT AT END "Le fils de " & PERE$ & "s'appelle" & FILS$ TO LBOX
ENDIF
```

```
SQL_EXEC FETCH
ENDWHILE
```

Voir aussi SQL_EXECSTR, SQL_EXEC_LONGSTR, SQL_ERROR%, SQL_ERRMSG$

Instruction SQL_EXECSTR

Exécute un ordre SQL.

Syntaxe	**SQL_EXECSTR** *ordre-SQL [, variable , [variable , ...]] [USING handle-curseur]*			
Paramètres	ordre-SQL	CSTRING	I	ordre SQL à exécuter
	variable		I	liste de variables NCL
	handle_curseur	INT(4)	I	valeur du curseur

1. ordre-SQL est soit une variable host de type chaîne, soit une chaîne de caractères entre guillemets qui contient le texte de l'ordre SQL à exécuter.
2. Lorsque vous utilisez l'instruction SQL EXEC, les variables host sont situées directement dans le texte de la requête SQL. Lorsque vous utilisez l'instruction SQL_EXECSTR, les variables host sont des paramètres de l'instruction.
3. Lorsque vous utilisez l'instruction SQL_EXECSTR, l'emplacement de chaque variable host est repéré par un caractère ":" dans le texte de la requête. A chaque ":" correspond la variable host passée en paramètre au rang équivalent : le premier ":" correspond à la première variable host passée en paramètre, et ainsi de suite.
4. Les autres fonctionnalités de l'instruction SQL_EXECSTR sont identiques à celles de l'instruction SQL EXEC.

```
;Exemple
LOCAL REQ$, TABLE$, PERE$, FILS$
LOCAL AGE%, IND1%, IND2%, CURS1%

TABLE$ = "FAMILLE"
AGE% = 20
REQUETE$ = "SELECT NOM, AGE, NOMENFANT INTO : :,:,: : FROM '" &\
TABLE$ & "' WHERE AGE > :"

; ---- Ouvre un curseur
CURS1%=SQL_OPENCURSOR%

; ---- Sélection des personnes de plus de 20 ans
; dans la table FAMILLE
SQL_EXECSTR REQU$, PERE$, IND1%, AGE%, FILS$, IND2%, AGE% USING CURS1%

WHILE SQL_ERROR% = 0
IF IND2% = -1
INSERT AT END PERE$ & " n'a pas de fils" TO LBOX
ELSE
INSERT AT END "Le fils de " & PERE$ &"s'appelle" & FILS$ TO LBOX
ENDIF
SQL_EXEC FETCH USING CURS1%
ENDWHILE

; ---- Fermeture du curseur
SQL_CLOSECURSOR
```

Voir aussi ***SQL_EXEC*****,** ***SQL_EXEC_LONGSTR*****,** ***SQL_OPENCURSOR%*****,** ***SQL_CLOSECURSOR*****,** ***SQL_ERROR%*****,** ***SQL_ERRMSG$***

Instruction SQL_EXEC_LONGSTR

Exécute un ordre SQL.

Syntaxe	**SQL_EXEC_LONGSTR** *adresse-chaîne-sql, adresse-tableau-var, num-curseur*			
Paramètres	adresse-chaîne-sql	POINTER	I	adresse de la chaîne de caractères contenant l'ordre SQL à exécuter
	adresse-tableau-var	POINTER	I	adresse du tableau contenant les variables réceptrices (voir indicatrices)
	num-curseur	INT(2)	I	valeur du curseur

1. L'ordre exécuté peut correspondre à n'importe quel ordre SQL du langage hôte (DML ou DDL). La taille de la chaîne dépend du SGBDR utilisé ; elle est illimitée pour certains moteurs, et limitée à 4096 caractères pour d'autres.
2. adresse-chaîne-sql est l'adresse de la chaîne qui contient l'ordre SQL à exécuter.
3. adresse-tableau-var est un tableau de segments NCLVAR qui décrit les variables hôtes NCL. Si vous utilisez un ordre SQL qui n'utilise pas de variable hôtes, saisissez 0 dans adresse-tableau-var.
4. Lorsque vous utilisez l'instruction SQL_EXEC_LONGSTR, l'emplacement de chaque variable host est repéré par un caractère ":" dans le texte de la requête. A chaque ":" correspond la variable host au rang équivalent dans le tableau des variables réceptrices : le premier ":" correspond à la première variable du tableau, et ainsi de suite.
5. Le segment NCLVAR et les constantes utilisées sont déclarés dans le fichier librairie NSDBMS.NCL de la manière suivante :

```
SEGMENT NCLVAR
POINTER PTR_VAR
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
```

6. Ce tableau de segments doit contenir un indice de plus que le nombre de variables utilisées (le dernier élément contenant 0). C'est pourquoi il est conseillé de remplir préalablement ce tableau de segment par des 0 en utilisant l'instruction

NCL FILL. Cette solution garantit que l'élément 0 existe réellement, ainsi la fin du parcours des variables hôtes est connue.

7. Si aucun curseur n'a été ouvert, alors la valeur du curseur doit être celle du DEFAULT_CURSOR : -1.

8. L'instruction SQL_EXEC_LONGSTR annule et remplace l'ancienne fonction SQL_EXECLONGSTR%. Toutefois, vous trouverez dans les lignes de commentaires du fichier NSDBMS.NCL le code à saisir pour utiliser la fonction SQL_EXECLONGSTR%.

9. Les autres fonctionnalités de l'instruction SQL_EXEC_LONGSTR sont identiques à celles de l'instruction SQL_EXEC.

```
;Exemple
LOCAL NCLVAR VARLIST[3]  ; pour 2 variables
LOCAL SQL_STR$  ; chaîne à passer
LOCAL VAR1%, VAR2$  ; variables réceptrices
LOCAL CONDITION%  ; variable d'entrée

; ---- RAZ du tableau
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, -1

WHILE SQL_ERROR% = 0
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%

SQL_EXEC_LONGSTR @SQL_STR$, @VARLIST, -1
IF SQL_ERROR% = 0
MESSAGE "SELECT ", VAR1% && VAR2$
ENDIF
ENDWHILE
```

Voir aussi FILL (NCL), NSDBMS.NCL, SQL_EXEC, SQL_EXECSTR, SQL_ERROR%, SQL_ERRMSG$

Gestion des curseurs

On distingue deux catégories de curseurs :

1. Les premiers sont gérés comme des piles LIFO (Last In Firt Out) et utilisent les deux APIs suivantes :
   a) c1%=SQL_OPENCURSOR%
   b) SQL_CLOSECURSOR%
2. Les seconds autorisent une fermeture explicite d'un curseur donné et utilisent les deux APIs suivantes :
   a) c2%=SQL_OPENTHECURSOR%
   b) SQL_CLOSETHECURSOR(c2%)

Bien que les deux modes puissent coexister dans un même applicatif, SQL_ERROR% retournera une erreur si vous essayez d'exécuter les commandes suivantes :

- SQL_CLOSETHECURSOR alors que le curseur a été ouvert avec la commande SQL_OPENCURSOR%,
- SQL_CLOSECURSOR alors que le curseur a été ouvert avec la commande SQL_OPENTHECURSOR%.

Gestion en pile

Fonction SQL_OPENCURSOR%

Ouvre un curseur, et retourne son handle.

Syntaxe	**SQL_OPENCURSOR%**
Valeur retournée	INT(4)

1. Après ouverture, ce curseur peut ensuite être utilisé grâce à :

