

# **NatStar**

---

Version 3.00 Edition 1

# **NS-DK**

---

Version 3.00 Edition 1

# **NatWeb**

---

Version 4.00 Edition 1

# **NS-DBR User Guide and Programming Manual**

*Information in this document is subject to change without notice as a result of changes in the product. The software described in this document is furnished under a license agreement. The software may be used or copied only in accordance with the terms of the agreement. It is illegal to reproduce the software on any medium unless specifically authorized within the terms of the agreement. No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose other than the purchaser's personal use without the written permission of Nat System.*

© 2004 Nat System. All rights reserved.

The Nat System name and logo, NatStar / Adaptable Development Environment, NatStar / Graphical Builder, NatStar / Information Modeling and NatStar / Process Modeling are registered trademarks of Nat System.

All other trademarks mentioned in this document are registered trademarks of their respective companies.

Ref. no NSR300DRX00001

# Contents

About this manual .....	xi
Organization of the Manual .....	xi
Conventions .....	xiii

## **Chapter 1 ..... Introduction**

Overview of NS-DB's Existing Functionality .....	1-3
Objectives of the Upgrade from NS-DB to NS-DBR .....	1-4
Schematic Overview of NS-DBR .....	1-5

## **Chapter 2 ..... Functional Overview**

Scope of NS-DBR .....	2-3
Using NS-DBR in a Distributed Architecture .....	2-4
PREFETCH: Optimizing Network Access .....	2-5
Defining the NS-DBR Working Database in the Initialization File .....	2-6
Some Definitions .....	2-7
Transaction .....	2-7
COMMIT .....	2-7
ROLLBACK .....	2-7
Cursor .....	2-8
Starting a transaction .....	2-8
Stopping a transaction .....	2-8
Isolating Transactions and Visibility of Updated Information .....	2-8
Index Clustering .....	2-9
Optimizing Performance .....	2-11
Clustering .....	2-11
Unclustered Database .....	2-11
Database Clustered by NCL .....	2-12
Database Clustered for Optimum Access Times .....	2-13
Using NSQBE .....	2-17
Creating a Virtual Database .....	2-18

## **Chapter 3..... Initialization File**

NSDB.INI Initialization File .....	3-3
Syntax for the Initialization File .....	3-4
[SYSTEM] Group .....	3-4
DBDIR Verb .....	3-4
SYSID Verb .....	3-5
JOURNAL Verb .....	3-7
[USER] Group .....	3-7
Macro Definitions .....	3-8
Database definitions .....	3-8
[END] Group .....	3-10
Sample .INI file That Uses NETBIOS .....	3-10
Examples of Multi-User Implementation .....	3-11
Sample initialization files for multi-user, Client/Server, Standalone mode .....	3-11
Sample initialization files for remote Client/Server mode .....	3-12
Sample initialization file that uses TCP/IP .....	3-14

## **Chapter 4..... SQL Syntax**

Data Description Language .....	4-3
Features of the DDL .....	4-3
Identifiers .....	4-3
Types .....	4-3
DDL Reference .....	4-4
Data Manipulation Language .....	4-24
Features of the DML .....	4-24
Identifiers .....	4-24
Declaratives .....	4-26
Purpose and Usage .....	4-26
VIEW Declarative .....	4-26
View Properties .....	4-27
Methods Associated with a View .....	4-27
DML Reference .....	4-29

## **Chapter 5..... A Simple Application**

Introduction .....	5-3
Features of the Data Description Language .....	5-3
Updating the NSDB.INI Initialization File .....	5-4

Describing the Data.....	5-5
Creating a Database .....	5-5
Loading the Library .....	5-5
Connecting to the Database .....	5-6
Creating a Table and an Index .....	5-6
Manipulating the Data.....	5-9
Inserting a Row into a Table .....	5-9
Searching a Table .....	5-10
Projecting All Columns .....	5-10
DISTINCT Clause and Functions .....	5-10
Retrieving Data in Host Variables.....	5-11
Scanning Table Rows .....	5-11
Updating a Table Row .....	5-12
Joins .....	5-12
Arranging the Results.....	5-14
ORDER BY .....	5-14
Modifying the Database Structure .....	5-15
Adding a Column .....	5-15
Adding and Building an Index .....	5-15

## **Chapter 6 .....SYSTEM CATALOG**

Introduction.....	6-3
Objects in the Catalog .....	6-5
Describing Objects in Another Language .....	6-6
Example.....	6-6
Details on the Contents of the Catalog .....	6-8
SYSTABLES Structure.....	6-8
SYSCOLUMNS Structure.....	6-8
SYSINDCOLS Structure.....	6-9
SYSINDEXES Structure .....	6-10
SYSSPACES Structure .....	6-11
SYSDATASPACES Structure .....	6-11

## Chapter 7 ..... Interface with NCL

## Chapter 8 ..... NS-DB Interface for 3GLs (C, Pascal)

Overview of the Interface .....	8-4
NS-DB API Entry Point.....	8-5
Z_COM Communication Area.....	8-6
ZIO Input-Output Area .....	8-8
REQ Request Area .....	8-9
Fx Function Code.....	8-11
DB_INIT Function .....	8-11
DB_TERM Function .....	8-11
DB_STARTRN Function.....	8-11
Call Arguments .....	8-12
DB_OPENCURSOR Function.....	8-12
Allocating a Cursor Implicitly .....	8-13
Allocating a Cursor Explicitly .....	8-13
DB_CLOSECURSOR Function.....	8-13
DB_STOPTRN Function.....	8-13
DB_COMPILE Function.....	8-13
DB_COMMIT Function.....	8-14
DB_ROLLBACK Function.....	8-14
DB_ISRT Function.....	8-14
DB_DLET Function .....	8-14
DB_SEARCH Function.....	8-14
DB_NEXT Function.....	8-15
DB_PREV Function .....	8-15
DB_MODIF Function .....	8-15
DB_GETFIRSTDB Function .....	8-15
DB_GETNEXTDB Function .....	8-15
DB_DROPDB Function .....	8-15
Accessing the Catalog Tables .....	8-16
Transactional Journaling.....	8-17
Definition.....	8-17
Implementation.....	8-17
Technical Description.....	8-17
Definition of a transaction .....	8-17

Definition of COMMIT	8-17
Definition of ROLLBACK	8-17
Starting a Transaction	8-17
Stopping a Transaction	8-18
Isolating Transactions and Visibility of Updated Information	8-18

## Chapter 9 .....NS-DB Utilities

DBCHCK Utility .....	9-4
Start-up syntax.....	9-4
Description .....	9-4
DBSERV Utility .....	9-7
Start-up syntax.....	9-7
Description .....	9-7
Details on network operation	9-8
Concurrent access control	9-9
Differences between network and standalone mode	9-10
Automatic suffixing of internal names	9-10
DCLGEN Utility .....	9-13
Start-up syntax.....	9-13
Description .....	9-13
Example.....	9-13
EXPORT Utility.....	9-14
Start-up syntax.....	9-14
Description .....	9-14
Example.....	9-14
IMPORT Utility .....	9-15
Start-up syntax.....	9-15
Description .....	9-15
Example.....	9-16
REMSRV Utility .....	9-17
Start-up syntax.....	9-17
Description .....	9-17
Example.....	9-17
SHUTDOWN Utility .....	9-18
Start-up syntax.....	9-18
Description .....	9-18
Examples .....	9-18

SQLLEX Utility .....	9-19
Command syntax .....	9-19
Description .....	9-19
TCPSRVDB Utility.....	9-20
Start-up Syntax .....	9-20
Description .....	9-20
Example.....	9-20

## **Chapter 10..... NSQBE Utility**

Overview .....	10-7
Components of NSQBE .....	10-8
Main Window .....	10-8
Title Bar .....	10-8
Menu Bar .....	10-8
Icon Bar .....	10-9
Data Base List Dialog Box .....	10-9
Utilities Menu .....	10-10
Exit Menu .....	10-11
<i>DatabaseName Transaction N</i> Dialog Box .....	10-11
Menu Bar .....	10-12
Rows Menu .....	10-12
Tables Menu .....	10-13
Exit Menu .....	10-14
<i>Store Rows on DatabaseName.TableName</i> Dialog Box .....	10-14
Columns .....	10-15
Left/Right .....	10-15
Store .....	10-15
Cancel .....	10-15
<i>Selection on DatabaseName.TableName</i> Dialog Box .....	10-16
Table columns .....	10-16
Projections .....	10-16
Selections... ..	10-17
Selection predicate .....	10-17
Remove .....	10-17
Projected columns .....	10-17
Selection .....	10-17
<i>Selection Result</i> Dialog Box .....	10-17
Selection Information .....	10-18
Selected list .....	10-19
<i>DatabaseName.TableName</i> Dialog Box .....	10-19
Table Name .....	10-20
Data storage .....	10-20



Column Info	10-20
Indexes	10-21
Column List	10-21
Create	10-21
Alter	10-21
Drop	10-22
Cancel	10-22
<i>TableName.IndexName</i> Dialog Box .....	10-22
Index Name	10-23
Index Info	10-23
Column List / Indexed Columns	10-23
Foreign Key	10-24
Create	10-24
Drop	10-24
Cancel	10-24
<i>New Index definition on TableName</i> Dialog Box .....	10-24
Index Name	10-25
Index Info	10-25
Column List / Indexed Columns	10-26
Indexed Columns	10-26
Foreign Key	10-26
Create	10-27
Drop	10-27
Cancel	10-27
<i>DatabaseName.NewTable</i> Dialog Box .....	10-28
Table Name	10-28
Data storage	10-28
Column Info	10-28
Indexes	10-29
Column List	10-29
Create	10-29
Alter	10-30
Drop	10-30
Cancel	10-30
<i>DCLRES</i> Dialog Box .....	10-30
NCL Language	10-30
C Language	10-30
Pascal Language	10-30
Save as	10-31
Cancel	10-31
<i>Import Tables from IXF Format</i> Dialog Box .....	10-31
File Name	10-32
Current Directory	10-32
Files	10-32
Directories	10-32
Import	10-32

Cancel	10-32
<i>Import from Drive:\Path\Fname.IXF</i> Dialog Box.....	10-32
Table Name	10-33
Storage location	10-33
Primary Key Election	10-33
OK	10-34
Cancel	10-34
<i>Export Table Using IXF/PC Format</i> Dialog Box.....	10-34
File Name	10-35
Current Directory	10-35
Files	10-35
Directories	10-35
Export	10-35
Cancel	10-35
<i>Monitor Information</i> Dialog Box .....	10-35
Monitor information	10-36
OK	10-36
Cancel	10-36
<i>Monitor Activity</i> Dialog Box .....	10-36
Global Activity	10-37
Active transactions	10-37
Server activity	10-38
Refresh	10-38
Cancel	10-38
<i>Transaction N on DatabaseName</i> Dialog Box.....	10-38
Counts	10-39
Total time	10-39
Avg time	10-39
Max time	10-39
Cancel	10-39
<i>Server Activity</i> Dialog Box.....	10-39
Total calls to server	10-40
Total calls to database	10-40
Total Elapse in database	10-41
Max Elapse in database	10-41
Avg Elapse in database	10-41
Total Elapse in server	10-41
Max Elapse in server	10-41
Avg Elapse in server	10-41
Current number of clients	10-41
Max number of clients in session	10-41
Using NSQBE .....	10-42
Opening or Creating a Database.....	10-42
Checking a Database for Physical Errors .....	10-42
Checking a Database for Logical Errors.....	10-43

Starting a Transaction.....	10-43
Adding a Row to a Table.....	10-44
Commit and Rollback.....	10-44
Executing a Query .....	10-45
Projecting Columns .....	10-45
Selecting a Search Predicate .....	10-46
Creating a Table .....	10-47
Creating a DATASPACE (NS-DB V.02) .....	10-49
Creating a TABLESPACE .....	10-49
Dropping a Table.....	10-49
Adding a New Column to a Table.....	10-50
Adding an Index .....	10-51
Generating Data Structures in NCL, C and Pascal .....	10-52
Importing a Table .....	10-52
Exporting a Table .....	10-53
Displaying Monitor Activity .....	10-54
Displaying Server Activity .....	10-55

## **Appendix A ..... NS-DBR Error Messages**

General Usage Guide .....	A-3
Description.....	A-4



## About this manual

When NS-DBR is integrated with your Nat System applications, it extends the NS-DB database manager with relational functions.

NS-DBR therefore provides an additional method for accessing the NS-DB engine.

You can now access NS-DB databases using SQL. This language is incorporated in the NS-SQL library, together with the DBM, ORACLE and SYBASE database managers.

There is no need to migrate your existing databases: NS-DBR adapts immediately to your existing NS-DB environment.

You can use the IMPORT utility to import external tables from IBM's relational DBMSs: DATABASE MANAGER and DB2.

## Organization of the Manual

This manual contains 10 chapters and an appendix:

<b>Chapter 1</b>	<b>Introduction</b> Presents the objectives of the evolution of NS-DB toward NS-DBR.
<b>Chapter 2</b>	<b>Functional Overview</b> Explains the general principles for using NS-DBR.
<b>Chapter 3</b>	<b>Initialization File</b> Describes the NS-DBR initialization file.
<b>Chapter 4</b>	<b>SQL Syntax</b> Describes SQL syntax.
<b>Chapter 5</b>	<b>A Simple Application</b> Presents a rapid description of how to begin using NS-DBR with an NCL example.
<b>Chapter 6</b>	<b>System Catalog</b> Describes the objects that are stored in the System Catalog.

<b>Chapter 7</b>	<b>Interface with NCL</b> See the "Service Libraries - volume 2", which describe in detail how to access NS-DB using NCL language.
<b>Chapter 8</b>	<b>NS-DB Interface for 3GLs (C, PASCAL...)</b> Describes the interfaces between programs and NS-DB.
<b>Chapter 9</b>	<b>NS-DB Utilities</b> Explains the various NS-DB utilities.
<b>Chapter 10</b>	<b>NSQBE Utility</b> Describes the NSQBE utility, which lets you access and manipulate NS-DB database information in a user-friendly and generic manner.
<b>Appendix A</b>	<b>NS-DBR Error Messages</b> Describes NS-DB error messages and codes that adhere to SQL norms.

