Manuel d'utilisation PostgreSQL

Table des matières

POSTGRESQL

Résumé

Ce chapitre vous présente la librairie NSW2PGxx qui vous permet d'interfacer les applications développées au moyen des outils de développement Nat System avec les versions clientes de PostgreSQL.

Introduction

La librairie NSW2PGxx vous permet d'interfacer les applications développées au moyen des outils de développement Nat System avec les versions clientes de PostgreSQL.

Correspondance entre les drivers et les versions de PostgreSQL

Le nom du driver est à utiliser notamment avec l'instruction THINGS_DB_INIT pour NatStar et l'instruction SQL_INIT pour NatStar, NatWeb et NS-DK.

Le tableau suivant présente la dernière version de PostgreSQL et le driver correspondant.

Version de PostgreSQL	Driver
PostgreSQL 9.6	NSnnPG96.DLL

nn correspond au numéro de version de l'interface que vous avez installée : W2 pour Windows 32 bits

Pour simplifier l'écriture dans la documentation, NSW2PG96 est appelée sous le nom générique NSW2PGxx.

La version PostgreSQL 9.6 ne supporte pas le mode RECORD/REEXECUTE.

Installation

Vérifiez que le fichier NSW2PGxx.DLL se situe dans le répertoire où se trouvent les DLLs de votre environnement Nat System (ex : C:\NSDK7\BIN, C:\NATSTAR\BIN...).

Les librairies SQL livrées avec vos outils de développement Nat System s'interfacent avec les DLLs fournies par le concepteur du SGBD et celles de Windows. De plus, il est parfois nécessaire de démarrer un utilitaire. Vérifiez votre configuration avec les manuels du concepteur de PostgreSQL.

Conversion implicite de données en sortie

Il est fortement conseillé de se référer au Manuel PostgreSQL et aux documentations des drivers livrées avec PostgreSQL pour connaître en détail les conversions possibles.
Néanmoins, pour certaines données, utilisez les conversions suivantes :

PostgreSQL	NCL
BIT(n)	CSTRING (n)
BIT varying	CSTRING (n)
BOOLEAN	INT(1)
SMALLINT	INT(2)
INTEGER	INT(4)
BIGINT	INT(4)
FLOAT	NUM(4)
DOUBLE PRESICION	NUM(8)
REAL	NUM(4)
DECIMAL (variable)	NUM(8)
NUMERIC (variable)	NUM(8)
SERIAL	NUM(4)
BIGSERIAL	NUM(8)
DATE (4 bytes no time of day)	CSTRING
TIMESTAMP (8 bytes)	CSTRING
TIMESTAMP with TZ (8 bytes)	CSTRING
TIME	CSTRING
TIMESTAMP with TZ (12 bytes, no date)	CSTRING
BYTEA	DYNSTR / SQL_IMAGE / FILE
CHARACTER varying (n) varchar (n)	CSTRING (n)
CHARACTER (n) CHAR (n)	CSTRING (n)
TEXT	DYNSTR
"CHAR"	CSTRING

(*) Taille égale à 30, pour couvrir tous les formats.
(**) Taille limitée à 255, sinon utiliser les types Blob.
(***) Taille égale à celle de la plus longue chaîne de la liste.

Catégories fonctionnelles de la librairie NSW2PGxx

Initialiser le SGBD, terminer l'utilisation

Instruction SQL_INIT

Charge le driver PostgreSQL et initialise l'environnement de communication Application/Base PostgreSQL.

Syntaxe	**SQL_INIT** *nom-DLL*			
Paramètre	nom-DLL	CSTRING	I	nom du SGBD à initialiser

1. Cette fonction doit être la première instruction SQL_ appelée par toute application voulant utiliser une version donnée de PostgreSQL depuis le NCL.
2. Le paramètre nom-DLL doit contenir le nom de la DLL générique 'NSW2PGxx' qui permet d'accéder à PostgreSQL pour une cible donnée.

```
;Exemple
SQL_INIT 'NSW2PG96'
If (sql_error%)
 Message 'error at init'&&sql_error%, sql_errmsg$(sql_error%)
 Exit
Endif
...
SQL_STOP
```

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

Instruction SQL_STOP

Termine l'utilisation de PostgreSQL en fermant toutes les bases de données et les curseurs.

Syntaxe	SQL_STOP

```
;Exemple
SQL_INIT 'NSW2PG96'
If (sql_error%)
 Message 'error at init'&&sql_error%, sql_errmsg$(sql_error%)
 Exit
Endif
...
SQL_STOP
```

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

Ouvrir et fermer une base de données

Instruction SQL_OPEN

Établit une connexion entre l'application et la 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 un serveur

1. PostgreSQL vous permet d'ouvrir plusieurs bases de données, ou bien ouvrir la même plusieurs fois, comme décrit ci-après.
2. Le paramètre nom-logique-base sert d'alias interne à l'application pour désigner la connexion, il correspond au nom physique de la base de donnée PostgreSQL.
3. Le paramètre chaîne-connexion correspond à la chaîne de connexion à une base de données locale ou distante. Sa syntaxe est :

   USERID[/PASSWORD][@<CONNECTOR>]

   où :

   a) [USERID] : nom du compte utilisateur sur le serveur

   b) [PASSWORD] : mot de passe pour l'accès au compte utilisateur. Si le mot de passe n'est pas défini, la chaîne de connexion peut être réduite à USERID@<CONNECTOR>.

   c) [@<CONNECTOR>] : un ensemble d'options pour définir et/ou configurer la connexion à distance à établir tels que le host, la base cible et le port, séparés par le caractère ';'. Quand une option est facultative, ne pas la renseigner.

```
;Exemple :
"host=MyHost;base=MyBase;port=MyPort;"
```

Ci-dessous la liste des options possibles actuellement pour le paramètre [@<CONNECTOR>]:

- host=MyHost : nom du serveur (ou adresse IP) sur lequel est installée la base PostgreSQL, optionnel si c'est une base locale.
- base=MyBase : nom de la base de données sur laquelle établir la connexion, optionnel si le nom logique contient déjà le nom physique de la base.
- port=MyPort : numéro du port sur lequel écoute le serveur PostgreSQL (5432 par défaut), optionnel.

```
;Exemple 1
;Chaîne de connexion la plus explicite :
sql_open 'PostGresDB', 'user/password@host=pollux;base=MyBase;port=5432'
; connexion à MyBase sur le serveur pollux et le port 5432. Si base n'est
pas renseigné, l'alias PostGresDB est pris comme nom de base à ouvrir
```

```
;;;;;; Traitements SQL
sql_close 'PostGresDB'

; Exemple 2
;Chaîne de connexion la plus simple :
sql_open 'MyBase', 'user/password@pollux'
; connexion à MyBase sur le serveur pollux et le port 5432
;;;;;; Traitements SQL
sql_close 'MyBase'
```

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

Instruction SQL_CLOSE

Ferme une connexion de la base de données.

Syntaxe	**SQL_CLOSE** *nom-logique-base*			
Paramètre	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 exécute automatiquement les SQL_CLOSE de ces bases de données.

> Reportez-vous aux exemples de l'instruction SQL_OPEN.

Gérer les erreurs

Fonction SQL_ERROR%

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

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

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 avertissement),
   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 PostgreSQL.
3. Le message (pour la log) se construit de la façon suivante : SQL_ERRMSG$(SQL_ERROR%).
4. 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".