```
SQL_EXEC SELECT ... USING handle-curseur
SQL_EXEC FETCH ... USING handle-curseur
```

2. Un curseur est une ressource interne gérée par la DLL Nat System qui permet de conserver par exemple la position acquise sur une ligne de la table pour le prochain appel SQL.
3. A l'initialisation du système, un seul curseur est défini : il est appelé DEFAULT_CURSOR.
4. En l'absence d'ouverture de curseur, c'est avec DEFAULT_CURSOR que seront systématiquement exécutés les ordres SQL y compris les ordres SELECT et FETCH qui gèrent des positions courantes sur la base de données.
5. Un problème survient si un ordre SQL différent de FETCH (par exemple UPDATE ou INSERT) est intercalé dans une séquence de balayage, la position courante sera perdue et le FETCH suivant l'ordre intercalé se terminera en erreur.
6. SQL_OPENCURSOR% permet de résoudre ce problème car tous les ordres SELECT FETCH seront alors exécutés sur ce nouveau curseur.
7. D'une manière générale on aura intérêt à ouvrir un nouveau curseur chaque fois que l'on voudra effectuer un parcours SELECT FETCH alors qu'un autre parcours du même type est en cours et non terminé sur le dernier curseur ouvert.
8. La DLL Nat System spécifique au SGBD stocke les curseurs en pile LIFO (Last In First Out) avec SQL_OPENCURSOR% qui empile et SQL CLOSECURSOR qui dépile.
9. Les règles suivantes sont appliquées pour affecter l'exécution d'un ordre sur un curseur :
   a) Tous les ordres sont toujours exécutés sur le curseur précisé.
   b) Si avec SQL EXEC, la clause USING n'est pas précisé, alors les ordres sont exécutés sur le curseur DEFAULT_CURSOR.
10. Lors de l'ouverture simultanée de plusieurs bases, le curseur ouvert par SQL_OPENCURSOR% est immédiatement associé à la base courante.
11. Si vous désirez ouvrir un curseur sur une autre base que la base courante, vous devez exécuter la commande SQL EXEC CHANGEDBCNTX :autrebase$ pour changer de base courante, avant d'exécuter le SQL_OPENCURSOR%.

Reportez-vous à l'exemple de l'instruction SQL_CLOSETHECURSOR.

Voir aussi ***SQL_CLOSECURSOR, SQL_OPENTHECURSOR%, SQL_CLOSETHECURSOR, SQL_ERROR%, SQL_ERRMSG$***

Instruction SQL_CLOSECURSOR

Ferme le curseur le plus récemment ouvert.

Syntaxe	**SQL_CLOSECURSOR**

1. SQL_CLOSECURSOR ferme le dernier curseur ouvert qui est situé au sommet de la pile LIFO (Last In First Out) des curseurs.
2. SQL_CLOSECURSOR ne peut fermer que les curseurs ouverts par SQL_OPENCURSOR%.

> Reportez vous à l'exemple de l'instruction SQL_CLOSETHECURSOR.

Voir aussi ***SQL_OPENCURSOR%, SQL_OPENTHECURSOR%, SQL_CLOSETHECURSOR, SQL_ERROR%, SQL_ERRMSG$***

Gestion avec fermeture explicite d'un curseur donné

Fonction SQL_OPENTHECURSOR%

Ouvre un curseur, et retourne son handle.

Syntaxe	**SQL_OPENTHECURSOR%**
Valeur retournée	INT(2)

1. Après ouverture, ce curseur peut ensuite être utilisé grâce à :

```
SQL_EXEC SELECT ... USING handle-curseur
SQL_EXEC FETCH ... USING handle-curseur
```

2. Un curseur est une ressource interne gérée par la DLL Nat System qui permet de conserver par exemple la position acquise sur une ligne de table pour le prochain appel SQL.
3. A l'initialisation du système, un seul curseur est défini : il est appelé DEFAULT_CURSOR.
4. En l'absence d'ouverture de curseur, c'est avec DEFAULT_CURSOR que seront systématiquement exécutés les ordres SQL y compris les ordres SELECT et FETCH qui gèrent des positions courantes sur la base de données.
5. Un problème survient si un ordre SQL différent de FETCH (par exemple UPDATE ou INSERT) est intercalé dans une séquence de balayage, la position courante sera perdue et le FETCH suivant l'ordre intercalé se terminera en erreur.
6. SQL_OPENTHECURSOR% permet de résoudre ce problème car tous les ordres SELECT FETCH seront alors exécutés sur ce nouveau curseur.
7. D'une manière générale, on aura intérêt à ouvrir un nouveau curseur chaque fois que l'on voudra effectuer un parcours SELECT FETCH alors qu'un autre parcours du même type est en cours et non terminé sur le dernier curseur ouvert.
8. Les règles suivantes sont appliquées pour affecter l'exécution d'un ordre sur un curseur :
   a) Tous les ordres sont toujours exécutés sur le curseur précisé.
   b) Si avec SQL_EXEC, la clause USING n'est pas précisé, alors les ordres sont exécutés sur le curseur DEFAULT_CURSOR.
9. Lors de l'ouverture simultanée de plusieurs bases de données, le curseur ouvert par SQL_OPENTHECURSOR% est immédiatement associé à la base courante.
10. Si l'on veut ouvrir un curseur sur une autre base que la base courante, il faut faire un SQL_EXEC CHANGEDBCNTX :autrebase$ pour changer de base courante, avant d'exécuter le SQL_OPENTHECURSOR%.

Reportez vous à l'exemple de l'instruction SQL_CLOSETHECURSOR.

Voir aussi ***SQL_OPENCURSOR%*****,** ***SQL_CLOSECURSOR*****,** ***SQL_CLOSETHECURSOR*****,** ***SQL_ERROR%*****,** ***SQL_ERRMSG$***

Instruction SQL_CLOSETHECURSOR

Ferme le curseur associé à un handle donné.

Syntaxe	**SQL_CLOSETHECURSOR** *handle_curseur*			
Paramètre	handle_curseur	INT(4)	I	handle du curseur à fermer

SQL_CLOSETHECURSOR ne peut fermer que les curseurs ouverts par SQL_OPENTHECURSOR%.

```
; ---- Exemple complet montrant l'utilisation des 2 catégories
; de curseurs (pour clarifier cet exemple, le code pour
; tester les erreurs n'a pas été saisi)

SQL_EXEC .... ; utilise le curseur par défaut
C1% = SQL_OPENCURSOR% ; ouvre le curseur C1%
SQL_EXEC UPDATE ... ; utilise le curseur par défaut
SQL_EXEC SELECT ... ; utilise le curseur par défaut
SQL_CLOSETHECURSOR C1% ; => erreur
C2% = SQL_OPENTHECURSOR% ; ouvre le curseur C2%
SQL_EXEC UPDATE ... ; utilise le curseur par défaut
SQL_EXEC UPDATE ... USING C1% ; utilise le curseur C1%
SQL_EXEC SELECT ... USING C2% ; utilise le curseur C2%
SQL_EXEC SELECT ... USING C1% ; utilise le curseur C1%
SQL_CLOSECURSOR ; ferme le curseur C1%
SQL_EXEC UPDATE .... ; utilise le curseur par défaut
SQL_EXEC SELECT .... USING C2% ; utilise le curseur C2%
SQL_CLOSECURSOR% ; => erreur
SQL_CLOSETHECURSOR C2% ; ferme le curseur C2%
SQL_EXEC SELECT ... ; utilise le curseur par défaut
```

Voir aussi ***SQL_OPENCURSOR%***, ***SQL_CLOSECURSOR***, ***SQL_OPENTHECURSOR%***, ***SQL_ERROR%***, ***SQL_ERRMSG$***

Gestion des images et des binaires en général

NS_FUNCTION IMAGEOFF, IMAGEON

Le mode IMAGEON permet de manipuler des objets binaires (de type bitmaps par exemple).
Le mode IMAGEOFF désactive cette fonctionnalité.

Syntaxe	**NS_FUNCTION** **IMAGEOFF** et **NS_FUNCTION IMAGEON**

1. IMAGEOFF est le mode actif par défaut.
2. La manipulation d'objets binaires s'effectue par l'intermédiaire du segment NCL :

```
SEGMENT SQL_IMAGE
INT REALSIZE(4) ; Taille d'allocation du buffer
INT LENGTH%(4) ; Taille réellement lue
; (lors du SELECT)
INT PTR%(4) ; Adresse du buffer
ENDSEGMENT
```

3. La taille maximum autorisée est de 32000 octets. Si vous souhaitez gérer des BLOBs (de grosses images), reportez vous à la section qui décrit leur utilisation (types TYPE_SQL_INSERT_BLOB% et TYPE_SQL_SELECT_BLOB%).
4. Tous les objets binaires ne sont pas forcément des images, et donc n'importe quel binaire peut être stocké.
5. Le stockage de binaires n'est pas de type "cross platform".