## Conventions

### Typographic conventions

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

### Notational conventions

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

### Operating conventions

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

## Icon Codes



Indicates specific information on using the software under DOS-Windows (all versions).



Indicates specific information on using the software under DOS-Windows 3.x (16 bits).



Indicates specific information on using the software under DOS-Windows 32 bits.



Indicates specific information on using the software under OS/2-PM (all versions).



Indicates specific information on using the software under OS/2-PM version 1.3 and later (16 bits).



Indicates specific information on using the software under OS/2-PM version 2.x (32 bits).



Indicates specific information on using the software under OS/2-PM 1.x or DOS-Windows 3.x (16 bits).



Indicates specific information on using the software under OS/2-PM 2.x or DOS-Windows 32 bits.



Indicates specific information on using the software under Macintosh.



## Chapter 1

# Introduction



### ***This chapter explains***

- Overview of NS-DB's Existing Functionality.
- Objectives of the Upgrade from NS-DB to NS-DBR.
- Schematic Overview of NS-DBR.

## Contents

Overview of NS-DB's Existing Functionality.....	1-3
Objectives of the Upgrade from NS-DB to NS-DBR .....	1-4
Schematic Overview of NS-DBR .....	1-5

## Overview of NS-DB's Existing Functionality

The current mechanism used by the NS-DB database manager has two main characteristics:

- Non-navigational

The programmer cannot select the index used to access data.

Depending on the specified query, NS-DB determines the optimum path for accessing the information.

NS-DB makes this choice dynamically according to the indexes available in the catalog when the query is invoked.

As a result, the programmer only manipulates logical structures. The database administrator can add physical structures (indexes) designed to speed up data access for existing process, independently of any programming.

- No set operations:

Each call to the NS-DB programming interface can only manipulate one instance of an object in the database.

The programmer needs to code a loop to update a set of related data items.

Set operations cannot be used to join several structures.

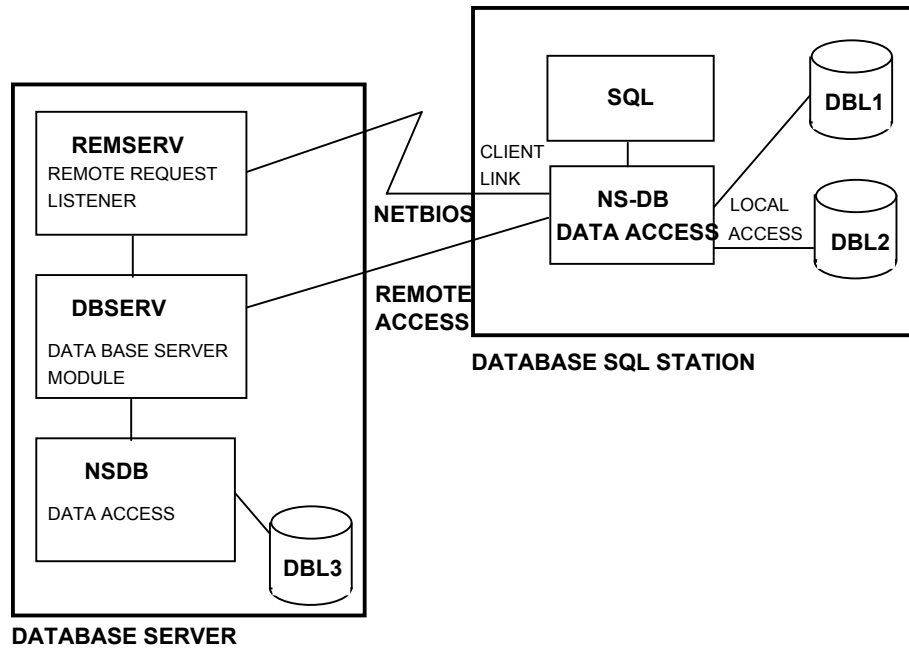
---

## **Objectives of the Upgrade from NS-DB to NS-DBR**

The upgrade to the programming interface has several objectives:

- 1.** To comply with SQL Data Definition Language (DDL) standards, defined by the IBM SAA, in order to improve the product's portability.
- 2.** To ensure that this DDL provides a standard method for managing referential integrity.
- 3.** To provide the NS-DBR programming interface with a Data Manipulation Language (DML) that allows data to be accessed using set operations, whenever this access mode seems appropriate.
- 4.** To ensure upward compatibility by enabling the same NCL program to access both the SQL programming interface and the existing NS-DB programming interface (which does not support set operations) according to the programmer's needs.

## Schematic Overview of NS-DBR





## Chapter 2

# Functional Overview



### ***This chapter explains***

- Scope of NS-DBR.
- Using NS-DBR in a Distributed Architecture.
- Defining the NS-DBR Working Database in the Initialization File.
- Some Definitions.
- Optimizing Performance.

## Contents

Scope of NS-DBR.....	2-3
Using NS-DBR in a Distributed Architecture.....	2-4
PREFETCH: Optimizing Network Access.....	2-5
Defining the NS-DBR Working Database in the Initialization File .....	2-6
Some Definitions .....	2-7
Transaction	2-7
COMMIT	2-7
ROLLBACK	2-7
Cursor	2-8
Starting a transaction	2-8
Stopping a transaction	2-8
Isolating Transactions and Visibility of Updated Information	2-8
Index Clustering	2-9
Optimizing Performance .....	2-11
Clustering	2-11
Unclustered Database	2-11
Database Clustered by NCL	2-12
Database Clustered for Optimum Access Times	2-13
➤ Example 1	
➤ Example 2	
➤ Example 3	
Using NSQBE	2-17
Creating a Virtual Database	2-18



## Scope of NS-DBR

NS-DBR is an SQL add-on for NS-DB.

- The NSnnSQL library complies with the same usage and implementation rules as the existing SQL interfaces in NCL:

Reminder:

DBMS	Libraries
DATA BASE MANAGER :	NSnnDBM
ORACLE :	NSnnORA
SYBASE/SQL SERVER :	NSnnSYBA
XDB :	NSnnXDB
NS-DBR :	NSnnSQL

- The programming interfaces for database access are described in the *NCL* and *Libraries* manuals (NSnnSQL library).



nn stands for the version number of the interface that you have installed.

*Example*

The following command loads NS-DBR:

```
SQL_EXEC SQL_INIT..... ; Comply with the SQL syntax
                        ; statements shown in this manual
```

- NSnnSQL loads the standard NSDB libraries, namely:

Libraries	Approximate memory requirement
NSnnDB	150 Kb
NSnnSQL	130 Kb

## Using NS-DBR in a Distributed Architecture

NS-DB can manage one or more local or remote databases.

NS-DBR is even more suitable for highly distributed structures; it allows you to define transactions that affect one or more local or remote databases by synchronizing multi-database COMMIT and ROLLBACK requests.

By installing the SQL engine on each station, which is possible due to the small amount of code, you can:

- Distribute your intermediate SQL processes more efficiently (e.g. ORDER BY); they will be executed using the station's CPU resources.
- Ensure that your remote servers only execute **input/output** operations on the database.

This strategy prevents the server from crashing due to several stations running a large number of complex SQL operations on the network.



Client/server implementations are detailed in the Libraries Programming Manual (NSDB Indexed Sequential Database Library). See Chapter 3 for details on how to implement multi-user mode.

## PREFETCH: Optimizing Network Access

1. The average time taken to transfer a buffer on a LAN (Local Area Network) is 50 - 100 milliseconds. However, this is much higher than the transfer time for local buffered data: 6 - 8 milliseconds.

To optimize transfer times for FETCH statements on remote databases, NS-DBR allows you to transfer rows in packets using a buffer associated with a remote database in the initialization file.

The size of this buffer can range from 2048 to 4096 bytes.

*Example: REMDB='SERV',REMOTE,BUFSIZE=2048 ;*

2. In this example, a PREFETCH buffer comprising 2048-byte packets is associated with the remote database. This means that the system will transmit the rows retrieved by a SELECT in groups when the FETCH command is executed. This significantly improves the performance of an application.
3. If a database is never updated by client stations, it may be advantageous to allow the network server to share its file and to inform NS-DB that all the processes on the network can access the database in read-only mode.

In this case, the READONLY parameter must be specified whenever the database is defined in the initialization file on a client station..

*Example: MYDB='X:\READ\TABLES',CASEINSENS, READONLY ;*

---

## Defining the NS-DBR Working Database in the Initialization File

NS-DB uses a TEMPORARY database to execute a number of relational operations.

This database is automatically added to the initialization file as follows:

```
[USER]  
SQLWKDB='Drive:\Path\<Fname>.<Ftype>';
```

SQLWKDB is a reserved logical database name used by NS-DBR. Obviously, for performance reasons, it cannot be a remote database.

This database must be added to the initialization file before you use NS-DBR for the first time, otherwise the following message will be displayed when NS-DBR initializes:

```
* -37 * INIT SQL WORK DB ERROR SQLEX -093
```

**Cause:** Unable to open the working database, i.e. UNKNOWN DATABASE ID (SQLWKDB has not been defined in the initialization file).



To implement multi-user client/server mode, you will need to define several temporary working databases - one for each NSDB PROCESS run by a user. The first one will be SQLWKDB, followed by SQLWKDB1- SQLWKDB9 (cf. Initialization file: Chapter 3).

## Some Definitions

### Transaction

A very precise definition of a transaction has been given by C-J DATE: UNIT OF WORK ON A DATABASE.

Hence, a transaction is a set of logically related instructions sent to a specific database.

The notion of a "transaction" is distinct from that of a program.

The same program can execute several transactions on the same database.

However, a transaction is the only entity recognized by the database integrity functions (COMMIT and ROLLBACK).

- When the COMMIT statement is executed, all the update operations performed by the transaction that requested the COMMIT are applied to the database and are visible to all other concurrent transactions.
- When the ROLLBACK statement is executed, it cancels all the update operations performed by the transaction that requested the ROLLBACK since the last COMMIT.

### COMMIT

When the COMMIT statement is issued, all the updates performed by a transaction since the last COMMIT or ROLLBACK statement or since the start of the transaction are permanently applied to the database (DO).

The updates are then known as "committed updates".

### ROLLBACK

When the ROLLBACK statement is issued, all the updates performed by a transaction since the last COMMIT or ROLLBACK statement or since the start of the transaction are removed from the database (UNDO).

---

## Cursor

A cursor is a database resource used to maintain the current position reached by a SELECT or FETCH statement.

One cursor must be defined for each:

- Position to be simultaneously maintained in the database.
- Simultaneous current position.

## Starting a transaction

The START TRANSACTION statement starts a transaction and allows you to access an object in a database. (This statement can be compared with **opening a database**).

The same PROCESS can start several transactions on one or more databases.

## Stopping a transaction

When a transaction is stopped, a COMMIT or ROLLBACK is automatically executed depending on how it was stopped. The following events stop a transaction:

- For a COMMIT statement:
  - Explicit STOP TRANSACTION request
  - Termination of the PROCESS that started the transaction: DOSEXIT (normal termination).
- For a ROLLBACK statement:
  - Explicit STOP TRANSACTION request
  - Termination of the PROCESS that started the transaction: TRAP CODE, CTRLC, etc. (abnormal termination)

## Isolating Transactions and Visibility of Updated Information

The database manager isolates and protects transactions.

- Rules for visibility and accessibility

The INSERT...INTO..., UPDATE and DELETE statements indicate an intention to update data. As a result, the system locks the physical pages containing the rows affected by the update.

If a table row is being updated by a transaction, the page that contains it is reserved by the transaction until the next COMMIT or ROLLBACK requested by this transaction. After this, the resource will be freed.

- Dealing with unstable, uncommitted updates

If a transaction updates information in the database and another transaction wants to read this information at the same time, there are three possible strategies:

- The system can allow access to the data, bearing in mind the risk that the update being performed by the first transaction may be incomplete, in which case the information read will be incorrect.

N.B.: This is the default option chosen by the system since, statistically, transactions are more likely to succeed than fail.

- The system prohibits access to the data being modified and returns an error code.
- The system returns the information stored on disk (opposite risk to first strategy).
- Abnormal termination of the server:

In addition to any problems regarding the physical stability of the journal, abnormal termination by the server has the following consequences:

- All the server transactions in progress that have not been committed are declared as IN FLIGHT (unstable).
- All IN FLIGHT transactions will be rolled back and any update operations in progress will be lost when the system is rebooted.

## Index Clustering

**Index clustering** (using CLUSTERING NUMBERS) is a mechanism designed to optimize data access times.

The response times for selections made on a database with unclustered tables can increase very quickly. These times are proportional to the size of the database.

The aim is therefore to arrange indexes with hierarchical dependencies (e.g. PRIMARY KEYS and FOREIGN KEYS) in contiguous physical storage areas and make a physical distinction between each contiguous index.

These "distinguishing markers" take the form of **CLUSTERING NUMBERS**.

In a clustered database, searches will be performed in the physical page that contains the indexes grouped together by their CLUSTERING NUMBER.

There clustering numbers are assigned to indexes. You must specify:

- The PRIMARY KEY INDEX when you use CREATE TABLE.
  - The SECONDARY INDEXES when you use CREATE INDEX.
-

For a composite index:

- The first field must be clustered.
- The secondary fields can be clustered if required.

CLUSTERING NUMBERS range from 0 - 254.

These CLUSTERING NUMBERS are stored in the SYSINDCOLS table in the system catalog (CLUTOP and ICLUBOT columns).

Although clustering is not compulsory (the absence of clustering numbers does not result in an error), it is strongly recommended.



# Optimizing Performance

## Clustering

We will use an example to illustrate the issue of clustering by observing what happens in:

- An unclustered database.
- A database clustered by NCL.
- A database clustered for optimum access times.

### Unclustered Database

An unclustered database is always searched sequentially.