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

5. PostgreSQL peut néanmoins retourner des valeurs d'erreur supérieures à 40000. Deux codes spécifiques ont donc été implémentés pour ne pas entrer en interférence avec les erreurs du driver supérieures à 32000 :
   a) -32700 : erreur spécifique PostgreSQL, consultable avec SQL_ERRMSG$(SQL_ERROR%),
   b) 32700 : warning spécifique PostgreSQL, consultable avec SQL_ERRMSG$(SQL_ERROR%).
6. Liste des codes et des messages d'erreurs internes Nat System :

Codes	Messages d'erreur	Cause
0	"NSSQLE000 ** UNUSED NATSYS ERROR CODE"	Pas d'erreur, l'exécution s'est bien déroulée.
+100	"NSSQLW100 ** NO ROW WAS FOUND OR LAST ROW REACHED"	Plus ou pas de ligne trouvée suite à un FETCH ou un SELECT.
-32001	"NSSQLE001 ** HEAP ALLOCATION ERROR"	Erreur interne d'allocation/désallocation mémoire.

-32002	"NSSQLE002 ** DYNAMIC ALLOCATION ERROR"	Erreur interne d'allocation/désallocation mémoire.
-32003	"NSSQLE003 ** DYNAMIC FREE STORAGE ERROR"	Erreur interne d'allocation/désallocation mémoire.
-32004	"NSSQLE004 ** NO MORE CURSORS AVAILABLE"	Trop de curseurs ouverts simultanément.
-32005	"NSSQLE005 ** NO MORE CURSORS OR TRYING TO DEALLOCATE ONLY CURSOR"	
-32006	"NSSQLE006 ** INVALID INTO CLAUSE in FETCH/SELECT"	Erreur de syntaxe dans la clause INTO d'un SELECT ou d'un FETCH.
-32007	"NSSQLE007 ** TOO MANY VARIABLES IN INTO CLAUSE"	Plus de 200 variables dans la clause INTO
-32008	"NSSQLE008 ** MISSING HOST VARIABLE AFTER ',""	Erreur de syntaxe dans une clause INTO. Une virgule de continuation est trouvée sans variable derrière.
-32009	"NSSQLW009 ** INTO CLAUSE : NOT ENOUGH VARIABLES"	Un SELECT contient une clause INTO dont le nombre de variables est inférieur à celui qui sera renvoyé par la requête. Le système remplit néanmoins les variables hôtes qui ont été mises à sa disposition.
-32010	"NSSQLW010 ** AN OPENED CURSOR WAS CLOSED BY SYSTEM"	Suite à l'arrivée d'un nouvel ordre SQL sur un curseur, le système a forcé la fermeture du curseur qui contenait une requête active.
-32011	"NSSQLE011 ** WHERE/VALUE CLAUSE : NOT ENOUGH VARIABLES"	Nombre insuffisant de variables hôtes dans la table reçue pour pouvoir substituer les variables spécifiées dans la clause WHERE.
-32012	"NSSQLE012 ** INVALID INPUT VARIABLE DATA TYPE"	Type de donnée invalide dans une clause WHERE.
-32013	"NSSQLE013 ** MISSING 'OF' AFTER 'WHERE CURRENT'	Erreur de syntaxe dans l'ordre UPDATE WHERE CURRENT OF.
-32014	"NSSQLE014 ** NO OUTPUT VARIABLES DEFINED FOR FETCH"	Le FETCH ainsi que le SELECT qui l'a précédé n'ont pas défini de variables de sortie (clause INTO).

-32015	"NSSQLE015 ** CURSOR NOT READY (MISSING SELECT) "	Tentative de FETCH sans SELECT préalable, ou curseur fermé par le système entre l'ordre SELECT et l'ordre FETCH.
-32016	"NSSQLE016 ** INVALID SQL DATA TYPE"	Type de donnée invalide en sortie.
-32017	"NSSQLE017 ** INVALID DATA CONVERSION REQUESTED"	Conversion de type invalide en sortie. STRING -> NUM NUM -> STRING REAL -> INTEGER INTEGER -> REAL
-32018	"NSSQLE018 ** NUMERIC DATA TYPE: INVALID LENGTH"	Longueur invalide pour un type de donnée (ex : réel longueur 48).
-32019	"NSSQLE019 ** INVALID DECIMAL PACKED FORMAT"	La conversion des données en format décimal n'a pu être faite.
+32020	"NSSQLW020 ** STRING DATA TRUNCATED"	La chaîne passée en variable est plus courte que la zone reçue du SGBD. Une troncature de la zone reçue s'est donc produite.
-32021	"NSSQLE021 ** RESET STORAGE ERROR"	Erreur de désallocation du heap interne.
+32022	"NSSQLW022 ** FUNCTION NOT SUPPORTED IN POSTGRESQL DATABASE"	L'instruction exécutée n'a aucun effet.
-32023	"NSSQLE023 ** TOO MANY OPENED DATABASES"	Plus de 5 bases de données ouvertes simultanément.
-32024	"NSSQLW024 ** DB ALREADY OPENED"	La base utilisée avec SQL_OPEN a déjà été ouverte.
-32025	"NSSQLE025 ** DB NOT PREVIOUSLY OPENED"	Tentative de fermeture d'une base qui n'a pas été ouverte.
-32026	"NSSQLE026 ** INVALID DATABASE NAME REF"	Utilisation d'un nom de base inconnue dans la clause AT de l'ordre SQL_EXEC (base non ouverte).
-32028	"NSSQLE028 ** UNABLE TO GET POSTGRESQL LOGIN"	Echec de la connexion (erreur sur le nom du serveur, etc.).
-32029	"NSSQLE029 ** POSTGRESQL VARIABLE INPUT BIND FAILED"	Incompatibilité de type entre une variable et un champ de la base.
-32030	"NSSQLE030 ** POSTGRESQL VARIABLE OUTPUT BIND FAILED"	Incompatibilité de type entre une variable et un champ de la base.

-32031	"NSSQLE031 ** POSTGRESQL BUFFER FILL ERROR"	Saturation du buffer (suite à des conversions de données...).
-32032	"NSSQLE032 ** RPC PARAMETER NAME EXPECTED"	Appel à distance de procédures : il manque le nom d'un paramètre.
-32033	"NSSQLE033 ** TOO MANY RPC PARAMETERS"	Appel à distance de procédures : trop de paramètres ont été spécifiés.
-32034	"NSSQLE034 ** RPC PROCEDURE NAME EXPECTED"	Appel à distance de procédures : il manque le nom de la procédure.
-32035	"NSSQLE035 ** NOT ENOUGH PARAMETERS FOR RPC CALLS"	Appel à distance de procédures : il manque des paramètres.
-32036	"NSSQLE036 ** INVALID RPC PARAMETERS SUPPLIED"	Appel à distance de procédures : des paramètres incorrects ont été spécifiés.
-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 : SQL SERVER ERROR DURING RPC PREPARATION"	
-32064	"NSSQLE064 ** RPC : SQL SERVER ERROR DURING RPC EXECUTION"	
-32065	"NSSQLE065 ** RPC : SQL SERVER 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 : SQL SERVER ERROR DURING SIZE BUFFERING EXECUTION"	
-32071	"NSSQLE071 ** EXEC : SQL SERVER 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 : SQL SERVER : 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"	
-32013	"NSSQLE103 ** NO CONNECTION AVAILABLE TO UPDATE IMAGE/TEXT"	
-32104	"NSSQLE104 ** CONNECTION CLOSED BY SERVER"	
-32200	"NSSQLE200 ** COMPUTE RESULT IN PROGRESS"	

Voir aussi Messages d'erreur du chapitre La librairie NSnn_SQL, SQL_ERRMSG$

Fonction SQL_ERRMSG$

Retourne le message d'erreur (chaîne de caractères) 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 dernier message stocké dans une zone de travail de la DLL lors de l'occurrence de l'erreur. Elle ne tient donc pas compte du code-erreur passé en paramètre.
2. Reportez-vous à la fonction SQL_ERROR% pour une liste détaillée des messages et codes d'erreur