```
; ATTENTION ! Nous fournissons ici un exemple pour ORACLE. Un exemple
propre à chaque SGBD est fourni dans leur documentation respective.
; ----------------------
; Evt INIT de la fenêtre
; ----------------------
GLOBAL HBMP%

; --------------------------
; Evt TERMINATE de la fenêtre
; --------------------------
DELETEBMP(HBMP%)

; --------------------------
; Evt autre de la fenêtre
; --------------------------
; ------------------------------------------------------------
; Lecture du fichier bitmap,
; Insertion de l'image dans la base de données,
; puis récupération de cet insertion depuis la base de données
; ------------------------------------------------------------
LOCAL DEST$(80),DATA%,SIZE%(4),NBREAD%(2),FILE%,NIL%,FNAME$
LOCAL SQL_IMAGE LOCALIMAGE
```

```
; ---- En ce point, le mode par défaut est IMAGEOFF

SQL_EXEC CREATE TABLE T_IMAGE(NUMERO NUMBER(8),\
DESCRIPTION VARCHAR2(80),\
IMAGE LONG RAW)

; ---- Changement de mode
SQL_EXEC NS_FUNCTION IMAGEON

; ---- Lecture du fichier et transfert dans DATA%
FNAME$ = "C  :\WINDOWS\MARQUISE.BMP"
SIZE%=FGETSIZE%(FNAME$) ; = 25000 dans cet exemple
NEW SIZE%,DATA%,"
FILE%=F_OPEN%(1,FNAME$)
F_BLOCKREAD FILE%, DATA%, SIZE%, NBREAD%
IF F_ERROR%
MESSAGE "ERROR", "Failed to load " & FNAME$ &"!"
F_CLOSE FILE%
DISPOSE DATA%
RETURN 1
ENDIF

; ---- Insertion dans la table t_image
LOCALIMAGE.REALSIZE% = SIZE%
LOCALIMAGE.LENGTH% = SIZE%
LOCALIMAGE.PTR% = DATA%
SQL_EXEC INSERT TO T_IMAGE\
VALUES (1,"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%

; ---- Récupération de la bitmap dans la base de données
; On doit allouer une taille plus grande car à priori on
; ne peut pas connaître à l'avance la taille de l'image que l'on va
sélectionné.
LOCALIMAGE.REALSIZE% = 30000
NEW LOCALIMAGE.REALSIZE%,LOCALIMAGE.PTR%,"
SQL_EXEC SELECT IMAGE INTO :LOCALIMAGE\
FROM T_IMAGE\
WHERE NUMERO = 1
IF SQL_ERROR% <> 0
MESSAGE "SELECT IMAGE",SQL_ERROR% && SQL_ERRMSG$(SQL_ERROR%)
ELSE
; ---- Affichage de l'image dans le contrôle bitmap CTRLBMP
; (ici LOCALIMAGE.length% vaut 25000 dans notre cas)
FNAME$="C:\WINDOWS\SOUVENIR.BMP"
FILE%=F_CREATE%(1,FNAME$)
F_BLOCKWRITE FILE%,\
LOCALIMAGE.PTR%,\
LOCALIMAGE.REALSIZE,\
```

```
LOCALIMAGE.LENGTH%
IF F_ERROR%
MESSAGE"ERROR" ,"Erreur lors de l'écriture" & FNAME$ &"!"
F_CLOSE FILE%
DISPOSE LOCALIMAGE.PTR%
RETURN 1
ENDIF
HBMP%=CREATEBMP%(FNAME$)
CRTL = HBMP%
F_CLOSE_FILE%
DISPOSE LOCALIMAGE.PTR%
ENDIF
DISPOSE LOCALIMAGE.PTR%

; ---- Retour au mode par défaut
SQL_EXEC NS_FUNCTION IMAGEOFF
```

Voir aussi NSDBMS.NCL, SQL_ERROR%, SQL_ERRMSG$, TYPE_SQL_INSERT_BLOB%, TYPE_SQL_SELECT_BLOB%

Récupérer la durée d'exécution d'un ordre SQL

Fonction SQL_GETTIME%

Retourne la durée en millisecondes de la dernière commande SQL exécutée.

Syntaxe	**SQL_GETTIME%**
Valeur retournée	INT(4)

```
;Exemple
LOCAL HOR8%
HOR8% = SQL_INITMULTIPLE% ("NSw2OR8")

SQL_OPEN "base", ""
MESSAGE "Open a duré", SQL_GETTIME% && "ms"

SQL_EXEC SELECT ...
MESSAGE "le select a duré", SQL_GETTIME% && "ms"
```

***Voir aussi** SQL_STARTTIMER, SQL_STOPTIMER, SQL_GETTIMER%, SQL_ERROR%, SQL_ERRMSG$*

Instruction SQL_STARTTIMER

Démarre le chronomètre.

Syntaxe	SQL_STARTTIMER

Reportez-vous à l'exemple de l'instruction SQL_GETTIMER%.

***Voir aussi** SQL_STOPTIMER, SQL_GETTIMER%, SQL_ERROR%, SQL_ERRMSG$*

Instruction SQL_STOPTIMER

Arrête le chronomètre.

Syntaxe	SQL_STOPTIMER

Reportez-vous à l'exemple de l'instruction SQL_GETTIMER%.

Voir aussi ***SQL_STARTTIMER*****,** ***SQL_GETTIMER%*****,** ***SQL_ERROR%*****,** ***SQL_ERRMSG$***

Fonction SQL_GETTIMER%

Retourne suivant la valeur de son paramètre le temps SQL, le temps NCL ou le temps global (SQL + NCL) consacré à l'exécution des ordres situés entre l'instruction STARTTIMER et STOPTIMER.

Syntaxe	**SQL_GETTIMER%** *(timertype)*			
Paramètre	timertype	INT(2)	I	type du TIMER
Valeur retournée	INT(4)			

1. Timertype peut prendre les 3 valeurs suivantes :
   a) CONST SQL_TIME
   b) CONST NCL_TIME
   c) CONST GLOB_TIME
2. Si timertype vaut SQL_TIME retourne le temps SQL uniquement.
3. Si timertype vaut NCL_TIME retourne le temps NCL uniquement.
4. Si timertype vaut GLOB_TIME retourne le temps SQL + le temps NCL.

```
;Exemple
...
SQL_STARTTIMER
SQL_EXEC SELECT COL1,COL2 INTO  :A$,  :B$ FROM TABLE1
WHILE SQL_ERROR% = 0
SQL_EXEC FETCH
INSERT AT END A$,B$ TO LBOX
ENDWHILE
SQL_STOPTIMER
INSERT AT END "TEMPS SQL =" & SQL_GETTIMER%(SQL_TIME) TO LBOX
INSERT AT END "TEMPS NCL =" & SQL_GETTIMER%(NCL_TIME) TO LBOX
INSERT AT END "TEMPS TOTAL=" & SQL_GETTIMER%(GLOB_TIME) TO LBOX
...
```

Voir aussi ***SQL_STARTTIMER*****,** ***SQL_STOPTIMER*****,** ***SQL_ERROR%*****,** ***SQL_ERRMSG$***

Gérer la trace

Instruction SQL_LOGGINGOFF

Arrête la trace en cours.

Syntaxe	SQL_LOGGINGOFF

```
;Exemple
SQL_LOGGINGON "C:\MATRACE.TRC"

SQL_INIT "NSw2S11"
SQL_OPEN "MABASE", ""

SQL_LOGGINGOFF
```

Voir aussi ***SQL_LOGGINGON*****,** ***SQL_ERROR%*****,** ***SQL_ERRMSG$***

Instruction SQL_LOGGINGON

Démarre la trace automatique de toutes les requêtes exécutées dans une application.

Syntaxe	**SQL_LOGGINGON** *nom-fichier-trace*			
Paramètre	nom-fichier-trace	CSTRING	I	nom du fichier dans lequel la trace sera sauvegardée

1. Si on veut tout tracer, cette instruction doit être la première,
2. Le fichier sera réinitialisé lors de l'exécution de la fonction s'il existe, créé sinon.
3. La trace se poursuit jusqu'à la rencontre de l'ordre SQL_LOGGINGOFF (ou de la fin d'exécution).