In actual fact, if no clustering numbers have been defined, the engine sets a default of **255**.

As a result, each INDEX in the unclustered database will contain this number,

*Example:*

```
SQL EXEC CREATE TABLE CLIENT      \
(
  CLIENT_ID INTEGER NOT NULL ,      \
  CLIENT_NAME CHAR(30),             \
  PCLIENT_NAME CHAR(30),            \
)
PRIMARY KEY ICLIENT
SQL EXEC CREATE TABLE ARTICLE     \
(
  ART ID INTEGER NOT NULL,           \
  ART NAME CHAR(30),                 \
  PRICE NUM(8),                      \
)
PRIMARY KEY IARTICLE
```

Since the primary keys have not been clustered by the user, each one is assigned a default clustering number of **255**.

There is no device to differentiate between:

- Keys of the same type.
- The same clustering number.
- And so on.

This results in physical collisions and extremely poor performance.

---

These records are physically arranged as follows.

<b>255</b> CLIID	CLIENT DATA....
------------------	-----------------

<b>255</b> ARTID	ARTICLE DATA....
------------------	------------------

<b>255</b> 1	Jones	Peter
<b>255</b> 1	Sneakers	69.00
<b>255</b> 2	Smith	Paul
<b>255</b> 2	Tennis racket	98.80
<b>255</b> 3	Washington	George
<b>255</b> 3	Shorts	25.00

A search performed on clustering number **255** will be sequential. The engine cannot differentiate between similar keys. This increases the time taken by the search.

You may choose not to cluster your database if you only want to carry out a series of tests on a very small database. However, performance soon deteriorates as the database expands.

### Database Clustered by NCL

In NCL, tables are created with the segment editor, which **automatically** assigns CLUSTERING NUMBERS.

The CLUSTERING NUMBER is automatically incremented by 1 once it has been assigned to an index. This is the system used by the NSQBE utility.

This clustering method is the simplest to implement.

It results in highly acceptable performance levels.

**Example:**

```
SQL_EXEC CREATE TABLE CLIENT      \
(                                     \
  CLIENT_ID INTEGER NOT NULL,       \
  CLIENT_NAME CHAR(30),             \
  PCLIENT_NAME CHAR(30),           \
)                                     \
PRIMARY KEY ICLIENT (1 CLIID 1)
```

```
SQL EXEC CREATE TABLE ARTICLE \
( \
  ART_ID INTEGER NOT NULL, \
  ART_NAME CHAR(30), \
  PRICE NUM(8), \
) \
PRIMARY KEY IARTICLE (2 ARTID 2)
```

The primary keys are clustered by NCL: 1 CLIID 1, 2 ART\_ID 2.

<b>1 ClientID 1</b>	CLIENT DATA....
---------------------	-----------------

<b>1 1 1</b>	Jones	Peter
<b>1 2 1</b>	Smith	Paul
<b>1 3 1</b>	Washington	George

<b>2 ArticleID 2</b>	ARTICLE DATA....
----------------------	------------------

<b>2 1 2</b>	Sneakers	69.00
<b>2 2 2</b>	Tennis Racket	98.80
<b>2 3 2</b>	Shorts	25.45
<b>2 4 2</b>	Socks	5.90

A search is performed on the clustered indexes and collisions never occur. Keys with the same type are differentiated by their CLUSTERING NUMBER, which means that the search can immediately locate the key specified in the search command.

## Database Clustered for Optimum Access Times

This is the most complex of all clustering methods but it is also the most efficient.

It involves examining the relationships between your tables and using clustering numbers to group together indexes that are frequently used together in queries.

Two indexes that are frequently used in conjunction with each other will have the same CLUSTERING NUMBER.

➤ **Example 1**

```

SQL_EXEC CREATE TABLE CLIENT      \
(
  CLIID INTEGER NOT NULL , \
  CLI_NAME CHAR(30) , \
  PCLI_NAME CHAR(30) , \
) \
PRIMARY KEY ICLIENT (1 CLIID 1)

SQL_EXEC CREATE TABLE INVOICE      \
(
  INVID INTEGER NOT NULL , \
  CLIID INTEGER NOT NULL , \
  DESCR(30) , \
  AMOUNT(8) , \
) \
PRIMARY KEY IINVOICE (1 CLIID 1, 1 INVID 1)

```

An invoice can only be recorded if it can be associated with a client.

A client can have several invoices.

This is the most efficient physical arrangement:

1 CLIID 1	CLIENT DATA....
-----------	-----------------

1 CLIID 1	1 INVID 1	INVOICE DATA...
-----------	-----------	-----------------

1 2 1	Jones	Peter
-------	-------	-------

1 1 1	1 1 1	Electricity	50.62
1 1 1	1 2 1	Phone	95.45

1 2 1	Smith	Paul
-------	-------	------

1 2 1	1 3 1	Electricity	234.00
1 2 1	1 4 1	TV	155.00

1 2 1	1 5 1	Auto insur.	305.65
-------	-------	-------------	--------

And so on for client 3, client 4, etc.

A search is performed on the Client ID so that we can access all the related information directly.

However, this method has one **limitation** that can be observed when searching for all clients whose ID is greater or equal to N: the system has to sequentially read through all the invoices associated with the client

### ➡ Example 2

Let's add the ZipCode field to the Client table and index it. This will cater for the fact that a client may live in different areas..

Since the ZipCode is not closely related to the Invoice table, its clustering number will need to be distinct from the ones selected for the primary key and for the Invoice table. This will prevent collisions during a search, which would have an adverse effect on performance.



#### General rule

An autonomous secondary index, i.e. one which is not logically related to any other indexes, must be assigned a unique number.

If you wish, you can reserve a range of clustering numbers for this purpose. You can, for example, reserve all numbers in the range [120..145] to cluster this type of index.

Type the following instruction:

```
CREATE INDEX I1 ON CLIENT PRIMARY KEY (2 ZIPCODE 2 )
```

The data will be physically arranged as follows:

1 ClientID 1	2 INVID 2	CLIENT DATA...
--------------	-----------	----------------

1 ClientID 1	1 INVID 1	INVOICE DATA...
--------------	-----------	-----------------

**➤ Example 3**

```
SQL_EXEC CREATE TABLE CLIENT      \
(
  \
  CLIID INTEGER NOT NULL , \
  CLI_NAME CHAR(30) , \
  PCLI_NAME CHAR(30) , \
  \
)
PRIMARY KEY ICLIENT (1 CLIID 1)

SQL_EXEC CREATE TABLE INVOICE      \
(
  \
  INVID INTEGER NOT NULL , \
  CLIID INTEGER NOT NULL , \
  DESCR(30) , \
  AMOUNT(8) , \
  \
)
PRIMARY KEY IINVOICE (2 CLIID 2, 3 INVID 3)
```

An invoice can only be recorded if a client can be associated with it.

A client can have several invoices.

The most efficient physical arrangement is as follows:

<b>1 CLIID 1</b>	CLIENT DATA...
------------------	----------------

1 1 1	Jones	Peter
1 2 1	Smith	Paul
1 3 1	Washington	George

<b>2 INVID 2</b>	INVOICE DATA...	CLIENTID
------------------	-----------------	----------

<b>2 1 2</b>	Electricity	50.62	CliID1
<b>2 2 2</b>	Tel	95.45	CliID2
<b>2 3 2</b>	Electricity	23.00	CliID3
<b>2 4 2</b>	TV	155.00	CliID4

2 5 2	Auto insur.	305.65	ClIID5
...	...	...	...

3 CLIID 3	2 INVID 2
3 1 3	2 1 2
3 1 3	2 2 2
3 2 3	2 3 2
3 2 3	2 4 2
3 2 3	2 4 2

If a join is made to retrieve the description of an invoice for client X, the client will initially be searched for in the last physical page shown above.

The pointer to this page searches for the client, finds the PRIMARY KEY and sends a pointer to the physical page containing PRIMARY KEY INVID as the first field followed by the INVOICE DATA.

This arrangement occupies more disk space than the previous one and access times are increased, although not always noticeably. Since the CLIID and INVID data items are duplicated on several physical pages, this method guarantees security if a pointer is lost in one of the physical pages.

## Using NSQBE

When you create indexes, NSQBE tells you which CLUSTERING NUMBERS have not yet been used anywhere in the database.

NSQBE allows you to cluster your databases as in NCL.

If you discover that your indexes are unclustered, you can rectify this problem at any time with the aid of NSQBE. In this case, you need to export your tables, delete the existing database and then import the tables. The import operation allows you to modify the CLUSTERING NUMBERS.



Only the primary key is kept after an import. Any secondary keys need to be recreated and clustered via the *<Get Cluster>* button

## Creating a Virtual Database

You can define a virtual database to improve the performance of your SQL statements. You must specify that the database is virtual when you define it in the NSDB.INI file.

A virtual database is defined as follows:

```
MYDB='\\VIRTUAL\\';
```

Transactional journalling must be set on (JOURNAL = YES).

Real databases and virtual databases have the same characteristics. The only difference is their location. A real database is located on disk, while a virtual database is located in read/write memory.

All SQL operations performed on a virtual database are faster than those executed on a real database.

However, all the data is lost when a virtual database is closed.

Hence, a virtual database defines a temporary working space.

The NS-DBR working databases (SQLWKDB) and TABLESPACES can also be virtual.

A virtual TABLESPACE can be created for:

- ◆ A real database
- ◆ A virtual database.

On the other hand, a virtual database cannot define real tablespaces. If an attempt is made to do so, NSDB overrides the definition by creating a virtual tablespace.

The following error message is returned when a virtual database is opened:

```
SQLEX+21 DATABASE IS VIRTUAL.
```

Example of defining a virtual database:

```
MYDB1='\\VIRTUAL\\';  
SQLWKDB='\\VIRTUAL\\';
```

Example of creating a virtual TABLESPACE:

```
CREATE TABLESPACE SPACE1 '\\VIRTUAL\\'
```



## Chapter 3

# Initialization File



***This chapter  
explains***

- NSDB.INI Initialization File.
- Syntax for the Initialization File.

## Contents

NSDB.INI Initialization File.....	3-3
Syntax for the Initialization File .....	3-4
[SYSTEM] Group	3-4
DBDIR Verb	3-4
➤ Syntax	
➤ Description	
SYSID Verb	3-5
➤ Syntax	
➤ Description	
JOURNAL Verb	3-7
➤ Syntax	
➤ Description	
[USER] Group	3-7
Macro Definitions	3-8
Database definitions	3-8
➤ Syntax	
➤ Description	
➤ Example	
[END] Group	3-10
Sample .INI file That Uses NETBIOS	3-10
Examples of Multi-User Implementation	3-11
Sample initialization files for multi-user, Client/Server, Standalone mode	3-11
➤ Initialization file: local server	
➤ Initialization file: client	
Sample initialization files for remote Client/Server mode	3-12
➤ Initialization file: remote server	
➤ Initialization file: local server	
➤ Initialization file: client	
Sample initialization file that uses TCP/IP	3-14
➤ Initialization file: remote server	
➤ Initialization file: client	

## **NSDB.INI Initialization File**

The environment parameter, `NS-DB=Drive:\Path\Filename.INI`, should specify the name of a text file containing a description of the NS-DB system.

This initialization file describes the structure of your database resources.

The NSDB.INI file will be interpreted by the NS-DB initialization routine.

## Syntax for the Initialization File

An initialization file can contain comments. As in C, a comment begins with `/*` and ends with `*/`.

*Example:*

```
/* This is a comment */
```

The NSDB.INI files contains three groups of information:

- SYSTEM
- USER
- END.

### [SYSTEM] Group

This group defines NS-DB's general run-time parameters (NS-DB directory, Client/Server mode, etc.).

#### DBDIR Verb

The DBDIR verb defines:

- The NS-DB working directory.
- The TRACE options.
- The MONITORING options.
- A number of general NSDB run-time parameters.

#### ➤ Syntax

```
DBDIR='Drive:\Path'  
[,TRACE='NO' | 'TrFile']  
[,MONITOR='NO' | 'MnName']  
[,MAXTRANS=1..1024]  
;
```

#### ➤ Description

- **DBDIR**

The DBDIR verb specifies the working directory, which must already exist.

During each initialization, the optimizer generates a temporary file: NSWK000X.TMP.

- **TRACE**

The TRACE parameter is normally used if an unexplained NS-DB bug occurs in your application. It defines a file.

- This file stores all calls to NS-DB.
- It is stored in the NS-DB directory.
- Its name cannot exceed 8 characters.

If an unexplained NS-DB error occurs, contact Nat System for full details on how to implement the TRACE features.

If TRACE='NO', the TRACE is disabled.

- **MONITOR**

Defines the name of the shared memory used by the NS-DB engine and the DBMON.EXE utility.

If MONITOR='NO', the MONITOR is disabled.

- **MAXTRANS**

Specifies the maximum number of transactions that NS-DB can execute on a node.

This number defaults to 50.

When DB\_OPEN or SQL EXEC START TRANSACTION is executed, NS-DB returns the message TOO MANY TRANSACTIONS (if the specified number of transactions is too low).

## SYSID Verb

The SYSID verb defines the internal database SERVER ID.

This verb must be set, even though it is only used for client/server access.

Each machine on the network must have a unique SYSID.

1. For NETBIOS:

The SYSID verb must be defined by a name on the network.

2. For TCP/IP:

The SYSID verb must specify the server name and port number. To communicate with the TCP/IP protocol, a process must define two items of information:

- The IP network address of the machine used to run it.
- The TCP address of the process on the machine.

The machine's network address (or IP address) is defined over 4 bytes in the following format: a.b.c.d.

This address identifies the machine across the entire network.

To simplify using the IP address, you can create aliases. These are defined in the *hosts* file:

---

```
192.12.0.42 unix srv
192.12.0.50 os2 srv
192.12.0.51 unix cli1
```

To access a remote process via TCP, you need to know the input port of the process that you want to access (on the remote machine identified by the IP address). This is provided by the process' TCP address or *port number*.

A TCP/IP address is therefore identified by the combination: *IP address, port number*.