Voir aussi Messages d'erreur du chapitre La librairie NSnn_SQL, SQL_ERROR%

NS_FUNCTION CALLBACK

Cette fonction permet de rediriger les messages d'erreurs vers une fenêtre applicative. Chaque fois qu'un message d'erreur apparaît, un événement est envoyé à la fenêtre.
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*			
Variables	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. 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.
2. 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.
3. Pour annuler cet ordre, il suffit de passer un handle de fenêtre égal à zéro.
4. 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 des autres structures) :

```
SEGMENT DB_POSTGRESQL_CLIENT_STRUCT
  INT   errorType(2)     ; =1 => Warning , =2 => Erreur
  INT   dbError(2)       ; Error code in abs()
  CHAR  dbErrorStr(512) ; Error message (with a '\0')
ENDSEGMENT
```

5. Le type de message d'erreur (clientmsg) est envoyé dans PARAM12%. Le handle de la structure est envoyé dans PARAM34%.

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

NS_FUNCTION ERRORCOUNT

Récupère le nombre d'erreurs ou de messages d'erreurs rencontrés lors de l'exécution d'une requête.

La NS_FUNCTION ERRORCOUNT est utile principalement pour conserver la compatibilité avec d'autres bases de données telles que Microsoft SQL Server ou Sybase.

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

1. Les messages d'erreurs sont numérotés 0 ou 1.
2. Seul le dernier message d'erreur est accessible.

```
;Exemple
SQL_EXEC NS_FUNCTION ERRORCOUNT INTO :ERRORCOUNT%
MESSAGE 'NOMBRE D'ERREURS',ERRORCOUNT%
```

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

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.

La NS_FUNCTION GETERROR est utile principalement pour conserver la compatibilité avec d'autres bases de données telles que Microsoft SQL Server ou Sybase.

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$*

Exécuter un ordre SQL : SELECT, INSERT, UPDATE, CREATE TABLE . . .

Instruction SQL_EXEC

Exécute un ordre SQL : SELECT, INSERT, UPDATE, CREATE TABLE...

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 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.
6. Les commandes SQL_EXEC, SQL_EXECSTR et SQL_EXEC_LONGSTR dépendent du langage SQL accepté par le SGBD utilisé (Cf. Manuels PostgreSQL ou http://www.postgresql.com).
7. 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$
```

8. 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,

f) DYNSTR.

9. Les conversions autorisées entre les types NCL et les types SQL sont définis dans le paragraphe Conversion implicite de données en sortie de ce même manuel.

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

11. 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.

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

13. Un indicateur est une variable entière NCL qui peut prendre les valeurs suivantes :

a) NULL_VALUE_INDICATOR (-1) : cette valeur indique que la variable NCL associée qui le précède vaut NULL.

b) 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.

14. En langage SQL, NULL ne signifie pas 0 ou une chaîne vide (""). PostgreSQL distingue les chaînes vides ("") des chaînes NULL (non saisies).

15. PostgreSQL ne vérifie pas la cohérence du type de la donnée à insérer avec le type de la colonne réceptrice, une conversion implicite est effectuée le cas échéant. Exemples :

a) Un char non numérique inséré dans une colonne de type entier est inséré nul (0). En effet, la conversion s'arrête au premier char non numérique, exemple : '3 petits cochons' inséré dans un entier, donne '3'.

b) Un float inséré dans une colonne de type entier est tronqué à sa partie entière (3.25 à 3).

c) Une date non valide insérée dans une colonne de type date, est insérée nulle (0000-00-00 format US, par défaut).

16. PostgreSQL retire tous les blancs des chaînes de caractères (Trim), ce qui signifie que les blancs en fin de chaîne ne sont jamais insérés, ni pris en compte dans l'égalité entre deux chaînes.

17. Pour les images/textes, l'insertion se fait à partir de buffers mémoire type NCL CHAR ou NCL SQL_IMAGE dont la taille est limitée à 64k.

18. L'insertion et l'extraction à partir de fichiers (TYPE_SQL_INSERT_BLOB%) sont possibles.

19. Toute donnée PostgreSQL peut être stockée dans tout type NCL, mais il est fortement conseillé d'utiliser le bon type NCL recommandé ci-dessus avec la bonne taille pour ne pas perdre en précision lors des conversions. Sachant qu'à

l'insertion, les NULL sont différents d'une chaîne vide (""), ces deux requêtes sont totalement différentes :

a) SELECT * from MyTable where MyCol is NULL;
b) SELECT * from MyTable where MyCol='';

Pour les images/textes, on peut utiliser le Type NCL TYPE_SQL_SELECT_BLOB% pour les extraire sous forme de fichiers. Les images/textes de moins de 64k peuvent être extraites directement dans des buffers mémoire avec les type NCL Char ou NCL SQL_IMAGE.

```
;Exemple 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%
;;;;;;;;;;;;;;;;;; tester l'erreur f_error%
localimage.realsize = size%
localimage.length% = size%
localimage.ptr% = dataptr%

sql_exec insert into test_image values (1, "tintin sans milou",
:localimage)
;;;;;;;;;;;;;;;;;; tester l'erreur sql_error%

f_close file%
dispose dataptr%

sql_image ns_function imageoff

;Exemple 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
;;;;;;;;;;;;;;;;;; tester l'erreur sql_error%

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

```
f_blockwrite file%, localimage.ptr%, localimage.realsize, nbread%
;;;;;;;;;;;;;;;;;; tester l'erreur f_error%
f_close file%

dispose localimage.ptr%
sql_exec ns_function imageoff
```

20. Une transaction de mise à jour doit commencer par un ordre BEGIN. Si la transaction est validée, il faut la finir par un ordre COMMIT, sinon, par un ordre ROLLBACK. Par défaut (sans ordre BEGIN), PostgreSQL exécute les transactions en mode "auto commit" : chaque instruction est exécutée dans sa propre transaction et le COMMIT est fait implicitement à la fin de l'instruction lorsque l'exécution est réussie (dans le cas contraire, c'est un ROLLBACK qui est effectué).

```
;Exemple 3
SQL_EXEC BEGIN
SQL_EXEC drop table TestTest
SQL_EXEC insert...
...
If SQL_ERROR% = 0
   SQL_EXEC COMMIT
else
   SQL_EXEC ROLLBACK
endif
```

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

Instruction SQL_EXECSTR

Exécute un ordre SQL : SELECT, INSERT, UPDATE, CREATE TABLE...

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 simple 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 et de chaque indicateur de nullité est repéré par un caractère ":" dans le texte de la requête. A chaque ":" correspond la variable host ou un indicateur de nullité passé 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
REQ$ = "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 REQ$, PERE$, IND1%, AGE%, FILS$, IND2%, AGE% USING CURS1%

WHILE SQL_ERROR% = 0
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 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 très long ordre SQL : SELECT, INSERT, UPDATE, CREATE TABLE...

Syntaxe	**SQL_EXEC_LONGSTR** *adresse-chaîne-sql, adresse-tableau-var, num-curseur*			
Paramètres	adresse-chaîne-sql	INT(4)	I	adresse de la chaîne de caractères qui contient l'ordre SQL à exécuter
	adresse-tableau-var	INT(4)	I	adresse du tableau qui contient les variables réceptrices (voire 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 est illimitée.