```
;Exemple
SQL_LOGGINGON "C:\MATRACE.TRC"

SQL_INIT "NSw2DB26"
SQL_OPEN "MABASE", ""

SQL_LOGGINGOFF
```

Voir aussi SQL_LOGGINGOFF, SQL_ERROR%, SQL_ERRMSG$

Gestion des erreurs

Mode de gestion non centralisé des erreurs

Fonction SQL_ERROR%

Retourne le code d'erreur de la dernière instruction SQL_ effectuée.

Syntaxe	**SQL_ERROR%**
Valeur retournée	INT(4)

1. SQL_ERROR% respecte la norme SQL, la fonction retourne donc :
   a) 0 s'il n'y a eu aucune erreur,
   b) un nombre positif pour toute erreur non fatale (l'instruction a été exécutée, mais émet un Warning),
   c) un nombre négatif pour toute erreur fatale (l'instruction n'a pu être exécutée).
2. Cette fonction est applicable à toutes les fonctions des drivers pour le SGBD.
3. Deux types de codes retour sont renvoyés par l'interface :
   a) Les codes retours SQL spécifiques au SGBD qui sont décrits dans le manuel du fournisseur.
   b) Les codes retours Nat System spécifiques à l'interface. Ils correspondent aux codes d'erreurs inutilisés par le SGBD hôte. Leur numérotation est signée et elle est de la forme "32XXX".

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

4. Une description des erreurs propres à chaque driver se trouve dans les chapitres précédents cette annexe :

```
;Exemple
MOVE "SAMPLE" TO B$
SQL_OPEN B$, ""
IF SQL_ERROR% < 0
MESSAGE "Erreur fatale sur" && B$, SQL_ERRMSG$(SQL_ERROR%)
MESSAGE "Danger !", "Application arrêtée"
RETURN
ELSEIF SQL_ERROR% > 0
MESSAGE "Warning sur" && B$, SQL_ERRMSG$(SQL_ERROR%)
ELSE
MESSAGE "OK", "Base" && B$ && "ouverte"
ENDIF
```

Voir aussi ***SQL_ERRMSG$***, ***NS_FUNCTION ERRORCOUNT***, ***NS_FUNCTION GETERROR***, ***NS_FUNCTION CALLBACK***

Fonction SQL_ERRMSG$

Retourne le message d'erreur (chaîne de caractères) associé à l'erreur qui s'est produite lors de la dernière instruction SQL effectuée.

Syntaxe	**SQL_ERRMSG$** *(code-erreur)*			
Paramètre	code_erreur	INT(4)	I	code d'erreur
Valeur retournée	CSTRING			

1. SQL_ERRMSG$ renvoie le message stocké dans une zone de travail de la DLL lors de l'occurrence de l'erreur et qui a pour code code-erreur.
2. Une description des erreurs propre à chaque driver se trouve dans leur documentation associée.

Reportez-vous à l'exemple de l'instruction SQL_ERROR%.

Voir aussi ***SQL_ERROR%, NS_FUNCTION_ERRORCOUNT, NS_FUNCTION_GETERROR, NS_FUNCTION CALLBACK***

NS_FUNCTION ERRORCOUNT

Récupère le nombre d'erreurs ou de messages d'erreurs rencontrés lors de l'exécution d'une requête. La numérotation des messages commence à 0.

Syntaxe	**NS_FUNCTION ERRORCOUNT INTO** *:nombre-erreurs*			
Paramètre	nombre-erreurs	INT(4)	O	nombre d'erreurs ou de messages d'erreurs rencontrés lors de l'exécution d'une requête

```
;Exemple
LOCAL NBERROR%
SQL_EXEC NS_FUNCTION ERRORCOUNT INTO :NBERROR%
MESSAGE "NOMBRE D'ERREUR",NBERROR%
```

Voir aussi ***NS_FUNCTION GETERROR*****,** ***SQL_ERROR%*****,** ***SQL_ERRMSG$*****,** ***NS_FUNCTION CALLBACK***

NS_FUNCTION GETERROR

Récupère le numéro d'erreur indiqué par son occurrence. La numérotation des erreurs est comprise entre 0 et la valeur retournée par NS_FUNCTION ERRORCOUNT moins 1.

Syntaxe	**NS_FUNCTION GETERROR** *:index_erreur INTO :numéro_erreur*			
Paramètres	index_erreur	INT(4)	I	Index du numéro d'erreur
	numéro_erreur	INT(4)	O	Numéro de l'erreur

```
;Exemple
LOCAL I%, ROW_COUNT%, ERROR%

MOVE 0 TO ROW_COUNT%

SQL_EXEC NS_FUNCTION ERRORCOUNT INTO :ROW_COUNT%
;Récupère le nombre d'erreurs dans ROW_COUNT%
IF ROW_COUNT% <> 0
 MOVE 0 TO I%
 WHILE i% < ROW_COUNT%
  SQL_EXEC NS_FUNCTION GETERROR :i% INTO :ERROR%
   ;Récupère pour chaque erreur son numéro dans ERROR%
  MESSAGE "ERROR" && I%, SQL_ERRMSG$(ERROR%)
  I% = I% + 1
 ENDWHILE
ENDIF
```

Voir aussi ***NS_FUNCTION ERRORCOUNT***, ***SQL_ERROR%***, ***SQL_ERRMSG$***, ***NS_FUNCTION CALLBACK***

Mode de gestion centralisé des erreurs

Ce mode de gestion des erreurs permet une gestion plus fine et plus puissante des erreurs, que celle effectuée par SQL_ERROR%, SQL_ERRMSG$, NS_FUNCTION ERRORCOUNT et NS_FUNCTION GETERROR.

NS_FUNCTION CALLBACK

Permet d'implémenter une gestion centralisée des erreurs à un seul endroit dans un applicatif, on n'est plus obligé d'appeler SQL_ERROR% et SQL_ERRMSG$ après chaque appel à un ordre.

Syntaxe	**NS_FUNCTION CALLBACK** *:handle-fenêtre, :evt-utilisateur*			
Paramètres	handle-fenêtre	INT(4)	I	handle de fenêtre
	evt-utilisateur	INT(4)	I	événement utilisateur (USER0 à USER15)

1. Sous UNIX, handle-fenêtre doit prendre comme valeur le handle Nat System de la fenêtre qui doit recevoir les notifications en cas d'erreur.
2. Pour toutes les autres cibles, handle-fenêtre doit être renseigné par la fonction NCL GETCLIENTHWND%(...) laquelle reçoit en entrée le handle Nat System de la fenêtre qui doit recevoir les notifications en cas d'erreur.
3. Pour déterminer le traitement, vous devez programmer l'événement utilisateur. Pour obtenir une notification de l'événement dans evt-utilisateur mettre : 0 pour USER0, 1 pour USER1,.... ou 15 pour USER15.
4. Pour annuler cet ordre, il suffit de passer un handle de fenêtre égal à zéro.
5. Par ailleurs, les erreurs ou les avertissements natifs du moteur sur lequel on se trouve sont retournés dans une structure propre à chaque moteur (reportez vous au fichier NSDBMS.NCL pour le détail de chacune des structures).
6. Le type de message d'erreur est envoyé dans PARAM12%. Le handle de la structure est envoyé dans PARAM34%.

```
; Montre un exemple d'activation et de désactivation du mode de
; gestion des erreurs centralisées, avec un exemple de description du ;
code NCL à placer
; dans l'événement USER1 de la fenêtre CATCHERR.
; ---------------------------------------
; Dans l'événement INIT de la fenêtre MAIN
; ---------------------------------------
GLOBAL HDLE_CATCHERR%

; ---------------------------------------------
; Dans l'événement TERMINATE de la fenêtre MAIN
; ---------------------------------------------
IF HDLE_CATCHERR% <> 0
CLOSE HDLE_CATCHERR%
HDLE_CATCHERR%=0
ENDIF

; -------------------------------------------------
; Dans l'événement SELECTED du checkbox CKCALLBACK
; (champs input à 0) de la fenêtre MAIN
```

```
; -----------------------------------------------
LOCAL USER_EVENT%
LOCAL WINDOW_HANDLE%

IF CKCALLBACK = CHECKED%
; ---- Active le mode de gestion d'erreur
; centralisée par CALLBACK
IF HDLE_CATCHERR% = 0
OPEN CATCHERR,0,HDLE_CATCHERR% ; fenêtre où seront
; centralisées les erreurs
ENDIF
MOVE GETCLIENTHWND%(HDLE_CATCHERR%) TO WINDOW_HANDLE%
MOVE 1 TO USER_EVENT%
SQL_EXEC NS_FUNCTION CALLBACK :WINDOW_HANDLE% , :USER_EVENT%
MESSAGE "START CALLBACK" && WINDOW_HANDLE% && USER_EVENT%", SQL_ERROR% &&
SQL_ERRMSG$(SQL_ERROR%)
ELSE
; ---- Désactive le mode de gestion d'erreur
; centralisée par CALLBACK
MOVE 0 TO WINDOW_HANDLE%
MOVE 1 TO USER_EVENT%
SQL_EXEC NS_FUNCTION CALLBACK :WINDOW_HANDLE% , :USER_EVENT%
MESSAGE "STOP CALLBACK" && WINDOW_HANDLE% && USER_EVENT%", SQL_ERROR% &&
SQL_ERRMSG$(SQL_ERROR%)
IF HDLE_CATCHERR% <> 0
CLOSE HDLE_CATCHERR%
HDLE_CATCHERR%=0
ENDIF
ENDIF

; -----------------------------------------------
; Dans l'événement USER1 de la fenêtre CATCHERR
; -----------------------------------------------
; ATTENTION ! Nous fournissons ici un exemple pour SYBASE 11. Chaque SGBD
ayant
; chacun sa propre structure interne de gestion d'erreur, un exemple propre
à
; chaque SGBD est fournit dans leur documentation respective.