The NSDB.INI initialization file uses the aliases defined in the *hosts* file instead of the IP addresses, i.e. *servername, portnumber*.

These IDs must be assigned by the systems engineer responsible for the network.

### ➤ Syntax

```
SYSID='NAME'|'SERVERNAME:PORT_NUMBER'
      [,NETDLL=NETWORK DLL NAME]
      [,NETLSTN='Drive:\Path\NetListener.EXE']
      [,MAXNAME=1..255]
      [,MAXCOMMAND=1..255]
      [,MAXSESSION=1..255]
      ;
```

### ➤ Description

#### • NETDLL

Identifies the name of the communications library used.

This information is essential if at least one of the databases specified in the .INI file is defined as REMOTE.

It tells the system the name of the network library used for client/server communication.

For NETBIOS, these libraries can be:

ACSNETB	supplied by IBM Communication Manager
NB30	supplied by Microsoft Lan Manager 2.0
ACSNETB	supplied by Microsoft Lan Manager 2.1
NBNETWR	supplied by NOVELL NETWARE
NS02NWR	supplied by Nat System

For TCP/IP, the name of the library is:

TCPIPDDL	supplied by IBM/TCP pour Windows.
----------	-----------------------------------

#### • NETLSTN

Used by the DBSERV utility (NSDB DATABASE SERVER) when it starts up.

It tells the DBSERV server the name of the communication module used to listen to NSDB queries transmitted by the network.

By default, the DBSERV utility assumes that it is a local server and can only receive NS-DB calls from CLIENTS located on the same machine.

- For **NETBIOS**, the listener name is REMSRV.EXE.

- For **TCP/IP**, the listener name is TCPSRVDB.EXE
- **MAXNAME, MAXCOMMAND, MAXSESSION**  
For NETBIOS, these three verbs:
  - Define the maximum number of:  
names  
commands  
NETBIOS sessions that can be used by the NETBIOS LISTENER and database server.
  - Are only required in a client/server context.

The default values used by DBSERV are:

MAXNAME=16  
MAXCOMMAND=16  
MAXSESSION=16

You may need to increase these values if the number of NETBIOS CLIENTS simultaneously connected to the database server is increased.

For TCP/IP, only the number of sessions is defined.

## JOURNAL Verb

The Journal verb activates transactional journaling.

### ➤ Syntax

**JOURNAL= YES | NO ;**

### ➤ Description

#### **JOURNAL**

If **YES**, transactional journaling is enabled.

If **NO**, transactional journaling is disabled.

## [USER] Group

The USER group is used to specify the location of databases and macros.

It lists:

- All databases opened by the applications on the machine.
- All macros (\$(xxxxx)=) used to create TABLESPACES or DATASPACEs (NSDB V.02).

## Macro Definitions

A macro can be used to specify a pathname as follows:

```
$(Mydir)='C:\Mydir\DIR1\';
```

Macros are used, when TABLESPACES or DATASPACEs are created, to help ensure that the database remains portable.

When the application is ported, the macros used to define the location of its DATASPACEs and TABLESPACEs must be updated in the initialization file used by the new storage device.

*Example:*

*Porting a database from the disk C:\MYDIR\DIR1 to the disk D:\YOURDIR\DIR2.*

*If the NSDB.INI file defines the \$(MYDIR) macro as follows:*

```
$(MYDIR)='C:\MYDIR\DIR1\';
```

*and a TABLESPACE is created using this macro:*

```
Create TABLESPACE space1 $(MYDIR)
```

*then, before moving the database to the D:\ drive, the pathname specified by the macro will need to be modified as follows:*

```
$(MYDIR)='D:\YOURDIR\DIR2\';
```

*This will dynamically update the location of the TABLESPACE.*



Macros can also be used to specify database locations.

## Database definitions

### ➤ Syntax

```
Logical_Database_Name='Location'['Macro_Name FName' ['\VIRTUAL\'  
[.REMOTE |TCPIP | LOCAL]  
[.CASESENS |CASEINSENS ]  
[.READONLY]  
[.BUFSIZE=buffer-size]  
;
```

### ➤ Description

- *Logical\_Database\_Name*

Specifies the name used to identify the database in an application. This name must be used by the application whenever it refers to the database. It cannot exceed 8 alphanumeric characters.



- *Location*

Specifies the physical location of the database's PRIMARY TABLESPACE. This may be a disk file or remote server. For NETBIOS, the location comprises:

- The database's pathname.
- The name of the server that contains the database, if it is defined on a remote server.

For TCP/IP, the location comprises:

- The database's pathname.
- The name of the server followed by the port number, if the database is defined on a remote server.
- Example:

```
'servername:portnumber'
```

- *Macro\_Name FName*

These items are defined as follows: *\$(Mydir)Mydb.dat*.

They specify the physical location of the database.

*\$(Mydir)* is the name of the macro that specifies the pathname of the *Mydb.dat* file.

*\$(Mydir)* will be fully substituted by the character string assigned earlier to the macro.



Macros are also defined in the USER group. Their syntax is described above.

- **\\VIRTUAL\\**

Uppercase only.

Defines a virtual database.



a virtual database can only be defined if transactional journalling is enabled (JOURNAL= YES).

- **READONLY**

This parameter is used in the initialization files on the client stations when you want the database to be shared in read-only mode by the clients on the network.

By including this declaration in the NSDB.INI file, you will optimize network access (cf. Chapter 2).

This parameter is incompatible with the parameters REMOTE and LOCAL

- **BUFSIZE**

Sets the size of a transfer buffer.

One way of optimizing transfer times on remote databases is to transfer packets of data in a buffer associated with the REMOTE database.

This parameter is particularly effective for a large number of successive FETCH statements (cf. "PREFETCH: Optimizing Network Access").

**➡ Example**

```
/* LOCAL SINGLE-USER DATABASE */
MYDB1='Drive:\Path\FName',CASESENS ;

/* REMOTE DATABASE ON OS/2 NETBIOS SERVER */
MYDB2='ServName',REMOTE ;

/* REMOTE DATABASE ON TCP/IP SERVER */
MYDB2='Servername:portnumber', TCPIP [bufsize=4096]

/* MULTI-USER DATABASE ON LOCAL SERVER */
MYDB3='SERVNAME',LOCAL ;
```

In the last case, the server is supported by the same station as the client. This definition is equivalent to using the command NS0XDBCL.SERVNAME directly in NS-DB, which allows MYDB3 to be shared locally.

```
/* MULTI-USER SERVER DATABASE IN READ-ONLY MODE */
MYDB4='C:\Path\FName', CASEINSENS, READONLY ;
```

This statement allows clients to access MYDB4 on the server in read-only mode.

**[END] Group**

This group should simply contain:

```
END_INIT ;
```

This command is the syntactic end of file marker for the .INI file, used by the NSDB analyzer.

**Sample .INI file That Uses NETBIOS**

```
/*=====*/
/* NATSYSTEM DATABASE initialization file */
/*=====*/
[SYSTEM]
DBDIR='D:\NSDBR';
SYSID='SERV1' ,
NETDLL='NB30' , /* NB30 => Microsoft Lan Man driver */
NETLSTN='D:\NSDBR\BIN\REMSRV.EXE' ;
JOURNAL=YES ;

[USER]
123456='D:\MYDB\MYDB1';
Mydb='D:\MYDB\MYDB',CASEINSENS;
Data='DBSERV',REMOTE;

SQLWKDB='D:\MYDB\TEMP'; /* Mandatory temporary database*/
/*SQLWKDB='\\VIRTUAL\\'; Virtual temporary database */
MYDB1='\\VIRTUAL\\'; /* Virtual database */
```

```
$(MACRO)='D:\MYDIR\DIR1\';  
Mydata='$(Macro)Mydata.dat';  
/* Mydata will be stored in D:\MYDIR\DIR1\ */  
  
[END]  
END_INIT ;
```

## Examples of Multi-User Implementation



To implement multi-user client/server mode, you will need to define several temporary databases, each of which will correspond to an NSDB process run by a user. The first database will be SQLWKDB, followed by SQLWKDB1 - SQLWKDB9.

### Sample initialization files for multi-user, Client/Server, Standalone mode

This type of implementation requires two initialization files: one for the server and one for the client. These files must be in different locations or have distinct names.

The DBSERV utility must use the server file instead of the client file.

To ensure that this is the case, the relevant file must be specified in the session used to run DBSERV. For example:

```
[C:\] SET NS-DB=C:\NSDBR\DB\NSDB.SRV  
[C:\] DBSERV
```

After this, the client file should be specified:

```
[C:\] SET NS-DB=C:\NSDBR\DB\NSDB.INI
```

#### ➤ Initialization file: local server

```
[SYSTEM]  
DBDIR='C:\NSDBR';  
SYSID='SERVNAME';  
JOURNAL=YES;  
  
[USER]  
MYDB1='C:\NSDBR\DB\MYDB.DAT';  
  
[End]  
END_INIT;
```

**➤ Initialization file: client**

```
[SYSTEM]
DBDIR='C:\NSDBR';
SYSID='CLINAME';
JOURNAL=YES;

[USER]
MYDB1='SERVNAME',LOCAL;
SQLWKDB='C:\NSDBR\TEMP';
SQLWKDB1='C:\NSDBR\TEMP1';
SQLWKDB2='C:\NSDBR\TEMP2';
...
SQLWKDB9='C:\NSDBR\TEMP9';

[End]
END_INIT;
```

**Sample initialization files for remote Client/Server mode**

This type of implementation requires a **local server** to be declared.

The SYSID verb must be unique for each machine.

If the network attempts to run the same process several times on the remote server, a NETBIOS error will occur

**'SCINIT 8013 or 8021\*\*\*CONNECTION FAILURE SQLEX-42'**

**Cause:** This message indicates that the SYSID verb for the client machine has already been identified by the network.

**Corrective action:** This problem can be solved by declaring an intermediate local server.

Therefore, three initialization files must be declared:

- One for the server on the remote machine.
- One for the local server (on the client station).
- One for the client on the client station.

The files on the client machine must be in different locations or have distinct names.

The local server file must be specified in the session used to run DBSERV on the client machine. For example:

```
[C:\] SET NS-DB=C:\NSDBR\DB\NSDB.SRV
[C:\] DBSERV
```

After this, the client file should be specified:

```
[C:\] SET NS-DB=C:\NSDBR\DB\NSDB.INI
```

**➤ Initialization file: remote server**

```
[SYSTEM]
DBDIR='C:\NSDBR';
SYSID='REMSERVNAME',
NETDLL='ACSNETB',
NETLSTN='C:\NSDBR\BIN\
        REMSRV.EXE';
JOURNAL=YES;

[USER]
MYDB1='C:\NSDBR\DB\MYDB.DAT';

[END]
END_INIT;
```

**➤ Initialization file: local server**

```
[SYSTEM]
DBDIR='C:\NSDBR';
SYSID='LOCSERVNAME',
NETDLL='ACSNETB';
JOURNAL=YES;

[USER]
MYDB1='REMSERVNAME', REMOTE;

[END]
END_INIT;
```

**➤ Initialization file: client**

```
[SYSTEM]
DBDIR='C:\NSDBR';
SYSID='CLINAME', NETDLL='ACSNETB';
JOURNAL=YES;

[USER]
MYDB1='LOCSERVNAME', LOCAL;
SQLWKDB='C:\NSDBR\TEMP';
SQLWKDB1='C:\NSDBR\TEMP1';
SQLWKDB2='C:\NSDBR\TEMP2';
...
SQLWKDB9='C:\NSDBR\TEMP9';

[END]
END_INIT;
```

## **Sample initialization file that uses TCP/IP**

### **➤ Initialization file: remote server**

```
[SYSTEM]
dbdir='D:\NSDB\SQL' ,
TRACE='NO' , MONITOR='DBMON' ,
MAXTRANS=512 ;

Sysid='DBSRV:2200' ,
NETDLL='TCPIPDLL',
NETLSTN='c:\NSDBR\TCPSRVDB.EXE' ;

Journal=YES ;

[USER]
MYDB='C:\nsdbr\db\MYDB.DAT'
END_INIT ;
```

### **➤ Initialization file: client**

```
[SYSTEM]
dbdir='D:\NSDB\SQL' ;

Sysid='CLINAME:2000' ,
NETDLL='TCPIPDLL';
Journal=YES ;

[USER]
MYDB='DBSERV:2200', TCPIP, BUFSIZE=4096;

END_INIT ;
```

## Chapter 4

# SQL Syntax



***This chapter  
explains***

- Data Description Language.
- Data Manipulation Language.

## Contents

Data Description Language .....	4-3
Features of the DDL	4-3
Identifiers	4-3
Types	4-3
DDL Reference	4-4
Data Manipulation Language .....	4-23
Features of the DML	4-23
Identifiers	4-23
Declaratives	4-25
Purpose and Usage	4-25
VIEW Declarative	4-25
➤ Syntax	
➤ Notes	
View Properties	4-26
➤ Fixed View Properties	
➤ Variable View Properties	
Methods Associated with a View	4-26
➤ DESCRIBE Method	
➤ FETCH Method	
➤ CLOSE Method	
➤ DELETEWHRCURRENT (*) Method	
➤ DELETE Method (*)	
➤ UPDATEWHRCURRENT Method (*)	
➤ UPDATE Method (*)	
➤ INSERT Method (*)	
DML Reference	4-28



# Data Description Language

The Data Description Language is commonly abbreviated to DDL.