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 et de chaque indicateur de nullité est repéré par un caractère ":" dans le texte de la requête. A chaque ":" correspond la variable host ou un indicateur de nullité 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
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
;(ne fonctionne pas avec PostgreSQL 4.0)
```

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_EXEC_LONGSTR%.

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,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
```

Voir aussi FILL, SQL_EXEC, SQL_EXECSTR, SQL_ERROR%, SQL_ERRMSG$

NS_FUNCTION CHANGEOPTION

Modifie les paramètres de connexion entre le serveur et la partie cliente.

Syntaxe	**NS_FUNCTION CHANGEOPTION** *:paramètre , :option*			
Paramètres	paramètre	CSTRING	I	paramètre à modifier
	option	*	I	valeur à affecter au paramètre

* selon la valeur de paramètre, la seconde variable sera soit un ENTIER, soit une CHAINE (cf. tableau ci-dessous).

Paramètres	Type NCL	Minimum	Option	Valeur par défaut
ANSINULL	INTEGER		TRUE/FALSE	FALSE
ANSIPERM	INTEGER		TRUE/FALSE	FALSE
ARITHABORT	INTEGER		TRUE/FALSE	FALSE
ARITHIGNORE	INTEGER		TRUE/FALSE	FALSE
AUTHOFF	STRING		"sa" "sso" "oper"	
AUTHON	STRING		"sa" "sso" "oper"	
CHAINXACTS	INTEGER		TRUE/FALSE	FALSE
CURCLOSEON	INTEGER		TRUE/FALSE	FALSE
CURREAD	STRING		STRING VALUE	NULL
CURWRITE	STRING		STRING VALUE	NULL
Paramètres	Type NCL	Minimum	Option	Valeur par défaut
DATEFIRST	INTEGER		1=SUNDAY 2=MONDAY .... 7=SATURDAY	SUNDAY
DATEFORMAT	INTEGER		1=MDY 2=DMY 3=YMD 4=YDM 5=MYD 6=DYM	MDY
FIPSFLAG	INTEGER		TRUE/FALSE	FALSE
FORCEPLAN	INTEGER		TRUE/FALSE	FALSE

FORMATONLY	INTEGER		TRUE/FALSE	FALSE
GETDATA	INTEGER		TRUE/FALSE	FALSE
IDENTITYOFF	STRING		table name	NULL
IDENTITYON	STRING		table name	NULL
ISOLATION	INTEGER		1=LEVEL1 3=LEVEL3	LEVEL1
NOCOUNT	INTEGER		TRUE/FALSE	FALSE
NOEXEC	INTEGER		TRUE/FALSE	FALSE
PARSEONLY	INTEGER		TRUE/FALSE	FALSE
QUOTED_IDENT	INTEGER		TRUE/FALSE	FALSE
RESTREES	INTEGER		TRUE/FALSE	FALSE
ROWCOUNT	STRING	0	INTEGER VALUE	0
SHOWPLAN	INTEGER		TRUE/FALSE	FALSE
STATS_IO	INTEGER		TRUE/FALSE	FALSE
STATS_TIME	INTEGER		TRUE/FALSE	FALSE
Paramètres	Type NCL	Minimum	Option	Valeur par défaut
STR_RTRUNC	INTEGER		TRUE/FALSE	FALSE
TEXTSIZE	INTEGER		INTEGER VALUE	32768
TIMEOUT	INTEGER	0		120 secondes
TRUNCIGNORE	INTEGER		TRUE/FALSE	FALSE

N.B. : TRUE = 1 et FALSE = 0

```
;Exemple
LOCAL PARAMETRE$
LOCAL TMPS%

MOVE 30 TO TMPS%
MOVE "TIMEOUT" TO PARAMETRE$

SQL_EXEC NS_FUNCTION CHANGEOPTION :PARAMETRE$, :TMPS%
```

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 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 KILLQUERY

Détruit la requête envoyée au serveur.

Cette fonction a été créée pour éviter de bloquer une application lorsqu'une commande FETCH risque de prendre trop de temps avant de se terminer.

Syntaxe	NS_FUNCTION KILLQUERY

Cette fonction peut donc être utilisée pour annuler l'exécution d'une requête en cours de traitement par le serveur.

```
;Exemple
LOCAL I%, PRIXTTC%, TOTAL%

I% = 0
TOTAL% = 0
SQL_EXEC SELECT PTTC FROM LFACTURE WHERE NOFACT = 10
WHILE SQL_ERROR% = 0
IF I% >= 4
SQL_EXEC NS_FUNCTION KILLQUERY
BREAK
ELSE
SQL_EXEC FETCH INTO  :PRIXTTC%
TOTAL% = TOTAL% + PRIXTTC%
I% = I% + 1
ENDIF
ENDWHILE
MESSAGE 'La somme des ' & I% & ' premières lignes de la facture n° 10 est
égale à', TOTAL%
```

NS_FUNCTION ROWCOUNT

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

Syntaxe	**NS_FUNCTION ROWCOUNT INTO** *:nb-enregistrement*			
Paramètre	nb-enregistrement	INT(4)	I	nombre d'enregistrements

```
;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
enregistrements 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** SQL_ERROR%, SQL_ERRMSG$*

NS_FUNCTION SETCURSORMODE

Précise le mode d'utilisation du curseur ouvert par les fonctions SQL_OPENCURSOR% ou SQL_OPENTHECURSOR%.

Syntaxe	**NS_FUNCTION SETCURSORMODE** *:mode*			
Paramètre	mode	INT(4)	I	mode d'utilisation d'un curseur

1. Il existe deux modes d'utilisation d'un curseur :
   a) le mode par défaut,
   b) le mode compatible.
2. Une valeur est associée à chaque mode et chaque mode a une taille par défaut :
   a) Valeur DB_POSTGRESQL_CURSORDEFAULT
      i. Mode = défaut
      ii. Taille par défaut = 100 enregistrements
   b) Valeur DB_POSTGRESQL_CURSORNONE
      i. Mode = compatible
      ii. Taille par défaut = 1 enregistrement
3. Intérêt comparé de ces modes :
   a) Le mode DB_POSTGRESQL_CURSORNONE (0) permet d'effectuer toutes les requêtes sans autoriser le positionnement relatif.
   b) Le mode DB_POSTGRESQL_CURSORDEFAULT (3) permet de lire en un seul Fetch l'ensemble des enregistrements contenus dans le buffer spécifié. Ceci est particulièrement adapté à une petite sélection (<1000 enregistrements).
4. La taille peut être modifiée grâce à la commande NS_FUNCTION SETBUFFERSIZE.
5. Le mode par Défaut est le mode standard. Lors d'un parcours de table, chaque FETCH sera exécuté un à un.
6. Le mode Compatible est le mode qui permet de changer le mode curseur (donc de passer en mode langage) sur un curseur donné. Ce mode assure la compatibilité avec les versions précédentes du serveur SQL.

Cette fonction doit être utilisée dans des cas très précis. Il est conseillé de passer dans ce mode de curseur pour effectuer une requête précise puis de revenir immédiatement au mode par défaut.

```
;Exemple
LOCAL MODECURS%

MOVE DB_POSTGRESQL_CURSORBINDING TO MODECURS%
SQL_EXEC NS_FUNCTION SETCURSORMODE :MODECURS%

; les recherches s'effectueront selon le mode "Compatible"
; (langage de commande)
```

Voir aussi NS_FUNCTION SETBUFFERSIZE

NS_FUNCTION STATEMENT

Récupère l'instruction exacte de la requête envoyée au moteur SQL.

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

1. La clause INTO (même précisée dans la requête SELECT) n'est jamais tracée.
2. Si la taille de la variable HOST est plus petite que la requête réelle, la requête sera tronquée et trois points seront ajoutés à la fin de la chaîne de caractères.