; ---- Définitions contenues dans NSDBMS.NCL
; CONST CLIENTMSG 1
; CONST SERVERMSG 2
; SEGMENT DB_SYBASE_CLIENT_STRUCT
; INT SEVERITY(4)
; INT MSGNUMBER(4)
; CHAR MSGSTRING(1024)
; INT MSGSTRINGLEN(4)
; INT OSNUMBER(4)
; CHAR OSSTRING(1024)
; INT OSSTRINGLEN(4)
; ENDSEGMENT
; SEGMENT DB_SYBASE_SERVER_STRUCT
; INT MSGNUMBER(4)
; INT STATE(4)
; INT SEVERITY(4)
; CHAR TEXT(1024)
; INT TEXTLEN(4)
```

```
; CSTRING SVRNAME(131)
; INT SVRNLEN(4)
; CSTRING PROC(131)
; INT PROCLEN(4)
; INT LINE(4)
; INT STATUS(4)
; CSTRING SQLSTATE(7)
; INT SQLSTATELEN(4)
; ENDSEGMENT

LOCAL MESSAGETYPE%, PTR%

MESSAGETYPE% = PARAM1% ; Message type CLIENT ou SERVEUR
PTR% = PARAM34% ; contient un pointeur sur une structure de type erreur
EVALUATE MESSAGETYPE%
WHERE CLIENTMSG
INSERT AT END "CLIENT" TO LISTBOX
INSERT AT END "" TO LISTBOX
INSERT AT END "SEVERITY " & DB_SYBASE_CLIENT_STRUCT(PTR%).SEVERITY TO
LISTBOX
INSERT AT END "MSGNUMBER " & DB_SYBASE_CLIENT_STRUCT(PTR%).MSGNUMBER TO
LISTBOX
IF (DB_SYBASE_CLIENT_STRUCT(PTR%).MSGSTRINGLEN > 0)
INSERT AT END "MSGSTRING " & DB_SYBASE_CLIENT_STRUCT(PTR%).MSGSTRING TO
LISTBOX
ENDIF
INSERT AT END "OSNUMBER " & DB_SYBASE_CLIENT_STRUCT(PTR%).OSNUMBER TO
LISTBOX
IF (DB_SYBASE_CLIENT_STRUCT(PTR%).OSSTRINGLEN > 0)
INSERT AT END "OSSTRING " & DB_SYBASE_CLIENT_STRUCT(PTR%).OSSTRING TO
LISTBOX
ENDIF
INSERT AT END "OSSTRINGLEN " & DB_SYBASE_CLIENT_STRUCT(PTR%).OSSTRINGLEN TO
LISTBOX
ENDWHERE
WHERE SERVERMSG
INSERT AT END "SERVER" TO LISTBOX
INSERT AT END "" TO LISTBOX
INSERT AT END "MSGNUMBER " & DB_SYBASE_SERVER_STRUCT(PTR%).MSGNUMBER TO
LISTBOX
INSERT AT END "STATE " & DB_SYBASE_SERVER_STRUCT(PTR%).STATE TO LISTBOX
INSERT AT END "SEVERITY " & DB_SYBASE_SERVER_STRUCT(PTR%).SEVERITY TO
LISTBOX
IF (DB_SYBASE_SERVER_STRUCT(PTR%).TEXTLEN > 0)
INSERT AT END "TEXT " & DB_SYBASE_SERVER_STRUCT(PTR%).TEXT TO LISTBOX
ENDIF
INSERT AT END "TEXTLEN " & DB_SYBASE_SERVER_STRUCT(PTR%).TEXTLEN TO LISTBOX
IF (DB_SYBASE_SERVER_STRUCT(PTR%).SVRNLEN > 0)
INSERT AT END "SVRNAME " & DB_SYBASE_SERVER_STRUCT(PTR%).SVRNAME TO LISTBOX
ENDIF
INSERT AT END "SVRNLEN " & DB_SYBASE_SERVER_STRUCT(PTR%).SVRNLEN TO LISTBOX
IF (DB_SYBASE_SERVER_STRUCT(PTR%).PROCLEN > 0)
INSERT AT END "PROC " & DB_SYBASE_SERVER_STRUCT(PTR%).PROC TO LISTBOX
ENDIF
INSERT AT END "PROCLEN " & DB_SYBASE_SERVER_STRUCT(PTR%).PROCLEN TO LISTBOX
INSERT AT END "LINE " & DB_SYBASE_SERVER_STRUCT(PTR%).LINE TO LISTBOX
INSERT AT END "STATUS " & DB_SYBASE_SERVER_STRUCT(PTR%).STATUS TO LISTBOX
```

```
IF (DB_SYBASE_SERVER_STRUCT(PTR%).SQLSTATELEN > 0)
INSERT AT END "SQLSTATE " & DB_SYBASE_SERVER_STRUCT(PTR%).SQLSTATE TO
LISTBOX
ENDIF
INSERT AT END "SQLSTATELEN " & DB_SYBASE_SERVER_STRUCT(PTR%).SQLSTATELEN TO
LISTBOX
ENDWHERE
ENDEVALUATE
```

Voir aussi NSDBMS.NCL, SQL_ERROR%, SQL_ERRMSG$, NS_FUNCTION ERRORCOUNT, NS_FUNCTION GETERROR

Codes d'erreur

Ces codes d'erreur sont internes à la librairie NSnn_SQL.LIB.

Codes	Messages d'erreur	Cause
+100	** NO ROW WAS FOUND OR LAST ROW REACHED	Fin de séquence de recherche effectuée par l'ordre FETCH
-201	** OUT OF MEMORY	Mémoire insuffisante
-202	** FILE NOT FOUND	Fichier non trouvé lors d'un SQL_INIT(MULTIPLE%)
-203	** INVALID FILE	Fichier non valide lors d'un SQL_INIT(MULTIPLE%)
-204	** INIT ERROR	Erreur d'initialisation lors d'un SQL_INIT(MULTIPLE%)
-205	** SQL_INIT WAS ALREADY USED	Appel à SQL_INIT plus d'une fois avec des DLL différentes
-206	** LIBRARY ALREADY LOADED	Librairie déjà chargée lors d'un SQL_INIT(MULTIPLE%)
-207	** TOO MANY LIBRARIES OPENED	Le nombre maximum de librairies ouvertes simultanément est atteint
-208	** CAN'T USE SQL_INITMULTIPLE%, USE SQL_INIT	Utilisation de SQL_INITMULTIPLE% en mode single
-209	** CAN'T USE SQL_INIT, USE SQL_INITMULTIPLE%	Utilisation de SQL_INIT en mode multiple
-210	** USE SQL_STOP BEFORE ANOTHER SQL_INIT	Appel à SQL_INIT plus d'une fois avec des DLL différentes
-211	** INVALID HANDLE	Utilisation d'un handle invalide
-212	** LIBRARY NOT LOADED	Utilisation d'une fonction SQL_.. avant un SQL_INIT(MULTIPLE%)
-213	** STOP_DATABASE ERROR. LIBRARY MAY NOT BE UNLOADED	Appel à SQL_STOP avant SQL_INIT
-214	** PATH NOT FOUND	Chemin de recherche de la librairie introuvable (lors d'un init)
-215	** TOO MANY OPENED FILES	Trop de fichiers ouverts en même temps (lors d'un SQL_LOGGINGON)
-216	** CAN'T ACCESS FILE	Fichier inaccessible (par exemple, tentative d'écriture d'un log sur un fichier protégé)

-217	** INVALID FILE NAME	Nom de fichier invalide
-218	** NOT A DOS DISK	Tentative d'accès à un disque non DOS
-219	** GENERAL OPEN FAILURE	Echec d'ouverture de fichier
-220	** DISK FULL	Disque plein lors d'une tentative d'écriture dans le fichier log
-221	** DRIVE IS LOCKED	Disque inaccessible en écriture
-222	** SHARING VIOLATION	Tentative d'accès concurrent à un fichier
-223	** SHARING BUFFER EXCEEDED	Buffer plein
-224	** WARNING : PROBLEM DURING MODULE LIBERATION	Libération de module impossible (lors d'un SQL STOP)
-225	** INVALID PARAMETER	Paramètre invalide
-226	** ALREADY LOGGING	Le mode trace est déjà actif
-227	** PARAMETER SIZE GREATER THAN 65535, NOT SUPPORTED IN THIS VERSION	La taille spécifiée pour une variable hôte est trop grande

NS_FUNCTION GIVECOM

Récupère dans le segment COM_AREA les caractéristiques d'une table dont les composants sont inconnus lors de la sélection.
Cette fonction est particulièrement utile dans le traitement de requêtes dynamiques et permet de s'affranchir de la définition des variables hôtes et de la commande FETCH.

Syntaxe	**NS_FUNCTION GIVECOM INTO** *:caractéristiques-table*			
Paramètre	caractéristiques-table	INT(4)	I/O	pointeur sur le segment COM_AREA qui permet de récupérer les caractéristiques de la requête

1. Le segment COM_AREA (défini dans le fichier SQL_COM.NCL) est composé de différents champs dont deux pointeurs (HOST_PTR et SQL_PTR). Ces deux pointeurs peuvent être récupérés pour parcourir les tableaux contenant les variables NCL (le pointeur HOST_PTR) et les variables SQL (le pointeur SQL_PTR) concernées par l'ordre devant être exécuté.