## Features of the DDL

### Identifiers

There are several categories of identifiers, which are syntactically recognized as follows:

• <i>Identifier</i>	Lexical item beginning with an alphabetical character For example: ABCD , X_HGAH , T1252
• <i>Short_Identifier</i>	Identifier with a maximum length of 8 characters
• <i>Long_Identifier</i>	Identifier with a maximum length of 18 characters
• <i>Database_Name</i>	Short_Identifier
• <i>Macro_Name</i>	\$(.....) Identifier with a maximum length of 18 characters.
• <i>Sql_Name</i>	Long_Identifier
• <i>Table_Name</i>	Sql_Name
• <i>TableSpaceName</i>	Sql_Name
• <i>DataSpaceName</i>	Sql_Name
• <i>Index_Name</i>	Sql_Name
• <i>Column_Name</i>	Sql_Name
• <i>Clustering_Number</i>	1..254
• <i>Column_Definition</i>	Column_Name_Data_Type[NOT NULL]

### Types

The following DATA\_TYPES are manipulated and recognized syntactically:

• CHAR(n)	C character string (array of CHAR).
• VARCHAR(n)	PASCAL character string ( n<=255).
• VARGRAPHIC(n)	Variable length field with an undefined structure.
• VARGRAPHIC(n)	Fixed length field with an undefined structure.
• SMALLINT	16-bit binary field.
• INTEGER	32-bit binary field.
• SMALLINT	64-bit floating decimal field.

---

- SYSTIMESTAMP     NUM floating decimal field. This type has the following format:  
YYYYMMDDHHMMSS (year, month, day, hours, minutes, seconds) and is automatically set by NS-DB. This column contains the last modification date for each row.



The physical storage area occupied by variable length fields in the database (CHAR, VARCHAR, VARGRAPHIC) does not exceed their current length.  
A field defined as NULL only occupies one byte in the database when it contains a NULL value.

## **DDL Reference**


The commands described below are arranged in alphabetical order.

## ALTER TABLE Command

Alters the structure of an existing table by adding columns or defining a FOREIGN KEY INDEX.

**Syntax**                    **ALTER TABLE** *Table\_Name*  
                                 [ADD (*Column\_Definition* [.....*Column\_Definition*])] [*Foreign\_Key\_Definition*]

### Notes

1. The new columns are placed after the existing columns, i.e. they are added to the end of the table.
2. SYSTIMESTAMP columns can be added to control concurrent access (cf. Time Stamping, Chapter 9, DBSERV Utility).
3. When columns are added to a table that already contains occurrences, the new columns will be initialized with the following values:
  - NOT NULL columns will be initialized with binary zero values.
  - NULL columns will be set to NULL.
4.  NULL should be interpreted as 'VALUE CURRENTLY UNKNOWN' in the relational model.
5. The equivalent IBM, DBM and DB2 syntax, which only allows one column at a time to be added to a table, is slightly different from the one shown above.

### Example

```
ALTER TABLE CLIENT ADD  
( ADDRESS VARCHAR (50) )
```

## ALTER TABLE Command: FOREIGN\_KEY\_DEFINITION

Alters a table by defining a foreign key in the table.

**Syntax**                    **FOREIGN KEY** *Index\_Name Index\_Columns*  
                              [**TABLESPACE** *TableSpaceName*]  
                              [**REFERENCES** *Table\_Name* [**ON UPDATE RESTRICT**]  
                              [**ON DELETE** [**RESTRICT** | **CASCADE** | **SET NULL**]]

**Notes**

Details on referential integrity are provided in the description for the CREATE INDEX REFERENTIAL INTEGRITY command.

**Example**

```
ALTER TABLE INVOICE
FOREIGN KEY ICLIREF ( 4 CLIENTREF 4 ) TABLESPACE SPACE2
REFERENCES CLIENT
ON UPDATE RESTRICT
ON DELETE RESTRICT
```

## CREATE DATASPACE Command

### Syntax

```
CREATE DATASPACE DataSpaceName  
['Drive:\Path\Fname.typ'] | ['Macro_Name Fname.typ'] [DROP]
```

### Notes

1. A DATASPACE is a physical file structure designed to store the data belonging to a table.
2. DATASPACEs allow the user to isolate the storage area used for the data in a table from the storage area used for the primary key index (Cf. TABLESPACE).
3. The maximum size allowed for a DATASPACE is twice the maximum allowed for a TABLESPACE: 134 Mb \* 2 = 268 Mb.
4. A DATASPACE can only store data belonging to one table at a given time. A DATASPACE is associated with a table when the PRIMARY KEY is defined (CREATE TABLE).
5. A DATASPACE is a two-state object:
  - FREE: not currently storing a table.
  - BUSY: currently storing a table.
6. All DATASPACE definitions for a database are stored in the SYSDATASPACEs table in the database's SYSTEM CATALOG.
7. The maximum size allowed for a TABLESPACE is 268,431,360 bytes. This physical limit corresponds to the maximum size allowed for an NSDB table in the current version.
8. The DROP option physically deletes the file if it already existed on the disk when the CREATE DATASPACE command was called.

### 9. Portability

The location of a DATASPACE can be defined either statically or dynamically, depending on whether or not you want the database to be portable.

- A static location defines the full pathname of the DATASPACE (DRIVE:\PATH\FNAME.TYP), which makes it harder to port the database from one machine to another.
- On the other hand, a dynamic location, specified using predefined macros in the NSDB.INI file, makes the database more portable (Cf. Chapter 3, Initialization File).

When you move the database to another site, all you need to do is modify the NSDB.INI file by changing the pathname specified by the relevant macro.

*Example:*

*Porting a database from C:\MYDIR\DIR1 to D:\YOURDIR\DIR2.*

If the \$(MYDIR) macro has been defined in the NSDB.INI file as follows:

```
$ (MYDIR)='C:\MYDIR\DIR1\';
```

and a DATASPACE has been installed using this macro:

```
CREATE DATASPACE SPACE1 $(MYDIR)
```

to port the database to the D:\ drive, the pathname specified by the macro will need to be modified as follows:

```
$ (MYDIR)='D:\YOURDIR\DIR2\';
```

==> This modifies the location of the DATASPACE dynamically (Cf. Chapter 3, [USER] Group).

## 10. Organization/Disorganization

The disorganization level for a database is measured on a table-by-table basis.

Only tables whose rows are stored in a DATASPACE can be disorganized

A DATASPACE is said to be organized when:

- All the space freed by deletions and updates has been retrieved.
- All the records are arranged sequentially according to the table's PRIMARY KEY INDEX. Since the INDEX and DATA buffers are in the same sequence, performance is optimized for searches.

The REORG utility allows you to reorganize a database. It retrieves any unused space in a table resulting from deletions. It uses a temporary working area with a maximum size equal to the size of the reorganized TABLESPACE.

### Example

```
; STATIC, NON-PORTABLE LOCATION

CREATE DATASPACE SPACE1 'C:\NSDBR\DB\SPACE1.DAT' DROP

CREATE TABLE CLIENT
(
  CLIID INTEGER NOT NULL,
  INVID INTEGER,
  CLINAME VARCHAR (50)
)
PRIMARY KEY ICLIENT (1 CLIID 1)      TABLESPACE INDEX1
                                     DATASPACE SPACE1;

; DYNAMIC, PORTABLE LOCATION

CREATE DATASPACE SPACE1 '$(MYDIR)SPACE1.DAT' DROP

; $(MYDIR)='C:\NSDBR\DB\' has been defined in the NSDB.INI
;                                                                    file

CREATE TABLE CLIENT
(
  CLIID INTEGER NOT NULL,
  INVID INTEGER,
  CLINAME VARCHAR (50)
)
PRIMARY KEY ICLIENT (1 CLIID 1)      TABLESPACE INDEX1
                                     DATASPACE SPACE1;
```

## BUILD INDEX Command

Builds an index defined for an existing table.

**Syntax**                **BUILD INDEX** *Index\_Name*

**Notes**

1. This command must be used after defining a new index for an existing table with the CREATE INDEX command.
2. It builds the new index using the existing occurrences in the table.

**Example**

```
BUILD INDEX INAME
```

## CREATE INDEX Command

Creates an index in an NS-DB database.

**Syntax**                    **CREATE [UNIQUE] INDEX** *Index\_Name* **ON** *Table\_Name* **[PRIMARY KEY]**  
                              (*Index\_Column* [, *Index\_Column*..]) [*Index\_Referencial\_Integrity*]  
                              [ **TABLESPACE** *TableSpaceName* ] [ **DATASPACE** *DataSpaceName* ]

**Notes**

1. A UNIQUE index:
  - Ensures that the values in a column are unique.
  - Cannot be created if the column already contains duplicates.
2. If a UNIQUE indexed column is used in an INSERT clause, the error code and message "-005 DB DUPLICATE" will be displayed, indicating that the value being inserted already exists.
3. If UNIQUE is not specified, the index columns can contain duplicate values in the database.
4. The PRIMARY KEY verb, which is used to create a primary key, must be used with the CREATE TABLE command.
5. You can also use the PRIMARY KEY sub-command, which is part of the CREATE TABLE command and closer to the DB2 syntax.
6. The DATASPACE sub-command used with the CREATE TABLE command is only applicable to PRIMARY KEY INDEXES.

**Example**

```
CREATE INDEX I1 ON CLIENT PRIMARY KEY( 1 CLIID 1)

; is equivalent to:
PRIMARY KEY I1 ( 1 CLIID 1 )

; You can use either syntax interchangeably:
CREATE UNIQUE INDEX I2 ON CLIENT (2 CLINAME 2)

; The Client table cannot contain two clients with the same name.
```



## CREATE INDEX Command: INDEX\_COLUMNS

Defines the table columns that make up an index.

**Syntax**

```
(
    [Clustering_Number] Column_Name[ASC|DESC]
    [Clustering_Number][,...[Clustering_Number] Column_Name
    [ASC|DESC][Clustering_Number]]
)
```

### Notes

1. The CLUSTERING NUMBER is a mechanism designed to speed up data access by maintaining a close association between the logical and physical structure of the data.
2. Logically related data items are stored in contiguous physical areas.  
For example, by storing information on a client and details on the invoices for this client in contiguous physical areas, you can reduce the number of disk access operations required when this information is read sequentially.
3. Another advantage of clustering: if two tables are defined with identical key types, clustering numbers can be used to distinguish between them. This will reduce the number of redundant sequential reads caused by randomly arranged physical keys.  
For example, if Invoice ID and Client ID are both defined as INTEGER keys, a device is needed to differentiate Client 1 from Invoice 1.  
The device used will be the CLUSTERING BYTE, which will differ for each index and ensure that the keys are not combined within the same consecutive sequence:
 

1 NOCLI	=> Physiquement	0100000001
2 NOFAC	=> Physiquement	0200000001
4. A detailed definition of CLUSTERING NUMBERS is provided in Chapter 2.
5. ASC (default option) selects an ASCENDING sort order for the columns in the index.
6. DESC selects a DESCENDING sort order for the columns in the index.

### Example

```
CREATE INDEX ICLIINV ON CLIINVOICE
( 1 CLIID DESC , 3 INVID ASC 5 )
TABLESPACE SPACE1
```

## CREATE INDEX: INDEX\_REFERENTIAL\_INTEGRITY

Defines referential integrity for the table columns that form the index.

**Syntax**                    [[ON UPDATE RESTRICT]  
                               [ON DELETE *Index\_Name* [ CASCADE | RESTRICT | SET NULL]]

**Notes**

1. The FOREIGN KEY sub-command in the ALTER TABLE command can be used to express the same notion using an alternative syntax (IBM DB2 MODEL).
2. **Referential integrity** is a relational mechanism whereby any strong dependency constraints that exist between two database tables are checked by the DBMS rather than by each of the application programs that manipulate the tables.  
*For example, an invoice is related to one client only and cannot be created for a client that does not exist. In this case, it could be dangerous to delete a client from the client table without updating the client's invoices in the invoice table.*  
 Referential integrity can be enforced (for the Client / Invoice example) using the following verbs:

Verb	Results
ON UPDATE RESTRICT	The Client ID in the invoice table cannot be modified.
ON DELETE CASCADE	If the client is deleted, all corresponding invoices are automatically deleted.
ON DELETE SET NULL	The Client ID in the INVOICE table is set to NULL.

3. The advantage of defining referential integrity in the database is that a functional dependency constraint can be specified in a single location. This solution is simple, economical and open-ended.
4. For the time being, the ON DELETE CASCADE and ON DELETE SET NULL clauses are only supported syntactically.

**Example**

```
CREATE INDEX ICLIINV ON CLIINVOICE
( 1 CLIID DESC , 3 INVID ASC 5 )
ON DELETE ICLIINV RESTRICT
```

## CREATE TABLE Command

Creates a table in an NS-DB database

**Syntax** `CREATE TABLE Table_Name (Column_Definition [...,Column_Definition ])  
Primary_Key_Definition | Create_Index`

## Notes

The syntax analyzer requires the CREATE TABLE command to define the table's primary key together with the table. This is because NS-DB complies with the RS-8 relational rule (CODD RMV2: PRIMARY KEY FOR EACH BASE R\_TABLE), which stipulates this requirement.

### Example

```
CREATE TABLE CLIENT
(
  CLIID INTEGER NOT NULL ,
  CLINAME VARCHAR(50)
)
PRIMARY KEY ICLIENT ( 1 CLIID 1 )
```

## CREATE TABLE Command: PRIMARY\_KEY

Creates a PRIMARY KEY INDEX in an NS-DB database.

**Syntax**                    **PRIMARY KEY** *Index\_Name* (*Index\_Column* [ , *Index\_Column*...])  
                              [**TABLESPACE** *TableSpaceName* ] [**DATASPACE** *DataSpaceName*]  
                              [*Index\_Referencial\_Integrity*]

**Notes**

This command is identical to CREATE INDEX...ON ...PRIMARY KEY.

**Example**

```
CREATE I1 ON CLIENT PRIMARY KEY( 1 CLIID 1)
; is equivalent to
PRIMARY KEY I1 ( 1 CLIID 1 )
; Either syntax can be used interchangeably.
```

## CREATE TABLESPACE Command

Creates a TABLESPACE in an NS-DB database.

**Syntax**                    **PRIMARY KEY** *Index\_Name* (*Index\_Column* [ , *Index\_Column*...])  
[**TABLESPACE** *TableSpaceName* ] [**DATASPACE** *DataSpaceName*]  
[*Index\_Referencial\_Integrity*]

### Notes

**1. A TABLESPACE :**

is a physical file structure designed to store data belonging to an NS-DB database. can be real (on disk) or virtual (in memory) (cf. Chapter 2, Optimizing Performance).

the user can freely distribute parts of the database (tables and indexes) over several physical media. TABLESPACES also ensure that the database is portable. is a 3-state object:

- FREE : Not currently storing a table.
- UNCLUSTERED : Currently storing a table.
- CLUSTERED: Currently storing several tables.