```
;Exemple
LOCAL VALUES$, PHRASE$

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

SQL_EXEC NS_FUNCTION STATEMENT INTO :PHRASE$
MESSAGE "the query is :", PHRASE$

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

Gérer le curseur

Fonction SQL_OPENCURSOR%

Ouvre un curseur et retourne son handle.

Syntaxe	SQL_OPENCURSOR%
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 NSW2PGxx 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. SQL_OPENCURSOR% permet de résoudre ce problème car tous les ordres SELECT FETCH seront alors exécutés sur ce nouveau curseur.
6. 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.
7. La DLL NSW2PGxx spécifique au SGBD stocke les curseurs en pile LIFO (Last In First Out) avec SQL_OPENCURSOR% qui empile et SQL_CLOSECURSOR qui dépile.
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, le curseur ouvert par SQL_OPENCURSOR% est immédiatement associé à la base courante.
10. 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%.

```
; 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 .... ; utilise le curseur par défaut
```

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

Instruction SQL_CLOSECURSOR

Ferme le curseur le plus récemment ouvert et le dernier occupé par SQL_OPENCURSOR%.

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 doit fermer que les curseurs ouverts par SQL_OPENCURSOR%.
3. Les codes d'erreur que SQL_ERROR% renvoie en cas d'exécution de cette instruction peuvent être : -32003 ou -32005.
4. L'instruction SQL_CLOSECURSOR ne doit pas être utilisée avec le module IM de NatStar.
5. Nat System vous recommande d'une façon générale de privilégier l'utilisation de l'instruction SQL_CLOSETHECURSOR.

```
; ---- 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 .... ; utilise le curseur par défaut
```

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

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 NSW2PGxx 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. SQL_OPENTHECURSOR% permet de résoudre ce problème car tous les ordres SELECT FETCH seront alors exécutés sur ce nouveau curseur.
6. 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.
7. 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.
8. Lors d'une ouverture simultanée de plusieurs bases de données, le curseur ouvert par SQL_OPENTHECURSOR% est immédiatement associé à la base courante.
9. 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%.

```
;  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 .... ; utilise le curseur par défaut
```

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 .... ; utilise le curseur par défaut
```

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

Paramétrer le comportement du SGBD

NS_FUNCTION ANSIOFF, ANSION

Les deux NS_FUNCTION ANSIOFF et ANSION ont été créées pour pallier au problème suivant : la commande "SQL_EXEC UPDATE … WHERE …" fixe SQL_ERROR% à 0, même s'il n'y a aucun enregistrement.

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

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

1. ANSION 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 ASYNCOFF, ASYNCON

La NS_FUNCTION ASYNCOFF permet de positionner le mode asynchrone à OFF (valeur par défaut). En mode synchrone, l'application cliente reste en attente de la réponse de fin de traitement du serveur avant de reprendre la main.
La NS_FUNCTION ASYNCON permet de positionner le mode asynchrone à ON. En mode asynchrone, l'application peut continuer d'autres traitements en attendant la fin des traitements du serveur.

Syntaxe	**NS_FUNCTION ASYNCOFF** et **NS_FUNCTION ASYNCON**

1. En mode ASYNCHRONE, quel que soit le temps d'exécution d'une requête, l'utilisateur n'attend pas : il reprend immédiatement "la main" après le lancement de l'exécution. A l'inverse, dans le mode SYNCHRONE, il faut attendre la fin de l'exécution d'une requête pour avoir "la main".
2. Le mode d'exécution considéré par défaut est le mode SYNCHRONE.
3. La clause INTO :HOST_VARIABLE n'est pas supportée en mode ASYNCHRONE. L'exécution d'un ordre SELECT...INTO ou FETCH...INTO renvoie l'erreur : NSSQLE081 DATA NOT READY TO RESULT PROCESSING

```
SQL_EXEC NS_FUNCTION ASYNCOFF ; le mode d'exécution devient SYNCHRONE
SQL_EXEC NS_FUNCTION ASYNCON ; le mode d'exécution devient ASYNCHRONE
```

NS_FUNCTION DESCRIBEOFF, DESCRIBEON

Ces deux fonctions sont obsolètes. PostgreSQL détecte automatiquement le mode d'appel (implicite ou explicite) aux procédures ou aux fonctions stockées. Si vous utilisez déjà ces deux fonctions dans votre code, il est inutile de les retirer, mais sachez qu'elles n'ont aucune incidence.

Sert à spécifier le mode d'appel aux procédures ou aux fonctions stockées.

Syntaxe	**NS_FUNCTION DESCRIBEON** et **NS_FUNCTION DESCRIBEOFF**

1. DESCRIBEOFF est le mode par défaut.
2. Dans le mode DESCRIBEOFF, la description des paramètres est implicite pour l'utilisateur, mais entraîne une demande de 'describe' de la procédure ou de la fonction sur le serveur où est stockée la procédure ou la fonction.
3. Dans le mode DESCRIBEON, la description des paramètres de la fonction ou de la procédure doivent être faite explicitement par l'utilisateur et ceci de manière identique à celle réalisée lors de la création de la procédure ou de la fonction. Cette technique apporte un gain de performance important, puisqu'il n'est plus nécessaire de demander un 'describe' de la procédure ou de la fonction au serveur.

```
; ------------------------------------------------
; Exemple d'appel à une fonction en mode DESCRIBEOFF
; ------------------------------------------------
SEGMENT SELT
NOM$(5)
ENDSEGMENT
SEGMENT STB
SELT SEG[25]
ENDSEGMENT

LOCAL I% ; Récupère la valeur de retour de la fonction
LOCAL A$(10) ; Paramètre scalar
LOCAL ELT_TB$(5) ; Paramètre array : Fournit la description
; d'un élément du tableau de handle HTB%
LOCAL C% ; Paramètre scalar
LOCAL HTB% ; Handle sur le tableau
LOCAL NBROW% ; Nombre de lignes dans le tableau
...
SQL_EXEC NS_FUNCTION DESCRIBEOFF
NBROW%=25
SQL_EXEC NS_FUNCTION SETBUFFERSIZE :NBROW%
New STB , HTB%
... Remplissage du tableau (2ème paramètre en IN)
; ---- Appel de la fonction
SQL_EXEC SQL_PROC :I% = PACKAGE1.FUNCT1(:A$,ELT_TB$,C%)VALUES (:HTB%)
...
; ------------------------------------------------
; Exemple d'appel à une procédure en mode DESCRIBEON
```

```
; ------------------------------------------------
Segment SELT
NOM$(5)
ENDSEGMENT
SEGMENT STB
SELT SEG[25]
ENDSEGMENT
LOCAL F$(10) ; Paramètre scalar
LOCAL ELT_TB$(5) ; Paramètre array : Fournit la description
; d'un élément du tableau de handle HTB%
LOCAL NB% ; Paramètre scalar
LOCAL HTB% ; Handle sur le tableau
LOCAL NBROW% ; Nombre de lignes dans le tableau
...
SQL_EXEC NS_FUNCTION DESCRIBEON
NBROW%=25
SQL_EXEC NS_FUNCTION SETBUFFERSIZE :NBROW%
NEW STB , HTB%
F$='DUPONT'
NB%=0
; ---- Appel de la procédure PROCFILTER dont le rôle est :
; . - filtrer les noms commençant par F$
; . - remplir le tableau avec les enregistrements filtrés
; . - affecter NB% avec le nombre d'enregistrement trouvé
SQL_EXEC SQL_PROC PACKAGE1.PROCFILTER(:F$ IN [VARCHAR2,0],:ELT_TB$ OUT
[VARCHAR2,1],:NB% INOUT [NUMBER ,0])VALUES (:HTB%)
;... Après l'appel le tableau a été rempli par la procédure
; la variable NB% a été modifiée
```

NS_FUNCTION IMAGEOFF, IMAGEON

Le mode IMAGEON permet de manipuler des objets binaires (de type bitmaps par exemple) de taille limitée à 32 000 octets. Cette manipulation s'effectue dans le SEGMENT NCL SQL_IMAGE défini dans NSDMS.NCL.
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 SQL_IMAGE :

```
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 avec SQL_EXEC_LONGSTR (types Type pour les blobs 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".