```
; Définition de la structure de communication (GIVECOM
; INTO:)
SEGMENT COM_AREA
int reserved(4) ;réservé
int transaction(2) ;réservé
int statement(2) ;réservé
int host_ptr(4) ;pointeur vers un segment de type NCLELEMENT ;(définissant
les variables hôtes NCL)
int sql_ptr(4) ;pointeur vers un segment de type
;SQLELEMENT
    ;(définissant les colonnes des tables
;de la requête)
int com_ptr(4) ;réservé
int num_stat(2) ;type de requête
    ; 1 -> SELECT
    ; 2 -> UPDATE
    ; 3 -> DELETE
    ; 4 -> INSERT
    ; 5 -> Autres types de requêtes
int num_col(2) ; nombre de colonnes
int num_col_compute(2) ;nombre de colonnes COMPUTE
int len_buf_stat(2) ; taille du buf_stat ci-dessous
int buf_stat(4) ; pointeur sur un buffer contenant
;l'instruction FETCH INTO [ :,] suivie d'autant de « :, » ;que de variables
à parcourir dans le cas d'un SELECT.
int inited(2) ;TRUE si tout est OK, FALSE sinon. A
;tester
;toujours s'il est à TRUE
ENDSEGMENT
```

2. La bibliothèque de fonctions SQL_COM.NCL, fournit un ensemble de fonctions nécessaires à l'exploitation de la fonction NS_FUNCTION GIVECOM INTO :
   a) = la structure de communication,
   b) = les fonctions qui retournent la nature de la commande à exécuter,
   c) = l'ensemble des fonctions qui permettent de récupérer les pointeurs,
   d) = les types, les tailles et les noms des colonnes concernées par la sélection.
3. Lorsque la nature de la commande a été identifiée (utilisation des fonctions SQL_GET_STATEMENT% et SQL_GET_STATEMENT$) comme une clause SELECT, la commande SQL EXEC LONGSTR peut alors exécuter la requête remplissant la zone réceptrice d'où peuvent être extrait les résultats, avec les fonctions de la librairie NCL.
4. La liste des fonctions contenues dans la bibliothèque NCL est la suivante :

<table>
<tr><th>Nom de la fonction</th><th>Description</th></tr>
<tr><td>SQL_GET_HOSTPTR%</td><td>Récupère un pointeur sur un tableau de variables COM_NCLELEMENT (définition des variables réceptrices NCL).
<table><tr><td>Variable</td><td>COM_BUFFER%
INT(4)
Pointeur sur COM_AREA</td></tr><tr><td>Valeur retournée</td><td>INT(4)</td></tr></table></td></tr>
<tr><td>SQL_GET_SQLPTR%</td><td>Récupère un pointeur sur un tableau de variables COM_SQLELEMENT.
<table><tr><td>Variable</td><td>COM_BUFFER%
INT(4)
Pointeur sur COM_AREA</td></tr><tr><td>Valeur retournée</td><td>INT(4)</td></tr></table></td></tr>
<tr><td>SQL_GET_STATEMENT%</td><td>Récupère le type de commande qui a été exécuté (valeur entière) à partir du buffer num_stat du segment COM_AREA.
<table><tr><td>Variable</td><td>COM_BUFFER%
INT(4)
Pointeur sur COM_AREA</td></tr><tr><td>Valeur retournée</td><td>INT(2)</td></tr></table></td></tr>
</table>

<table>
<tr><td>SQL_GET_STATEMENT$</td><td>Récupère le type de commande qui a été exécuté (valeur alpha) à partir du buffer num_stat du segment COM_AREA et le convertit en valeur CSTRING.

Les valeurs de num_stat sont les suivantes :
1 pour SELECT
2 pour UPDATE
3 pour DELETE
4 pour INSERT
0 pour tout autre type de requête
<table><tr><td>Variable</td><td>STATEMENT%
INT(4)
Fonction SQL_GET_STATEMENT%</td></tr><tr><td>Valeur retournée</td><td>CSTRING</td></tr></table></td></tr>
<tr><td>SQL_GET_NBCOL%</td><td>Récupère le nombre de colonnes renvoyées par le STATEMENT.<table><tr><td>Variable</td><td>COM_BUFFER%
INT(4)
Pointeur sur COM_AREA</td></tr><tr><td>Valeur retournée</td><td>INT(2)</td></tr></table></td></tr>
<tr><td>SQL_GET_LENGTHFETCH%</td><td>Récupère la taille du buffer qui contient le FETCH.<table><tr><td>Variable</td><td>COM_BUFFER%
INT(4)
Pointeur sur COM_AREA</td></tr><tr><td>Valeur retournée</td><td>INT(4)</td></tr></table></td></tr>
</table>

<table>
<tr><td>SQL_GET_FETCHPTR%</td><td>Récupère le pointeur du buffer qui contient le FETCH.<table><tr><td>Variable</td><td>COM_BUFFER%
INT(4)
Pointeur sur COM_AREA</td></tr><tr><td>Valeur retournée</td><td>INT(4)</td></tr></table></td></tr>
<tr><td>SQL_GET_HOSTCOLUMNPTR%</td><td>Récupère le pointeur qui contient la donnée pour un élément du tableau des variables NCL.<table><tr><td rowspan="2">Variables</td><td>COM_BUFFER%
INT(4)
Pointeur sur COM_AREA</td></tr><tr><td>COLUMN%
INT(2)
Ordre de la variable NCL</td></tr><tr><td>Valeur retournée</td><td>INT(4)</td></tr></table></td></tr>
<tr><td>SQL_GET_HOSTCOLUMNTYPE%</td><td>Récupère le type de la donnée pour un élément du tableau des variables NCL (valeur entière).<table><tr><td rowspan="2">Variables</td><td>COM_BUFFER%
INT(4)
Pointeur sur COM_AREA</td></tr><tr><td>COLUMN%
INT(2)
Ordre de la variable NCL</td></tr><tr><td>Valeur retournée</td><td>INT(2)</td></tr></table></td></tr>
<tr><td>SQL_GET_HOSTCOLUMNTYPE$</td><td>Récupère le type de la donnée pour un élément du tableau des variables NCL (valeur alpha).<table><tr><td>Variable</td><td>TYPE%
INT(4)
Fonction SQL_GET_HOSTCOLUMNLENGTH%</td></tr><tr><td>Valeur retournée</td><td>CSTRING(80)</td></tr></table></td></tr>
</table>

<table>
<tr><td>SQL_GET_HOSTCOLUMNLENGTH%</td><td>Récupère la taille de la donnée pour un élément du tableau des variables NCL.

<table><tr><td rowspan="2">Variables</td><td>COM_BUFFER%
INT(4)
Fonction SQL_GET_HOSTCOLUMNLENGTH%</td></tr><tr><td>COLUMN%
INT(2)
Ordre de la variable NCL</td></tr><tr><td>Valeur retournée</td><td>INT(2)</td></tr></table></td></tr>
<tr><td>SQL_GET_SQLCOLUMNNAME$</td><td>Récupère le nom de la colonne du tableau des colonnes SQL.

<table><tr><td rowspan="2">Variables</td><td>COM_BUFFER%
INT(4)
Pointeur sur COM_AREA</td></tr><tr><td>COLUMN%
INT(2)
Ordre de la variable NCL</td></tr><tr><td>Valeur retournée</td><td>CSTRING(64)</td></tr></table></td></tr>
</table>

Nat System vous informe que les cinq fonctions suivantes ne sont guère utiles avec la NS_FUNCTION GIVECOM. Cependant, nous les laissons dans la documentation pour compatibilité avec les anciennes versions de la documentation.

Nom de la fonction	Description	Exemple
SQL_GET_SQLCOLUMNTYPE%	Récupère le type SGBD de la colonne du tableau des colonnes SQL.	FUNCTION SQL_GET_SQLCOLUMNTYPE% \ (INT COM_BUFFER%(4),INT COLUMN%(2))\ RETURN INT(2)

SQL_GET_SQLCOLUMNLENGTH%	Récupère la taille SGBD de la colonne du tableau des colonnes SQL.	FUNCTION SQL_GET_SQLCOLUMNLENGTH% \ (INT COM_BUFFER%(4),INT COLUMN%(2))\ RETURN INT(4)
SQL_GET_SQLCOLUMNSERVICE%	Récupère le service SGBD de la colonne du tableau des colonnes SQL (valeur entière).	FUNCTION SQL_GET_SQLCOLUMNSERVICE% \ (INT COM_BUFFER%(4),INT COLUMN%(2)) \ RETURN INT(2)
SQL_GET_SQLCOLUMNREF%	Récupère le numéro de la colonne en rapport avec compute.	FUNCTION SQL_GET_SQLCOLUMNREF% \ (INT COM_BUFFER%(4),INT COLUMN%(2)) \ RETURN INT(2)
SQL_GET_SQLCOLUMNSERVICE$	Récupère le service SGBD (valeur alpha).	FUNCTION SQL_GET_SQLCOLUMNSERVICE$ \ (INT service%(2)) \ RETURN CSTRING(80)