NCL generates tables in a CLUSTERED TABLESPACE.

**2. All TABLESPACE definitions for a database are stored in the SYSSPACES table in the database's SYSTEM CATALOG**

**3. The database filename specified in the initialization file is the first TABLESPACE automatically created when the database is initially opened. This TABLESPACE is known as the PRIMARY SPACE. It contains the objects belonging to the database's SYSTEM CATALOG together with all the index instances defined by the user without specifying a TABLESPACE.**

**4. The PRIMARY SPACE is the default storage area.**

**5. A virtual TABLESPACE can be defined to provide a temporary storage area for database items.**

**6. Portability**

The pathname specified in the initialization file when the database is created defines the location of the PRIMARY TABLESPACE, which will contain the database's SYSTEM CATALOG.

For a database to be portable, the PRIMARY TABLESPACE should not contain any data. The database tables should be stored in a separate TABLESPACE so that the PRIMARY TABLESPACE only contains their description.

The location of a TABLESPACE can be defined either statically or dynamically, depending on whether or not you want the database to be portable.

- A static location defines the full pathname of the TABLESPACE (DRIVE:\PATH\FNAME.TYP), which can reduce the portability of the database.
- On the other hand, a dynamic location, specified using predefined macros in the initialization file, improves the portability of the database.

When you move the database to another site, all you need to do is modify the initialization file by changing the pathname specified by the relevant macro.

*Example:*

*Porting a database from C:\MYDIR\DIR1 to D:\YOURDIR\DIR2.*

*If the \$(MYDIR) macro is defined in the NSDB.INI file as follows:*

```
$(MYDIR)='C:\MYDIR\DIR1\';
```

*and a TABLESPACE is installed using this macro:*

```
Create TableSpace space1 '$(MYDIR)FileName.typ'
```

*to port the database to the D:\ drive, the pathname specified by the macro will need to be modified as follows:*

```
$(MYDIR)='D:\YOURDIR\DIR2\';
```

*This modifies the location of the TABLESPACE dynamically (See Chapter 3, Initialization File).*

#### EXAMPLE OF A NON-PORTABLE DATABASE:

```
CREATE TABLESPACE SPACE1 'C:\NSDBR\DB\SPACE1.DAT' DROP
CREATE TABLESPACE SPACE2 'C:\NSDBR\DB\SPACE2.DAT' DROP
```

```
CREATE TABLE CLIENT
(
  CLIID INTEGER NOT NULL,
  INVID INTEGER,
  CLINAME VARCHAR (50)
)
PRIMARY KEY ICLIENT (1 CLIID 1) TABLESPACE SPACE1;
```

```
CREATE INDEX ICLIINV ON CLIENT
(2 INVID ASC 2) TABLESPACE SPACE2
```

#### EXAMPLE OF A PORTABLE DATABASE:

*The following macro must be defined in the [USER] group of the initialization file:*

```
$(MYDIR)='C:\NSDBR\DB';
```

*It will be used as follows when the TABLESPACE is created:*

```
CREATE TABLESPACE SPACE1 '$(MYDIR)SPACE1.DAT' DROP
CREATE TABLESPACE SPACE2 '$(MYDIR)SPACE2.DAT' DROP
```

```
CREATE TABLE CLIENT
(
  CLIID INTEGER NOT NULL,
  INVID INTEGER,
  CLINAME VARCHAR (50)
)
PRIMARY KEY ICLIENT (1 CLIID 1) TABLESPACE SPACE1;
```

```
CREATE INDEX ICLIINV ON CLIENT
(2 INVID ASC 2) TABLESPACE SPACE2
```

*The ALTERCAT.EXE utility (NSDB Version 02) can also be used to port a database from one physical location to another.*  
*The maximum size allowed for a TABLESPACE is 134,215,680 bytes. This physical limit corresponds to the maximum size allowed for an NSDB table in the current version.*  
*The DROP option physically deletes the file if it already existed on the disk when the CREATE TABLESPACE command was called.*

- 7. Sizing :** TABLESPACES are designed to store the rows in the tables defined by the CREATE TABLE command together with their corresponding indexes. In some cases, you may wish to define the precise amount of space actually required for all the data manipulated by a project.

General points:

- The amount of storage allocated for a TABLESPACE is measured in PAGES or BLOCKS.
- Each BLOCK consists of 2048 bytes. The minimum storage allocation unit is 10 BLOCKS, i.e.  $10 \times 2048 = 20480$  bytes.
- The size of the DATA BLOCKS and INDEX BLOCKS can be calculated for each table.
- When NS-DB stores data, it allocates an additional 20480 bytes whenever it requires more disk space. It closes the file and then reopens it to ensure that the clusters are permanently allocated and stable, particularly if a sudden power-failure occurs in the machine.
- Hence, an NSDB database starts with a minimum size of 20480 bytes and will expand in units of 20480 bytes.
- NSDB TABLESPACES do not require any specific reorganization during the database life-cycle. Any pages freed by deletions are retrieved and made available for future insertions. Pages are never returned to the Operating System. As a result, the physical size of the file never decreases.
- Not all of the space in a BLOCK is used to store user data.

- 8. Calculating the useful size of a block:**

For a BLOCK of **2048** bytes, **18** bytes are used up by the system and up to **50%** of the physical area is unoccupied (balanced tree algorithms).

Remainder =  $(2048 - 18) / 2 = 1015$  bytes.

- 9. Calculating the number of rows per block:**

$1015 / ((PKEYSIZE * 2) + ROWSIZE + 8)$

1015 is the useful size of a block, calculated above.

PKEYSIZE is the physical size of the PRIMARY KEY, including CLUSTERING NUMBERS. Each CLUSTERING NUMBER occupies 1 byte. Since the PRIMARY KEY is duplicated, this size is **doubled**.

ROWSIZE stands for the length of a table row.

8 is the number of bytes per row used by the system.

10. Calculating the area occupied by DATA BLOCKS  $\text{ROWCOUNT} / \text{Number of rows per block} * 2048$ .

11. Calculating the area occupied by INDEX BLOCKS  $1015 / (\text{PKEYSIZE} + 8)$   
Where 8 is the number of bytes per row used up by the system.

12. Example:

Suppose we have a table designed to store **20000 rows**, each with a length of **100 bytes**. The primary key is an **INTEGER** column with a length of **4 bytes**.

- The useful size of a block is:  $(2048-18) / 2 = 1015$   
Number of rows per block:  $1015 / ((\text{PKEYSIZE} * 2) + \text{ROWSIZE} + 8)$   
 $\text{PKEYSIZE} = \text{CLUSTERING NUMBER} + \text{INTEGER} + \text{CLUSTERING NUMBER}$   
 $= 1+4+1 = 6$   
 $\text{ROWSIZE} = 100$   
 $1015 / ((6*2) + 100 + 8) = 8 \text{ rows per block.}$
- Size occupied by DATA BLOCKS:  
 $\text{ROWCOUNT} / \text{Number of rows per block} * 2048$ .  
 $\text{ROWCOUNT} = 20000$   
Number of rows per block = 8  
 $(20000 / 8) * 2048 = 5120000 \text{ bytes}$   
5120000 bytes required to store 20000 rows.
- Size occupied by INDEX BLOCKS:  
 $1015 / (\text{PKEYSIZE} + 8)$   
8 is the number of bytes per row used up by the system  
 $1015 / 14 = 72 \text{ keys per index page}$
- Sizing (NSDB V.02):  
In this latest version, the key for an indexed row is no longer duplicated. This increases the size of the area available for user data.  
 $\Rightarrow \text{Number of rows per block: } 1015 / ((\text{PKEYSIZE}) + \text{ROWSIZE} + 8)$

### Example

```
CREATE TABLESPACE SPACE1 'D:\DB\SPACE1.DAT' DROP
CREATE TABLESPACE SPACE1 '$(MACRO) SPACE1.DAT' DROP
CREATE TABLESPACE SPACE1 '\\VIRTUAL\\
```



## DROPDB Command

Drops an NS-DB database.

**Syntax**                **DROPDB** *Database\_Name*

**Notes**

1. All the TABLESPACES defined for the database are deleted together with all the data they contain.
2. The information in the database will be permanently lost.

**Example**

```
DROPDB Mydb
```

## DROP INDEX Command

Drops an index defined earlier using the CREATE INDEX command.

**Syntax**                **DROP INDEX** *Index\_Name*

**Notes**

1. The PRIMARY KEY INDEX for a table can never be deleted.
2. Reminder: a table cannot exist without a PRIMARY KEY INDEX.
3. DROP INDEX does not retrieve the physical disk space occupied by the index.
4. The REORG DATABASE command retrieves the disk space occupied by objects deleted from the SYSTEM CATALOG.

**Example**

```
DROP INDEX Icliname
```

## DROP TABLE Command

Drops a table defined earlier using the CREATE TABLE command.

**Syntax**                **DROP TABLE** *Table\_Name*

**Notes**

1. All the indexes defined for the table will also be dropped from the SYSTEM CATALOG.
2. The DROP TABLE command does not retrieve the physical disk space occupied by the dropped table. The REORG DATABASE command must be used to retrieve the space occupied by any objects deleted from the SYSTEM CATALOG.

**Example**

```
DROP TABLE Table1
```

## REORG DATABASE Command

Reorganizes a database following one or more DROP TABLE or DROP INDEX commands.

**Syntax**                **REORG DATABASE** *Database\_Name*

**Notes**

1. *Database\_Name* stands for the logical database name defined in the NSDB.INI file.
2. This command retrieves the disk space occupied by any indexes or tables deleted by DROP commands.

**Example**

```
REORG DATABASE Mydb
```

## Data Manipulation Language

This part of the manual extends the description of the data access language to cover the functions for manipulating occurrences.

A basic data manipulation language is already defined and used in the NS-DB access library.

This language can be used with an NS-DB database which is accessed concurrently by NS-DBR in the same application.

NS-DBR is an add-on which allows developers to execute new relational operations that are not directly available in the NS-DB interface (joins, projections, set operations).

By virtue of their compatibility with SQL standards, these extensions guarantee portability and provide developers with a familiar interface that is similar to the ones used in other environments (DB2, DBM, etc.).

As a reference, we have used the grammar defined by IBM for the OS/2 relational DATABASE MANAGER. For the operations that have not been defined in this implementation of the language (e.g. OUTER JOINS), we have used the same expressions that are commonly found in comparable DBMSs (ORACLE, INGRES, SYBASE, etc.) and have selected the most effective implementation.

Some extensions are specific to NS-DBR, such as the use of named transactions for multi-database COMMIT and ROLLBACK operations.

The manipulation of VIEW objects is also specific to NS-DBR.



The ability to use more sophisticated functions can potentially reduce compatibility with the reference DBMS (IBM DB2/DBM).

## Features of the DML

### Identifiers

There are several categories of identifiers, which are syntactically recognized as follows:



Syntactic expressions shown in bold will be defined in a future version of the product

- *ColumnsTable* TabName.\*

• <i>ColumnName</i>	Column   TabName.Column
• <i>Columns</i>	ColumnsTable   ColumnName
• <i>Correlation_Name</i>	Sql_Name
• <i>Table_Name</i>	[Database_Name.]Tabname [Correlation_Name]   (FullSelect ) [Correlation_Name] Long_Identifier
• <i>Table_Expression</i>	TableName [ <b>Union Table_Expression</b> ]
• <i>Table_List</i>	Table_Expression [,Table_Expression..., Table_Expression]
• <i>Host_Variable</i>	Identifier
• <i>Sql_Variable</i>	@Identifier
• <i>Data_Value</i>	Constant   Host_Variable   Sql_Variable   NULL
• <i>Function</i>	MIN   MAX   SUM   AVG   COUNT
• <i>Simple_Item</i>	Constant   Column_Name
• <i>Element</i>	[ ( ] Simple_Item [ ) ]
• <i>Simple_Expression</i>	[ +   - ] Item [ - ] [Function] [ ( ] Simple_Item [ ) ]
• <i>OPERATOR</i>	The operators defined by IBM in the SQL grammar are mainly arithmetic: /   +   *   - This list may be extended in the future to include operators that manipulate non-numeric operands.
• <i>Expression</i>	Specifies a value SIMPLE_EXPRESSION [ OPERATOR EXPRESSION ]
• <i>Database_Name</i>	ALPHANUMERIC WITH A LENGTH OF 8 CHARACTERS. This is the logical database name specified in the NSDB.INI file.

## Declaratives

### Purpose and Usage

**Declaratives** are used to define SQL variables or VIEW objects.

The types used to declare these variables are the SQL types currently available in NS-DB:

- CHAR(n)
- VARCHAR(n)
- VARGRAPHIC(n)
- VARGRAPHIC(n)
- SMALLINT
- INTEGER
- SMALLINT
- SYSTIMESTAMP

The standard syntax for a declarative is as follows:

**VIEW** | *Data\_Type* @*Sql\_Name*

*Example* : `VARCHAR @MyVarChar`

### VIEW Declarative

#### ➤ Syntax

**VIEW** @NomVariable

#### ➤ Notes

The **VIEW** keyword is used to define an SQL view object.

Values are assigned using the MOVE command, as for standard types. The only data type that a MOVE command can assign to a VIEW variable is the result of a SELECT:

```
MOVE SELECT * FROM CLIENT TO @MyView
```

PROPERTIES and METHODS are automatically associated with VIEW objects.

---

## View Properties

The properties of a view include all the data associated with the view.