Il n'est pas possible d'utiliser ou de créer une table avec plus d'une colonne de type LONG RAW ou LONG.

```
;Exemple
;création de la table
SQL_EXEC CREATE TABLE T_IMAGE(NUMERO NUMBER(8), DESCRIPTION VARCHAR2(80),
IMAGE LONG RAW)
if sql_error% <> 0
  message 'error Create' , sql_errmsg$(sql_error%)
endif

;INSERT
LOCAL DEST$(80),DATA%,SIZE%(4),NBREAD%(2),FILE%,NIL%,FNAME$, hbmp%
LOCAL SQL_IMAGE LOCALIMAGE
SQL_EXEC NS_FUNCTION IMAGEON
IF SQL_ERROR% <> 0
 MESSAGE 'IMAGEON', SQL_ERROR% && SQL_ERRMSG$(SQL_ERROR%)
 RETURN 1
ENDIF
FNAME$ = "(NS-BMP)\TINTIN.BMP"
HBMP%=CREATEBMP%(FNAME$)
BMPT = HBMP%
```

```
;FGETSIZE% does not accept environnement variables
FNAME$ = "D:\TESTS\BMP\TINTIN.BMP"

SIZE%=FGETSIZE%(FNAME$)

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
; ---- Insert into table t_image
LOCALIMAGE.REALSIZE = SIZE%
LOCALIMAGE.LENGTH% = SIZE%
LOCALIMAGE.PTR% = DATA%
SQL_EXEC INSERT INTO T_IMAGE VALUES (1,'Une île entre le ciel et l ''eau',
\ :LOCALIMAGE)
IF SQL_ERROR% <> 0
 MESSAGE 'INSERT IMAGE', SQL_ERROR% && SQL_ERRMSG$(SQL_ERROR%)
 F_CLOSE FILE%
 DISPOSE DATA%
 RETURN 1
ENDIF
F_CLOSE FILE%
DISPOSE DATA%
; SELECT
LOCAL DEST$(80),DATA%,SIZE%(4),NBREAD%(2),FILE%,NIL%,FNAME$, hbmp%
LOCAL SQL_IMAGE LOCALIMAGE
SQL_EXEC NS_FUNCTION IMAGEON

IF SQL_ERROR% <> 0
 MESSAGE 'IMAGEON', SQL_ERROR% && SQL_ERRMSG$(SQL_ERROR%)
 RETURN 1
ENDIF
LOCALIMAGE.realsize = 30000
NEW LOCALIMAGE.realsize,LOCALIMAGE.PTR%
SQL_EXEC SELECT IMAGE INTO :LOCALIMAGE FROM T_IMAGE WHERE NUMERO = 1
IF SQL_ERROR% <> 0
 MESSAGE 'SELECT IMAGE',SQL_ERROR% && SQL_ERRMSG$(SQL_ERROR%)
ELSE
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%
; ---- default mode
SQL_EXEC NS_FUNCTION IMAGEOFF
```

Voir aussi NSDBMS.NCL, SQL_ERROR%, SQL_ERRMSG$, Type pour les blobs TYPE_SQL_SELECT_BLOB

NS_FUNCTION QUOTEOFF, QUOTEON

Le mode QUOTEON gère automatiquement les guillemets pour les paramètres passés en entrée dans les valeurs de type chaîne de caractères. Cette fonctionnalité est pratique pour effectuer des substitutions de chaînes au cours de l'exécution d'une application.
Le mode QUOTEOFF désactive ce mode.

Syntaxe	**NS_FUNCTION QUOTEOFF** et **NS_FUNCTION QUOTEON**

1. Le mode QUOTEON est le mode actif par défaut.
2. Les guillemets devront être saisis par l'utilisateur dans le mode QUOTEOFF et seront gérés automatiquement dans le mode QUOTEON.

```
MOVE 3 TO mode_cursor%
SQL_EXEC NS_FUNCTION SETCURSORMODE :mode_cursor%
```

```
;Exemple
LOCAL PERE$, ID% ,FILS$, IND1%, IND2%, ID%, A$
LOCAL CSTRING Req$(2000)
LOCAL CURSORMODE%

CURSORMODE% = DB_POSTGRESQL_CURSORNONE ;3
SQL_EXEC NS_FUNCTION SETCURSORMODE :CURSORMODE%

SQL_EXEC NS_FUNCTION QUOTEOFF

A$ ='"'&'PIERRE'&'"'
SQL_EXEC SELECT ID, PERE, FILS INTO :ID%, :PERE$:IND1%, :FILS$:IND2% FROM
TOTO WHERE PERE =:A$
IF SQL_ERROR% <> 0
    MESSAGE "Erreur ",SQL_ERRMSG$(SQL_ERROR%)
ENDIF

SQL_EXEC NS_FUNCTION STATEMENT INTO :Req$
INSERT AT END Req$ TO LISTBOX1
; this way we can trace the value of A$ ("PIERRE" here)
; SELECT ID, PERE, FILS FROM TOTO WHERE PERE ="PIERRE"

WHILE SQL_ERROR% = 0
 INSERT AT END PERE$  TO LISTBOX1
 SQL_EXEC FETCH
ENDWHILE
SQL_EXEC NS_FUNCTION QUOTEON
CURSORMODE% = DB_POSTGRESQL_CURSORDEFAULT ;0
SQL_EXEC NS_FUNCTION SETCURSORMODE :CURSORMODE%
A$ ='PIERRE'
SQL_EXEC SELECT ID, PERE, FILS INTO :ID%, :PERE$:IND1%, :FILS$:IND2% FROM
TOTO WHERE PERE =:A$
IF SQL_ERROR% <> 0
    MESSAGE "Erreur ",SQL_ERRMSG$(SQL_ERROR%)
```

```
 ENDIF

; All the variables have to be initialized even if their value will be null
SQL_EXEC NS_FUNCTION STATEMENT INTO :Req$
INSERT AT END Req$ TO LISTBOX1
; SELECT ID, PERE, FILS FROM TOTO WHERE PERE =@Param5
WHILE SQL_ERROR% = 0
 INSERT AT END PERE$ TO LISTBOX1
 SQL_EXEC FETCH
ENDWHILE
```

Voir aussi NSDBMS.NCL, SQL_ERROR%, SQL_ERRMSG$

NS_FUNCTION SETBUFFERSIZE

Spécifie la taille du buffer en nombre de lignes dans le cas des ARRAY FETCH et dans le cas des procédures stockées qui utilisent des paramètres de type tableau.