```
;Exemple
LOCAL COM_AREA_RET%, TOTAL_COL%, I%, NCL_PTR%, BUFFER_PTR%
LOCAL COMPUTE% , HEADER$, A$
MOVE "SELECT * FROM EMP" TO A$
SQL_EXECSTR A$
MOVE 0 TO COM_AREA_RET%

WHILE SQL_ERROR% = 0
 SQL_EXEC NS_FUNCTION GIVECOM INTO :COM_AREA_RET%
 IF COM_AREA_RET% = 0
  BREAK
 ENDIF
 INSERT AT END SQL_GET_STATEMENT$ (SQL_GET_STATEMENT%(COM_AREA_RET%) ) TO \
LISTBOX1
 ; récupération de la chaîne de la commande
 UPDATE LISTBOX1
 IF SQL_GET_STATEMENT%(COM_AREA_RET%) <> 1
  ; la valeur de la commande est différente de SELECT
  RETURN 1
 ENDIF

 MOVE SQL_GET_HOSTPTR%(COM_AREA_RET%) TO NCL_PTR%
```

```
 ; récupération du pointeur sur le tableau de variables NCL
 MOVE SQL_GET_NBCOL%(COM_AREA_RET%) + SQL_GET_NBCOMPUTE%(COM_AREA_RET%)\
TO TOTAL_COL%
 ; récupère le nombre de colonnes + le nombre de colonnes de
;type COMPUTE
 IF SQL_GET_LENGTHFETCH%(COM_AREA_RET%) <> 0
  ; si la taille du buffer Fetch est <> 0
  i% = SQL_GET_FETCHPTR%(COM_AREA_RET%)
  MOV i% , @A$ , 255
 INSERT AT END A$ TO LISTBOX1

  SQL_EXEC_LONGSTR SQL_GET_FETCHPTR%(COM_AREA_RET%) , NCL_PTR%, -1
  ; retrieve the pointer to the Fetch buffer + execute
 ELSE
  BREAK
 ENDIF
 WHILE SQL_ERROR% = 0
  MOVE 0 TO I%
  WHILE I% < TOTAL_COL%
  MOVE SQL_GET_HOSTCOLUMNPTR%(COM_AREA_RET% ,i% ) TO BUFFER_PTR%
   ; récupération d'un pointeur qui contient une var NCL
   IF BUFFER_PTR% = 0
    MOVE I% + 1 TO I%
    CONTINUE
   ENDIF
   MOVE " " to HEADER$
   IF SQL_GET_SQLCOLUMNSERVICE% (COM_AREA_RET%,i%) <> 1
   ; si le service n'est pas une colonne
    MOVE I% + 1 TO I%
    CONTINUE
   ENDIF
   EVALUATE SQL_GET_HOSTCOLUMNTYPE%(COM_AREA_RET% ,i% )
   ; évaluation du type de la colonne
   ;CONST TYPE_SQL_INT% 0
   ;CONST TYPE_SQL_STRING% 1
   ;CONST TYPE_SQL_CSTRING% 2
   ;CONST TYPE_SQL_NUM% 3
   ;CONST TYPE_SQL_INSERT_BLOB% 13
   WHERE 0 ; integer
    ; récupération de la taille de la donnée
   EVALUATE SQL_GET_HOSTCOLUMNLENGTH%(COM_AREA_RET% ,i%)
    WHERE 1 ; 1-byte integer
    ; récupère le nom de la colonne
     INSERT AT END HEADER$ && \
            SQL_GET_SQLCOLUMNNAME$(COM_AREA_RET%, I% ) && ":" &&\
       ASC% (COM_INT1(BUFFER_PTR%).i1) TO LISTBOX1
    ENDWHERE
    WHERE 2 ; c'est un entier de 2
    ; récupère le nom de la colonne
    INSERT AT END HEADER$ && \
        SQL_GET_SQLCOLUMNNAME$(COM_AREA_RET% ,i% )&&\
       ":" && COM_INT2(BUFFER_PTR%).i2 TO LISTBOX1
    ENDWHERE
    WHERE 4 ; c'est un entier de 4
    INSERT AT END HEADER$ && \
        SQL_GET_SQLCOLUMNNAME$(COM_AREA_RET% ,i% )&&\
        ":" && COM_INT4(BUFFER_PTR%).i4 TO LISTBOX1
```