They can be accessed in the standard way, using the following syntax:

```
@ViewName.@PropertyName
```

### ➤ Fixed View Properties

The fixed properties of a view are represented by the following set of variable names, which are available for each user-defined view:

@ViewName.@SQLCOLNO	Defines the number of the current column used by the DESCRIBE method.  NB: the first column is identified by 0. This property is automatically updated whenever the DESCRIBE method is called.
@ViewName.@SQLCOLNAME	Defines the name of the current column used by the DESCRIBE method.
@ViewName.@SQLCOLTYPE	Defines the type of the current column used by the DESCRIBE method.
@ViewName.@SQLCOLLEN	Defines the length of the current column used by the DESCRIBE method.

These properties are updated when the DESCRIBE method is called.

### ➤ Variable View Properties

The variable properties of a view include all the columns defined by the projection operation in the SELECT command assigned to the view by a MOVE statement.



Before executing a MOVE statement, all the variable properties of a view are blank.

## Methods Associated with a View

Methods are a standard set of routines designed to manipulate views.

### ➤ DESCRIBE Method

The DESCRIBE method describes the projection operation carried out by the SELECT statement assigned to a view.



Each call to the DESCRIBE method returns information about the projected data in the following variables:

```
@ViewName.@SQLCOLNAME,  
@ViewName.@SQLCOLTYPE,  
@ViewName.@SQLCOLLEN
```

and increments the variable containing the column number in the projection:

```
@ViewName.@SQLCOLNO
```

If @ViewName.@SQLCOLNO exceeds the number of projected columns (end of SELECT column description), DESCRIBE returns the following error message:

```
DATA BASE RECORD NOT FOUND SQLEX+002
```

#### Example

```
VIEW @View1  
START TRANSACTION Tr1 ON MYDB  
MOVE SELECT * FROM CLIENT  
        WHERE CLINAME IS NOT NULL TO @VIEW1  
@View1.DESCRIBE  
MOVE @View1.@SQLCOLNAME TO :NclHostVar  
MOVE @View1.@SQLCOLTYPE TO :NclHostVar
```



The types coded for these variables are described in Chapter 6, Section 6.4.2.

### ➤ **FETCH Method**

The FETCH method retrieves the next occurrence of the view in the database and returns its information in the view's property variables.

#### Example

```
MOVE SELECT lname, fname FROM CLIENT  
WHERE CLIID IS NOT NULL TO @VIEW1  
@View1.FETCH  
WHILE SQL_ERROR% = 0  
    MOVE @View1.@lname TO :NclHostVar  
    MOVE @View1.@fname TO :NclHostVar  
    @View1.FETCH  
ENDWHILE
```

### ➤ **CLOSE Method**

The CLOSE method closes the view and removes the link between the VIEW variable and the associated SELECT statement.



The methods used for updates (indicated by an asterisk) are not available in the initial version of the product.

These methods will comply with the VIEW UPDATABILITY constraints defined by T CODD and will be capable of updating complex views, such as joins between a number of tables.

---

**➤ DELETEWHRCURRENT (\*) Method**

The DELETEWHRCURRENT method deletes the occurrence currently referenced by the FETCH statement.

**➤ DELETE Method (\*)**

The DELETE method deletes all the occurrences associated with the view.

**➤ UPDATEWHRCURRENT Method (\*)**

The UPDATEWHRCURRENT method updates the occurrence currently referenced by the FETCH command.

**➤ UPDATE Method (\*)**

The UPDATE method updates all the occurrences associated with the view.

**➤ INSERT Method (\*)**

The INSERT method inserts an instance of a view into the database.

*Example:*

```
VIEW @Myview  
VARCHAR(10) @Client_Name  
INTEGER @Client_Number
```

## DML Reference



The commands described below are arranged in alphabetical order.

## AT Command

Specifies the database to be used by the next SQL statement.

**Syntax**                    *AT Database\_Name*  
**SELECT\_CLAUSE | INSERT\_STATEMENT | UPDATE\_STATEMENT**

### Notes

1. If several databases have been opened in a transaction, the first database opened will be the default database used by the interpreter for all SQL statements associated with the transaction.
2. Each SQL statement that does not apply to the first database opened must be preceded by the AT command, which specifies the relevant transaction.
3. If a series of SQL commands do not apply to the default transaction, the USE command should be called to improve the readability of the code.
4. The USE command has the same effect as the AT command.

### Example

```
START TRANSACTION TR1 ON MYDB1, MYDB2
AT MYDB2 SELECT ... FROM ... INTO

: is equivalent to
SELECT * FROM MYDB2.TABLE1

; or, in the case of a join between 2 tables in two different databases within
; the same transaction:
SELECT * FROM MYDB1.TABLE1, MYDB2.TABLE2....
```

**See also**                    START TRANSACTION, USE

---

## COMMIT Command

Physically updates the database with any changes made since the beginning of the transaction or the last COMMIT or ROLLBACK.

### Syntax

#### COMMIT TRANSACTION


*Transaction\_Name* /\* **FORMAT NSDBR** \*/ **COMMIT** [**WORK**]

### Notes

1. To enable this function, the following option must be specified in the NSDB.INI file: JOURNAL=YES.
2. If JOURNAL=NO is specified, transactional journalling will be disabled and changes are written to the disk as and when the update commands (INSERT, DELETE, UPDATE, etc.) are executed. In this case, the COMMIT statement returns a zero error code but has no effect.
3. **Rules:** The COMMIT statement and resource locking

When a statement is used to physically update stored data (CREATE TABLE, CREATE INDEX ...DROP TABLE, DROP INDEX, INSERT, DELETE, UPDATE, etc.), NS-DB locks the physical pages in memory that hold the TABLESPACES containing the modified data.

The physical pages that contain the TABLESPACES remain the exclusive property of the transaction until one of the following events occurs in the transaction:

- COMMIT
- ROLLBACK
- STOP TRANSACTION
- NORMAL TERMINATION of the program
- ABNORMAL TERMINATION of the program.
-  Any another transactions that attempt to update the data in the locked pages will receive the following message: PHYSICAL PAGE IN USE SQLEX-057

The COMMIT [WORK] form of the syntax is the DBM/DB2-compatible version of the COMMIT statement. It commits the default transaction activated by the following NS-SQL command:

- SQL\_OPEN which generates:  

```
START TRANSACTION SYSTRANS ON Database_Name
```

Database\_Name stands for the database name passed as a parameter to the SQL\_OPEN command.
- COMMIT and COMMIT [WORK] are therefore equivalent to:  

```
COMMIT TRANSACTION SYSTRANS
```

**Example**

```
; Start transaction TR1 on three databases  
  
START Transaction TR1 ON DB1,DB2,DB3  
:  
:  
COMMIT Transaction TR1
```

**See also**ROLLBACK, START TRANSACTION, STOP TRANSACTION

---

## DELETE Command

Deletes the table occurrence(s) that satisfy the condition(s) defined in the WHERE clause.

**Syntax**

```
DELETE  
FROMClause  
WHEREClause
```

**Notes**

1. Warning: if no conditions are defined in the WHERE clause, all the rows in the table will be deleted from the database.
2. The syntax for FROMClause and WHEREClause is identical to the syntax used with the SELECT statement. Refer to the SELECT statement for a detailed description of these clauses.
3. Expressing joins in the WHEREClause for the DELETE statement:
  - The fact that a UNIQUE PRIMARY KEY must be defined for each table means that WHEREClauses specifying joins between several tables can be used to delete occurrences without ambiguity.
  - In the initial version of the product, this feature is not fully operational for views. It will be implemented gradually.

**Example**

```
DELETE FROM CLIENT WHERE CLIID IN (100,200,3000,1400)
```

## FETCH Command

Fetches the rows from a prior SELECT statement..

**Syntax**                    **FETCH**  
**INTO** [*host\_variable*] [, *host\_variable*] | [*SQL\_Variable*] [, *SQL\_Variable*]  
[**USING** *Cursor\_name*]

### Notes

1. *Cursor\_name*: specifies the name of a cursor defined earlier using the SQL\_OPENCURSOR% function (see the *NS-SQL* section in the *Libraries* manual).  
If this parameter is omitted, the default cursor allocated by NS-DBR will be used.
2. The cursor will be positioned on the next row and the values retrieved will be assigned to the host variables specified in the INTO clause.

### Example

```
SELECT CLIID, CLINAME FROM CLIENT
FETCH INTO :CLIID, :CLINAME
; Searches the Client table and places the values of the table
; rows into the host variables :CLIID and :CLINAME.
; The current cursor will be used by default.

; This code is equivalent to the previous example.
MOVE SQL_OPENCURSOR% TO My_Cursor
SELECT CLIID, CLINAME FROM CLIENT USING My_Cursor
FETCH INTO :CLIID, :CLINAME
```

**See also**                    **SELECT**

---

## INSERT Command

General command used to insert a row into a table.

**Syntax**

```
INSERT
INTO Table_Name [ ( ColumnName [ ,ColumnName ... ] ) ]
[VALUES([host_variable] [,host_variable] [sql_variable]
[sql_variable])] | SubSelect
```

**Notes**

1. *Column\_name*: this parameter is optional. If it is omitted, all the columns in the record will be inserted.
2. **VALUES**: introduces a list of values to insert.
3. The number of values to be inserted must be equal to the number of columns specified with *ColumnName*.
4. If no columns are specified, the number of host variables must be equal to the total number of columns in the corresponding table.

**Example**

```
INSERT INTO CLIENT ( CLIID , CLINAME )
VALUES ( :CLIID , :CLINAME )
```



## MOVE Command

Transfers data between NCL (:HOST\_VARIABLES) and SQL (@SQL\_VARIABLES) using a syntax similar to the NCL MOVE instruction.

**Syntax**                    **MOVE** *Data\_Value* **TO** *Data\_Value*

**Example**

```
MOVE NULL TO @Cli_Name  
MOVE 1 TO @Cli_ID  
MOVE @SqlVariable TO :HostVariable
```

**See also**                    DECLARATIVES

---

## ROLLBACK Command

Cancels any updates performed since the start of the transaction or since the last COMMIT or ROLLBACK statement.

### Syntax

#### ROLLBACK TRANSACTION


*Transaction\_Name* /\* NSDBR FORMAT / [ROLLBACK [WORK]]

### Notes

1. To enable this function, the following option must be specified in the NSDB.INI file: JOURNAL=YES.
2. If JOURNAL=NO is specified, transactional journalling will be disabled and changes are written to the disk as and when the update commands (INSERT, DELETE, UPDATE, etc.) are executed. In this case, the ROLLBACK statement returns a zero error code but has not effect.
3. **Rules:** ROLLBACK and resource locking

When a statement is used to physically update stored data (CREATE TABLE, CREATE INDEX ...DROP TABLE, DROP INDEX, INSERT, DELETE, UPDATE, etc.), NS-DB locks the physical pages in memory that hold the TABLESPACES containing the modified data.

The physical pages that contain the TABLESPACES remain the exclusive property of the transaction until one of the following events occurs in the transaction:

- COMMIT
- ROLLBACK
- STOP TRANSACTION
- NORMAL TERMINATION of the program
- ABNORMAL TERMINATION of the program.
-  Any another transactions that attempt to update the data in the locked pages will receive the following message:

PHYSICAL PAGE IN USE SQLEX-057

The COMMIT [WORK] form of the syntax is the DBM/DB2-compatible version of the COMMIT statement. It commits the default transaction activated by the following NS-SQL command:

- SQL\_OPEN which generates:  
START TRANSACTION SYSTRANS ON Database\_Name  
Database\_Name stands for the database name passed as a parameter to the SQL\_OPEN command.
- ROLLBACK and ROLLBACK [WORK] are therefore equivalent to:  
ROLLBACK TRANSACTION SYSTRANS

**Example**

```
START Transaction TR1 ON DB1,DB2,DB3
:
:
ROLLBACK Transaction TR1
```

**See also** COMMIT, START TRANSACTION, STOP TRANSACTION

## SELECT Command

### SubSelect, FullSelect

General statement used to select a sub-set of the database. The result of a SELECT must be defined by a relationship.

**Syntax**

```
SELECT_Clause
FROM_Clause
[Into_Clause]
[WHERE_Clause]
[ORDER_BY_CLAUSE][GROUP_BY_CLAUSE [ HAVING_CLAUSE]]
```

### Example

```
SELECT * FROM Staff ;
SELECT Name,Dept,Salary
FROM STAFF,ORG
WHERE Staff.Id = Org.Manager
```

## SELECT Command: FROM\_CLAUSE

Lists the relational tables used by a SELECT statement.

**Syntax**                    **FROM** *Table\_List*

**Notes**

1. NS-DBR allows operations involving tables belonging to separate databases. In this case, a multi-database transaction must be started first of all. Subsequently, to access a table in a given database, the table name in the FROM clause must be prefixed by the database name.
2. If none of the table names in the FROM clause are preceded by a database name, when the SQL statement is compiled, the query will be associated exclusively with the first database specified in the START TRANSACTION statement.

**Example**

```
START TRANSACTION TR1 ON MYDB1 , MYDB2

SELECT * FROM MYDB2.CLIENT A , MYDB2.INVOICE B
WHERE A.CLIID=B.CLIREF

STOP TRANSACTION TR1
```

## SELECT Command: GROUP\_BY\_CLAUSE

Retrieves global information on groups of rows.

**Syntax**                    **GROUP BY** *ColumnName* [**HAVING\_CLAUSE**]

**Notes**

1. Each group of rows shares the same value, which is specified by the GROUP BY clause.
2. The expression typed after the SELECT statement can be:
  - The name of a column that appears in the GROUP BY clause.
  - A function: MIN | MAX | SUM | COUNT | AVG.

**Example**

```
SELECT Dept, MIN (Salary)
FROM Staff
GROUP BY Dept
; This code retrieves the minimum salary for each department
; in the staff table. Salaries will be grouped by department.
```

## SELECT Command: GROUP\_BY\_CLAUSE HAVING\_CLAUSE

Specifies the characteristics of the groups retrieved by the GROUP\_BY clause.