Syntaxe	**NS_FUNCTION SETBUFFERSIZE** *:taille-buffer*			
Variable	taille-buffer	INT(4)	I	taille du buffer

1. Par défaut, taille-buffer vaut 1. Dans le cas où une des colonnes sélectionnées est de type LONG, la taille par défaut est toujours à 1.
2. La valeur maximum varie en fonction des colonnes sélectionnées et de la taille mémoire autorisée par PostgreSQL pour gérer les tableaux (ARRAYs). Les paramètres ARRAYSIZE et MAXDATA permettent d'ajuster l'utilisation de cette mémoire (reportez-vous aux manuels PostgreSQL pour de plus amples informations).
3. Cette NS_FUNCTION optimise les transferts réseaux en permettant d'augmenter le nombre de lignes transférées lors d'un FETCH.
4. Un curseur par défaut est ouvert automatiquement après le SQL_OPEN. Ne pas oublier de positionner un SETBUFFERSIZE.
5. Une requête utilisant des BLOB ou des CLOB nécessitent un buffer de taille 1.

```
;Exemple 1
;Soit les segments suivants :
;segment S_COL1
;        int V_COL1
;endsegment

;segment S_COL2
;     cstring V_COL2(10)
;endsegment

;segment S_COL1_S
; S_COL1 SEG[10]
;endsegment

;segment S_COL2_S
        ;S_COL2 SEG[10]
;endsegment
LOCAL COL1%
LOCAL COL2$(10)
LOCAL HCOL1%, HCOL2%
Local int     I%(4)

sql_exec drop table JYM
SQL_EXEC CREATE TABLE JYM ( COL1 INTEGER , COL2 VARCHAR2(10) )
if sql_error% <> 0
  message 'error' , sql_errmsg$(sql_error%)
endif

; ---- Create segments
NEW S_COL1_S  , HCOL1%
NEW S_COL2_S  , HCOL2%
```

```
  S_COL1_S(HCOL1%).SEG[0].V_COL1 = 1
  S_COL2_S(HCOL2%).SEG[0].V_COL2 = 'Albert%%%'
  S_COL1_S(HCOL1%).SEG[1].V_COL1 = 2
  S_COL2_S(HCOL2%).SEG[1].V_COL2 = 'Bernard%%'
  S_COL1_S(HCOL1%).SEG[2].V_COL1 = 3
  S_COL2_S(HCOL2%).SEG[2].V_COL2 = 'DANIEL%%%'
  S_COL1_S(HCOL1%).SEG[3].V_COL1 = 4
  S_COL2_S(HCOL2%).SEG[3].V_COL2 = 'ROGER%%%%'

     Move 4 to I%
     SQL_EXEC NS_FUNCTION SETBUFFERSIZE :I% ;
 if sql_error% <> 0
   message 'error' , sql_errmsg$(sql_error%)
 endif

     SQL_EXEC INSERT INTO JYM (COL1, COL2) \
                DESCRIBE (:COL1%, :COL2$) \
                VALUES (:HCOL1%, :HCOL2%)
    if sql_error% <> 0
      message 'error' , sql_errmsg$(sql_error%)
    endif

dispose HCOL1%
dispose HCOL2%

;Exemple 2
;SEGMENT SNAME
;  CSTRING NAME(11)
;ENDSEGMENT

;SEGMENT BUF_NAME
  ;  SNAME EMP[10]
;ENDSEGMENT
sql_execstr 'CREATE OR REPLACE PACKAGE PACK_TEST AS\
    TYPE namec_arr is table of emp.ename%type index by binary_integer;
PROCEDURE IN_OUT(NAMEC_ARRAY OUT NAMEC_ARR, FETCH_SIZE IN OUT INTEGER);\
 cursor c_in_out is select ename from emp;\
END PACK_TEST;'
if sql_error% <> 0
  message 'error PACKAGE' , sql_errmsg$(sql_error%)
endif

sql_execstr 'CREATE OR REPLACE PACKAGE BODY PACK_TEST AS PROCEDURE
IN_OUT(NAMEC_ARRAY OUT NAMEC_ARR, FETCH_SIZE IN OUT INTEGER) IS \
counter binary_integer;\
begin\
  if not c_in_out%isopen then\
    open c_in_out;\
  end if;\
  counter := 1;\
  loop\
    fetch c_in_out into namec_array(counter);\
    exit when c_in_out%notfound or counter = fetch_size;\
    counter := counter + 1;\
  end loop;\
  if c_in_out%notfound then \
    fetch_size := counter - 1;\
```

```
    close c_in_out;\
  end if;\
end;\
end pack_test;'

if sql_error% <> 0
  message 'error BODY' , sql_errmsg$(sql_error%)
endif
;Puis
LOCAL CSTRING Ename(11)
LOCAL Nbr_Rows%, i%
LOCAL h_nos%,h_names%,h_names2%,h_dats%,curs%
MOVE 10 TO Nbr_Rows%
NEW BUF_NAME, h_names%
MOVE SQL_OPENCURSOR% TO CURS%
SQL_EXEC NS_FUNCTION SETBUFFERSIZE :Nbr_Rows% USING CURS%
; Bug 6765
SQL_EXEC SQL_PROC pack_test.in_out (:Ename OUT [VARCHAR2, 1],:Nbr_Rows% IN
[NUMBER, 0]) VALUES (:h_names%) USING CURS%
IF SQL_ERROR% <> 0
    Message 'error', SQL_ERROR%&': '&SQL_ERRMSG$(SQL_ERROR%)
ENDIF
MOVE 0 TO i%
WHILE i% < Nbr_Rows%
 INSERT AT END 'Longueur'&&length BUF_NAME(h_names%).EMP[i%].NAME&&
'Valeur='&BUF_NAME(h_names%).EMP[i%].NAME TO LISTBOX1
 MOVE i%+1 TO i%
ENDWHILE
SQL_CLOSECURSOR
DISPOSE h_names%
```

NS_FUNCTION TRIMCHAROFF, TRIMCHARON

Dans le mode TRIMCHARON, lors d'un SELECT les blancs en fin de chaîne sont supprimés. Il est surtout utile lorsque la colonne de la table est 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$, B$
; longstr is a varchar2(2000)
;   &      TEST CHAR(10)
SQL_EXEC INSERT INTO TOTO (ID, TEST, LONGSTR) VALUES (1, 'A234567890', \
'lgstr23456789')
if sql_error% <> 0
  message 'error INSERT' , sql_errmsg$(sql_error%)
endif
SQL_EXEC INSERT INTO TOTO (ID, TEST, LONGSTR) VALUES (2, 'A2345', \
'lgstr2345    ')
if sql_error% <> 0
  message 'error INSERT' , sql_errmsg$(sql_error%)
endif
SQL_EXEC INSERT INTO TOTO (ID, TEST, LONGSTR) VALUES (3, 'A',
'lgstr        ')
if sql_error% <> 0
  message 'error INSERT' , sql_errmsg$(sql_error%)
endif
; Default mode
; ----- This loop will show <A234567890>
; <A2345              >
; <A                       >
; <lgstr23456789>,
;<lgstr234            >
;<lgstr                  >
SQL_EXEC SELECT TEST, LONGSTR FROM TOTO
if sql_error% <> 0
  message 'error SELECT' , sql_errmsg$(sql_error%)
endif
INSERT AT END '{J[C],B[LIGHTRED]}DEFAULT' TO LISTBOX1
 WHILE SQL_ERROR% = 0
 SQL_EXEC FETCH INTO :C$, :B$
 if sql_error% <> 0
  BREAK
 endif
 INSERT AT END '{B[YELLOW]}CHAR=<' & C$ & '>'&'' TO LISTBOX1
 INSERT AT END 'VARCHAR=<' & B$ & '>'&'' TO LISTBOX1
ENDWHILE
; ----- mode
```