```
    ENDWHERE
    ENDEVALUATE
    ENDWHERE
    WHERE 2 ; c'est une chaîne C
    INSERT AT END HEADER$ && \
    SQL_GET_SQLCOLUMNNAME$(COM_AREA_RET%, I% ) &&\
    ":" && COM_STRING(BUFFER_PTR%).CS TO LISTBOX1
    ENDWHERE
    WHERE 3 ; c'est un réel
    EVALUATE SQL_GET_HOSTCOLUMNLENGTH%(COM_AREA_RET%,\ I%)
    ; récupère la taille de la colonne WHERE 4 ;
    ; récupère le nom de la colonne et la valeur
    ; et la valeur du réel de taille 4
    INSERT AT END HEADER$ && \
        SQL_GET_SQLCOLUMNNAME$(COM_AREA_RET%, I% ) &&\
 ":" && COM_FLOAT4(BUFFER_PTR%).f4 TO LISTBOX1
    ENDWHERE
    WHERE 8 ;
; récupère le nom de la colonne et la valeur
    ; du réel de 8
    INSERT AT END HEADER$ && \
        SQL_GET_SQLCOLUMNNAME$(COM_AREA_RET%, I% ) &&\
        ":" && COM_FLOAT8(BUFFER_PTR%).f8 TO LISTBOX1
      ENDWHERE
     ENDEVALUATE
    ENDWHERE
    ELSE
    ; récupère le type de la colonne NCL
    INSERT AT END "NCLType" && "INVALID" && \
    SQL_GET_HOSTCOLUMNTYPE%(COM_AREA_RET%, i%) TO \ LISTBOX1
    ; retrouve le type de colonne NCL
   ENDEVALUATE
   MOVE I% + 1 to I%
  ENDWHILE
  SQL_EXEC_LONGSTR SQL_GET_FETCHPTR%(COM_AREA_RET%),NCL_PTR%,-1
 ; exécution du fetch à partir du pointeur sur le buffer Fetch
 ENDWHILE
 UPDATE LISTBOX1
 IF SQL_ERROR% = 100
  INSERT AT END "END OF FETCH" TO LISTBOX1
  INSERT AT END "" TO LISTBOX1
 ENDIF
 IF SQL_ERROR% <> 100
  IF SQL_ERROR% > 0
   MESSAGE "WARNING" && SQL_ERROR% , SQL_ERRMSG$(SQL_ERROR%)
  ENDIF
  IF SQL_ERROR% < 0
   IF SQL_ERROR% = -32085;No more results to fetch
    INSERT AT END "END OF RESULT" TO LISTBOX1
   ELSE
    MESSAGE "ERROR" &&SQL_ERROR% , SQL_ERRMSG$(SQL_ERROR%)
   ENDIF
  ENDIF
 ENDIF
ENDWHILE
```

Voir aussi SQL_EXEC_LONGSTR

NS_FUNCTION ROWCOUNT

Récupère le nombre d'enregistrements affectés par une requête DELETE, UPDATE ou le nombre de FETCH effectué suite à un SELECT..

Syntaxe	**NS_FUNCTION ROWCOUNT INTO** *:nb-enregistrements*			
Paramètre	nb-enregistrements	INT(4)	O	nombre d'enregistrements affectés par une requête

```
;Exemple 1
LOCAL ROWCOUNT%
SQL_EXEC DELETE FROM TABPRODUIT WHERE NOPROD >= 30 AND NOPROD < 40
SQL_EXEC NS_FUNCTION ROWCOUNT INTO :ROWCOUNT%
; Si 10 enregistrements correspondent à ce filtre et donc 10
; enregistrement sont effacés, alors ROWCOUNT% contiendra 10.
; Si aucun enregistrement ne correspond à ce filtre alors ROWCOUNT% vaudra
0.

;Exemple 2
local var1%
local test$
LOCAL ROWCOUNT%
SQL_EXEC SELECT NUM, COL1 FROM BASE
IF SQL_ERROR% <> 0
 MESSAGE "Erreur ",SQL_ERRMSG$(SQL_ERROR%)
ENDIF
WHILE SQL_ERROR% = 0
 SQL_EXEC FETCH INTO:var1%,:test$
 IF SQL_ERROR% <> 0
  BREAK
 ENDIF
 INSERT AT END "Var1"&&var1%&& "test"&&test$ TO LISTBOX1
ENDWHILE
SQL_EXEC NS_FUNCTION ROWCOUNT INTO :ROWCOUNT%
Message "Number of occurences = ", ROWCOUNT%
```

***Voir aussi** NS_FUNCTION ANSION, NS_FUNCTION ANSIOFF, SQL_ERROR%, SQL_ERRMSG$*

NS_FUNCTION STATEMENT

Récupère la phrase exacte de la requête envoyée au moteur SQL. Le SELECT est tracé sans la clause INTO, même si elle est précisée.

Syntaxe	**NS_FUNCTION STATEMENT INTO** *:chaîne-requête*			
Paramètre	chaîne-requête	CSTRING	O	phrase de la requête envoyée au moteur SQL

Pour récupérer la totalité de la chaîne, il faut allouer dynamiquement la variable réceptrice.

```
; ----- Exemple avec Sybase Système dans le mode curseur 3
LOCAL VALUES$, PHRASE$

MOVE "BONJOUR" TO VALUES$
SQL_EXEC SELECT COL1 FROM TABLE WHERE COL2=:VALUES$

SQL_EXEC NS_FUNCTION STATEMENT INTO :PHRASE$
MESSAGE "La commande exacte est :", PHRASE$

PHRASE$ = SELECT COL1 FROM TABLE WHERE COL2='BONJOUR'
```

Commandes RECORD, REEXECUTE

La commande RECORD enregistre un ordre SQL en vue de sa réutilisation grâce à la commande REEXECUTE. Lors de l'appel de REEXECUTE, vous passez uniquement de nouvelles valeurs comme paramètres.

Syntaxe	**RECORD** *ordre-SQL* et **REEXECUTE**			
Paramètre	ordre-SQL	CSTRING	I	séquence SQL à enregistrer

1. Les paramètres de la séquence SQL doivent rester accessibles lors de l'exécution de la commande SQL_EXEC REEXECUTE.
2. Après un RECORD, tout ordre SQL autre que REEXECUTE annule le RECORD en cours.
3. Cette technique permet d'allier la souplesse du SQL dynamique à la rapidité d'exécution du SQL statique. En effet, un ordre SQL dynamique est utilisé lors de l'exécution de la commande RECORD, et lors de l'exécution de la commande REEXECUTE l'analyse de la requête au niveau du moteur a déjà été effectuée et seule une affectation des valeurs dans les variables hôtes est réalisée.
4. Ces deux ordres fonctionnent avec tous les ordres SQL ayant des paramètres et ceci même avec des curseurs.

```
;Exemple
LOCAL CODE%
LOCAL NOM$(25)

SQL_EXEC CREATE TABLE EMP(EMPNO INTEGER,ENAME CHAR(25))
CODE = 1
NOM$ = "NOM1"
SQL_EXEC RECORD INSERT INTO EMP VALUES (:CODE%,:NOM$)
FOR I= 2 TO 100
CODE = I
NOM$ = "NOM" & I
SQL_EXEC REEXECUTE
ENDFOR

; ---- La table EMP contient maintenant
; ( 1, "NOM1")
; ( 2, "NOM2")
; ( 3, "NOM3")
; ...
; ( 99, "NOM99")
; (100, "NOM100")
```

TABLEAU RECAPITULATIF POUR TOUS LES MOTEURS

Légende :

- X : implémenté sur le driver considéré
- **gras** -> mode par défaut

API	Oracle	RDB	Sybase	ODBC	DB2	Informix	MySQL	PostGre
ANSIOFF / ANSION	X	X	X	X	X			
IMAGEOFF / IMAGEON	X	X	X	X	X			
TRIMCHAROFF / TRIMCHARON	X	X	X	X	X			X
GIVECOM INTO :segment_handle	X	X	X	X	X			X
ROWCOUNT INTO :nb_record	X	X	X	X	X			X
STATEMENT INTO :requete_sql	X	X	X	X	X			
CHANGEDBCNTX :logicaldbname	X	X	X	X	X			X
CALLBACK :window_handle, :user_event	X	X	X	X	X			
ERRORCOUNT INTO :nb_error	X	X	X	X	X			X
GETERROR :index_error INTO :no_error	X	X	X	X	X			X
KILLQUERY	X		X					
SETBUFFERSIZE :buffer_size	X		X					X
DESCRIBEOFF / DESCRIBEON	X							
GETDBNAME INTO :logicaldbname	X							X
SETCURSORTYPE :cursortype		X						
ASYNCOFF / ASYNCON								
DATAREADY INTO :dataReady								
QUOTEOFF / QUOTEON			X	X	X			
GETCURRENTDBCNTX INTO :logicaldbname			X	X	X			
GETDBNAME INTO :physicaldbname			X	X	X			X
CHARTOHEXAOFF / CHARTOHEXAON			X	X	X			
SETCURSORMODE :mode			X	X	X			

CHANGEOPTION :parametre, :option			X	X	X			
GETTABLE :typobject, :ownername				X	X			
GETTABLEINFO :objecttype,:ownername,:tablename				X	X			
GETCOLUMN :objectname,:ownername,:refname				X	X			
GETINDEXCOLUMN :objectname,:ownername				X	X			
GETPRIMARYKEY :objectname,:ownername				X	X			
GETPROCEDURE				X	X			
GETPROCEDURECOLUMN :objectname,:ownername,:refname				X	X			
GETTYPEINFO :typsql%				X	X			
AUTOCOMMITOFF / **AUTOCOMMITON**				X	X			
RPCRETCODEOFF / RPCRETCODEON				X	X			
GETINFO :optionname,:status				X	X			

INDEX