```
SQL_EXEC NS_FUNCTION TRIMCHARON
if sql_error% <> 0
  message 'error TRIMCHARON' , sql_errmsg$(sql_error%)
endif
; -----  This will diplay  <A234567890>
; <A2345>
; <A>
; And <lgstr23456789>,
;<lgstr234>
;<lgstr>
INSERT AT END '{J[C],B[LIGHTRED]}TRIMCHARON' TO LISTBOX1
SQL_EXEC SELECT TEST, LONGSTR FROM TOTO
if sql_error% <> 0
  message 'error SELECT' , sql_errmsg$(sql_error%)
endif
WHILE SQL_ERROR% = 0
   SQL_EXEC FETCH INTO :C$, :B$
 if sql_error% <> 0
  BREAK
 endif
 INSERT AT END '{B[YELLOW]}CHAR=<' & C$ & '>'&'' TO LISTBOX1
 INSERT AT END 'VARCHAR=<' & B$ & '>'&'' TO LISTBOX1
ENDWHILE
; ---- Retour au mode par défaut
SQL_EXEC NS_FUNCTION TRIMCHAROFF
if sql_error% <> 0
  message 'error TRIMCHAROFF' , sql_errmsg$(sql_error%)
endif
```

Voir aussi SQL_ERROR%, SQL_ERRMSG$

Gérer la base de données courante

NS_FUNCTION CHANGEDBCNTX

Positionne une base de données comme base courante.

Cette fonction a été développée pour gérer plusieurs bases de données simultanément.

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

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

```
;Exemple
LOCAL LOGICALDBNAME$

SQL_OPEN "BASE1","scott/tiger@MYPOSTGRES"
SQL_EXEC .... ; BASE1 is the current database

SQL_OPEN "BASE2","scott/tiger@MYACCESS"
SQL_EXEC .... ; BASE2 is the current database

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

SQL_CLOSE "BASE1"
SQL_EXEC .... ; BASE2 is the current database

SQL_CLOSE "BASE2"
```

Voir aussi SQL_OPEN, SQL_CLOSE

NS_FUNCTION GETDBNAME

Récupère le nom logique de la base de données courante.

Syntaxe	**NS_FUNCTION GETDBNAME INTO** *:nom-physique-base*			
Paramètre	nom-physique-base	CSTRING	I/O	nom physique de la base de données courante

```
;Exemple
LOCAL DBNAME$,DBNAME2$

SQL_OPEN "BASE1","SCOTT/TIGER@T:MACHINE:SERV1"

SQL_OPEN "BASE2","SCOTT/TIGER@SERVICE_VENTE"

SQL_EXEC NS_FUNCTION GETDBNAME INTO :DBNAME$
MESSAGE 'LA BASE DE DONNEES COURANTE EST :',DBNAME2$
; Affiche le nom logique BASE2

SQL_CLOSE "BASE2"
SQL_EXEC NS_FUNCTION GETDBNAME INTO :DBNAME$
MESSAGE 'LA BASE DE DONNEES COURANTE EST :',DBNAME$
; Affiche le nom logique BASE1
```

***Voir aussi** SQL_OPEN, SQL_CLOSE, NS_FUNCTION CHANGEDBCNTX*

NS_FUNCTION LISTDBS

Retourne la liste de bases de données existante sur le serveur.

Syntaxe	**NS_FUNCTION LISTDBS** *[:filtre$]*			
Paramètre	filtre$	CSTRING	I	paramètre optionnel, qui peut contenir les caractères de recherche tel que le %

Si le paramètre filtre$ n'est pas défini, l'ensemble des bases est renvoyé.

```
;Exemple
local filtre$, db$

filtre$ = 'My%' ;toutes les bases commençant par 'My'
sql_exec ns_function listdbs :filtre$
while (sql_error%=0)
 sql_exec fetch into :db$
 message 'base', db$
endwhile
```

NS_FUNCTION LISTTABLES

Retourne la liste des tables de la base en cours.

Syntaxe	**NS_FUNCTION LISTTABLES** *[:filtre$]*			
Paramètre	filtre$	CSTRING	I	paramètre optionnel, qui peut contenir les caractères de recherche tel que le %.

Si le paramètre filtre$ n'est pas défini, l'ensemble des bases est renvoyé.

```
;Exemple
local filtre$, table$

filtre$ = 'T_%' ;toutes les tables commençant par 'T_'
sql_exec ns_function listtables :filtre$
while (sql_error%=0)
  sql_exec fetch into :table$
  message 'table', table$
endwhile
```

NS_FUNCTION LISTCOLUMNS

Retourne la liste des colonnes d'une table de la base en cours.

Syntaxe	**NS_FUNCTION LISTCOLUMNS** *:table$ [,:filtre$]*			
Paramètres	table$	CSTRING	I	une table de la base
	filtre$	CSTRING	I	paramètre optionnel, qui peut contenir les caractères de recherche tel que le %

Si le paramètre filtre$ n'est pas défini, l'ensemble des bases est renvoyé.

```
;Exemple
local filtre$, table$, col$

table$ = 'MyTable'
filtre$ = 'C_%' ;toutes les colonnes commençant par 'C_'
sql_exec ns_function listcolumns :table$, :filtre$
while (sql_error%=0)
   sql_exec fetch into :col$
   message 'col', col$
endwhile
```

Gérer le serveur PostgreSQL

NS_FUNCTION GETSTATUS

Retourne une chaîne de caractère avec le statut du serveur PostgreSQL après la dernière requête exécutée (équivalent à ce que retourne l'exécutable PostGresadmin).

Syntaxe	**NS_FUNCTION GETSTATUS INTO** *:status$*			
Paramètre	status$	CSTRING	I	statut du serveur PostgreSQL

```
;Exemple
local status$
sql_exec ns_function getstatus into :status$
if (sql_error% = 0)
 message 'PostgreSQL status', status$
else
 message 'status error', sql_error%&&sql_errmsg$(sql_error%)
endif
```

Gérer les blobs

Type pour les blobs TYPE_SQL_SELECT_BLOB%

Ce mode vous permet de manipuler des objets binaires de taille supérieure à 32000 octets, mais dont la taille reste cependant limitée par le SGBD.

Commentaires

1. Un type de données NCL du fichier NSDBMS.NCL est à affecter dans le champ Type_Var de la structure NCLVAR : TYPE_SQL_SELECT_BLOB%. Il doit être utilisé pour récupérer dans un fichier binaire donné un binaire stocké dans la base.
2. La sélection d'images avec TYPE_SQL_SELECT_BLOB% est illimitée.
3. Le mode du curseur doit être positionné à 3.

```
;Exemple
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 "Erreur ",SQL_ERRMSG$(SQL_ERROR%)
 ENDIF
SQL_EXEC NS_FUNCTION Statement INTO :Req$
IF SQL_ERROR% <> 0
    MESSAGE "Erreur ",SQL_ERRMSG$(SQL_ERROR%)
 ENDIF
Insert AT END Req$ TO LISTBOX1
; Création de la table
SQL_EXEC CREATE TABLE BIGIMAGE(NUMERO INTEGER, DESCRIPTION VARCHAR(255),
COLIMAGE IMAGE)
IF SQL_ERROR% <> 0
    MESSAGE "Erreur ",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$="INSERT INTO BIGIMAGE (NUMERO, DESCRIPTION, COLIMAGE) VALUES ( 1,'This
is a big picture > 32000 bytes', : )"
SQL_EXEC_LONGSTR @SQL$, @HL, DEFAULT_CURSOR
IF SQL_ERROR% <> 0
    MESSAGE "Erreur ",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$="SELECT COLIMAGE INTO : FROM BIGIMAGE WHERE NUMERO = 1"
SQL_EXEC_LONGSTR @SQL$, @HL, DEFAULT_CURSOR

IF SQL_ERROR% <> 0
    MESSAGE "Erreur ",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%
```

Voir aussi SQL_ERROR%, SQL_ERRMSG$

INDEX