**Syntax**                    **HAVING** *Search\_Condition*

### Notes

1. *Search\_Condition*: specifies a condition that returns TRUE, FALSE or UNKNOWN for a row involved in the following type of relationship:

```
[NOT] [Predicate |...] [ AND | ] [Search_Condition] [OR      ]
```

where:

Predicate = Basic Predicate	
Quantified Predicate	
BETWEEN Predicate	
NULL_Predicate	
LIKE_Predicate	
EXISTS_Predicate	
IN_Predicate	

2. The HAVING clause filters the selected groups by specifying a condition that must be satisfied by each group before it can appear in the result.

### Example

```
SELECT Dept, MIN (Salary)
FROM Staff
GROUP BY Dept
HAVING MIN (Salary) = 20000
; This code retrieves the minimum salary for each department
; in the staff table. Only minimum salaries of $20000 will
; appear in the final result.
```

## SELECT Command: INTO\_CLAUSE

Lists the user-defined variables that will receive the result of a SELECT statement.

**Syntax**            **INTO** [*HostVar*] [, *HostVar*] | [*SQL\_Variable*] [, *SQL\_Variable*]

**Notes**

1. The SELECT...INTO statement triggers a query whose result is placed in the user-defined variables. It is equivalent to a SELECT statement followed by a FETCH statement.
2. A user-defined variable can either be a HOST\_VARIABLE or an SQL\_VARIABLE.
3. A HOST\_VARIABLE begins with a colon (:), followed by an identifier representing a variable defined in the NCL section of the user-defined script.
4. An SQL\_VARIABLE begins with a '@', followed by an identifier representing an SQL variable defined by an SQL declarative. Its type will be one of the following: CHAR, VARCHAR, SMALLINT, INTEGER or FLOAT.
5. The main purpose of defining SQL variables is to enable SQL procedures to be written in a future version of the product.

**Example**

```
VARCHAR(20) @CLINAME  
SELECT CLIID,CLINAME FROM CLIENT INTO :CLIID , @CLINAME
```



## SELECT Command: ORDER\_BY\_CLAUSE

Arranges the rows retrieved by a query in ascending or descending alphabetic order.

**Syntax**                    **ORDER BY** *Simple\_Item* [ASC | DESC],...

**Notes**

1. *Simple\_Item*: specifies the attributes that define the query, which can be:
  - One of the column names specified after SELECT.
  - The sequence number of a column name specified after SELECT.
2. ASC (default) selects an ASCENDING sort order for the result.
3. DESC (default) selects a DESCENDING sort order for the result.
4. The columns specified after ORDER BY must have been specified in the SELECT statement.
5. The ORDER BY clause must appear after all the other clauses. It cannot be used in a sub-query.

**Example**

```
SELECT Dept, MIN(Salary)
FROM Staff
ORDER BY Dept DESC

SELECT Dept, MIN(Salary)
FROM Staff
ORDER BY 1 DESC
; Both queries obtain the same result. They retrieve the
; the department and minimum salary in the staff table
; and arrange the results in descending departmental order.
```

## SELECT Command: SELECT\_CLAUSE

Specifies the structure of the SELECT statement's result.

**Syntax**                 **SELECT [ ALL | DISTINCT ] \* | Columns [...Columns]**

**Notes**

1. The result will be a projection of one or more columns in the tables defined by FROM\_CLAUSE.
2. If the SELECT statement is used without the INTO clause, the command will be prepared but the query will not be executed until the corresponding FETCH command is used.
3. \*: stands for all the columns in the selected tables.
4. Options:
  - If DISTINCT is specified, duplicate rows will be eliminated from the result.
  - If ALL is specified, all the rows in the result will be retrieved.
  - If neither DISTINCT nor ALL are specified, ALL will be used by default.

**See also**                 FROM\_CLAUSE, WHERE\_CLAUSE, INTO\_CLAUSE

## SELECT Command: WHERE Clause

Restricts a query involving one or more tables to a sub-set of rows that satisfy the specified predicates.

### Syntax

**Where\_Clause :** **WHERE** *Search\_Condition*

### Notes

1. *Search\_Condition*: specifies a condition that returns TRUE, FALSE or UNKNOWN for a row in a relationship:

```
[ NOT ] [ Predicate [...] [ AND | ... ] [ Search_Condition ] [ OR ]
```

Where:

Predicate =	Basic Predicate
	Quantified Predicate
	BETWEEN Predicate
	NULL Predicate
	LIKE Predicate
	EXISTS Predicate
	IN Predicate

2. Since Cartesian products are not usually meaningful and often generate a large number of result rows, a WARNING is returned by the optimizer when they are included in a query.

3. Defining a join:

A join is a query whose result is obtained by combining columns from two or more tables. The tables are defined by the FROM clause and the combination of columns is specified by the WHERE clause. A join is only possible if the selected tables share at least one column.

There are two types of joins:

- **Inner joins** are the most common. They retrieve only the combinations of rows from two or more tables whose matching columns contain identical values.
- **Outer joins** retrieve rows based on matching columns from two or more tables but treats the tables asymmetrically.

This asymmetry applies either to the left-hand side or right-hand side of the condition and makes one of the tables the *dominant* table.

All the columns in the *dominant* table that satisfy the search conditions will be retrieved whereas all the columns in the *subservient* table that do not satisfy the search condition will contain NULL values.

Whenever columns from both tables satisfy the search condition, all the data will be displayed, as occurs with a simple inner join.

The operators used are:

- $\langle * \Rightarrow \rangle$  left-hand side is dominant
- $\langle \Rightarrow * \rangle$  right-hand side is dominant.

*Examples*

Suppose we have two tables, *Client* and *Address*, which have the following structure:

CLIENT		ADDRESS	
CliID	INTEGER	AddrID	INTEGER
Lastname	CHAR(15)	StateID	INTEGER
Firstname	CHAR (15)	State	CHAR(20)
Addr1	INTEGER	City	CHAR(15)
Addr2	INTEGER		

and contain the following instances:

CLIENT				
CliID	Lastname	Firstname	Addr1	Addr2
1	N1	Peter	1	
2	N2	Paul	2	4
3	N3	Jean	3	
4	N4	Leo	2	
5	N5	Paul	6	
6	N6	John	7	5

ADDRESS			
AddrID	StateID	State	City
1	05	N.York	NYC
2	13	Calif	LA
3	21	Florida	Miami
4	32	Texas	Dallas
5	19	Washing- ton	Seattle
6	13	Calif	SanFr
7	13	Calif	SanDr

**Inner join**

```
SELECT CliID, Lastname, Firstname, City
FROM Client, Address
WHERE addr1 = AddrID
AND state = "Calif"
```

The result will display all client IDs, last names, first names and cities for the people who live in California:

```
4 N4 Leo LA
5 N5 Paul SanFr
6 N6 John SanDi
```

**Outer Join**

- *Left-hand side is dominant*

```
SELECT CliID, Lastname, Firstname, City
FROM Client, Address
WHERE addr2 *= AddrID
```

Here, Client is the *dominant* table and Address is the *subservient* table. All the clients stored in the Client table are displayed, regardless of whether or not they have a secondary address. For clients that do not have a secondary address, the City field is set to NULL.

```
1 N1 Peter NULL
2 N2 Paul Dallas
3 N3 Jean NULL
4 N4 Leo NULL
5 N5 Paul NULL
6 N6 John Seattle
```

- *Right-hand side is dominant*

```
SELECT CliID, Lastname, Firstname, City
FROM Client, Address
WHERE addr2 =* AddrID
```

Here, Address is the *dominant* table and Client is the *subservient* table. If no secondary address is found for a client, the result displays NULL values together with the City stored in the Address table. For clients that have a secondary address, the ID, last name, first name and town are displayed.

```
0 NULL NULL NYC
0 NULL NULL LA
0 NULL NULL Miami
2 N2 Paul Dallas
6 N6 John Seattle
0 NULL NULL SanFr
0 NULL NULL SanDi
```

**Example**

```
SELECT CLIID FROM CLIENT
WHERE CLINAME LIKE 'M%' OR CLIADDR IS NOT NULL
```

## SELECT Command: WHERE\_CLAUSE BASIC\_PREDICATE

Compares two values.

<b>Syntax</b>	<b>Expression</b>	<b>[ =   ]</b>	<b>[ Expression   ]</b>
---------------	-------------------	----------------	-------------------------

<i>Expression</i>	[ =   ] [ <i>Expression</i>   ]
	[ < >   ] [ <b>Quantified_Predicate</b> ]
	[ <   ]
	[ >   ]
	[ <=   ]
	[ >=   ]
	[ *=   ] (* <i>Jointure externe à droite</i> *)
	[ =* ] (* <i>Right-hand outer join</i> *)

### Example

```
SELECT * FROM STAFF WHERE Dept = 'Sales'
```

## SELECT Command: WHERE\_CLAUSE BETWEEN\_PREDICATE

Compares a value with a set of values: Expression [NOT] BETWEEN Expression AND Expression  
equivalent to : Expression1 >= Expression2 AND Expression1 <= Expression3

**Syntax**                    **BETWEEN\_PREDICATE** : *Expression*  
                              [NOT] **BETWEEN** *Expression* **AND** *Expression*

**Example**

```
SELECT * FROM CLIENT WHERE CLINAME BETWEEN 'M' AND 'N'
```



## SELECT Command: WHERE\_CLAUSE EXIST\_PREDICATE

Tests whether rows exist.

**Syntax**                **EXISTS ( *Fullselect* )**

**Notes**

1. *Fullselect* does not return any values. The result of the predicate can never be UNKNOWN. The result is set to TRUE if the number of rows returned by *Fullselect* is different from 0.
  2. The EXISTS predicate has not yet been implemented in the latest version of NS-DBR.
-

## SELECT Command: WHERE \_CLAUSE IN \_PREDICATE

Compares a value with a set of values.

**Syntax**                      *Expression [NOT] IN (Fullselect) | (Constant [,Constant...]) | Expression*

**Notes**

*In\_predicate*: can almost always be replaced by equivalent expressions:

- Expression IN Expression <=> Expression = Expression
- Expression IN (Fullselect) <=> Expression = ANY(Fullselect)
- Expression IN (Constant1, Constant2,..., ConstantN) is the only form that does not have a simpler equivalent in the language.

**Example**

```
SELECT * FROM CLIENT WHERE CLIID IN (100,200,3000,1400)
SELECT * FROM CLIENT WHERE CLIID IN ( SELECT CLIREF FROM INVOICE WHERE
DESCRIPTION IS NOT NULL )
```

## SELECT Command: WHERE Clause LIKE Predicate

Retrieves character strings that match the format defined by the '\_' and '%' characters in the comparison string.

**Syntax** *Expression [NOT] LIKE String\_Constant*

### Notes

**1. Examples:**

*NAME LIKE '%SMITH%' returns TRUE if SMITH appears anywhere within NAME.*

*STATUS LIKE 'N\_' returns TRUE if STATUS has a length of 2 characters and begins with an N.*

*Wild\_Cards:*

- *%: Any string containing 0 or more characters.*
- *\_: Any single character.*

**2. The LIKE predicate can only be applied to CHAR or VARCHAR columns**

Depending on its structure, the LIKE predicate may or may not result in a full sequential scan of the table.

In the example above (*NAME LIKE '%SMITH%'*), even if NAME has been defined as an index for the table, all the rows will need to be scanned to locate those containing SMITH.

Conversely, in the case of *NAME LIKE 'SMITH%'*, which means 'all names beginning with SMITH', if an index is defined on NAME, the optimizer will restrict the search to occurrences of the NAME index that begin with 'SMITH'.

If a LIKE predicate will definitely result in a sequential scan, the following warning is generated by the optimizer when the query is compiled:

```
LIKE STRUCTURE IMPLIES SEQUENTIAL TABLE SCAN SQLEX+014
```

### Example

```
SELECT * FROM CLIENT WHERE CLNAME NOT LIKE 'D%'
```

## SELECT Command: WHERE\_CLAUSE NULL\_PREDICATE

Tests whether an expression is NULL or NOT NULL.

**Syntax**                      *Expression* **IS** [NOT] NULL

**Example**

```
SELECT * FROM CLIENT WHERE CLINAME IS NOT NULL
```

## SELECT Command: WHERE\_CLAUSE QUANTIFIED\_PREDICATE

Compares a value with a set of values.

**Syntax**            [ [ **SOME** ] ( *Fullselect* ) ] [ **ANY** ] [ **ALL** ]

**Notes**

1. If you specify:
  - **SOME** or **ANY**, the result will be TRUE if the specified condition is satisfied or if at least one value is returned by *Fullselect*.
  - **ALL**, the result of the predicate will be TRUE if *Fullselect* does not return any values or if the result of the condition is TRUE for each value returned by *Fullselect*.
2. Quantified predicates have not yet been implemented in the latest version of NS-DBR.

## START TRANSACTION Command

Starts an SQL TRANSACTION on one or more logical databases defined in the file identified by the NS-DB environment parameter.

**Syntax**                    **START TRANSACTION** *Transaction\_Name*  
                              **ON** *Database\_Name* [ , *Database\_Name...*]

**Notes**

1. An NS-DBR transaction is a temporary unit of work on one or more local or remote NS-DB databases.
2. Each transaction started has a name (TRANSACTION\_NAME) which is used by the STOP TRANSACTION, COMMIT and ROLLBACK statements to identify it.
3. The same program can start several transactions on a database. The updates performed by each transaction are locked until they are confirmed (COMMIT) or canceled (ROLLBACK) by the transaction. Any locks applied by the update statements are freed by these two commands.
4. If transactions are embedded, cursors can be used to specify which transaction each command applies to (Cf. *NSSQL Library* in the *Libraries* manual.). The cursor opened before the transaction was started is associated with this transaction. Any subsequent commands must specify the cursor used in order to define the transaction. This method could cause problems due to the way concurrent transactions are managed during update operations (Cf. COMMIT and ROLLBACK).
5. An SQL transaction differs from a DB\_STARTRN or a DB\_OPEN since it can perform and synchronize updates on several local or remote logical databases.

**Example**

```
START TRANSACTION Trans1 ON MYDB , MYDB2 , MYDB3
START TRANSACTION Trans2 ON MYDB4
:
:
COMMIT Trans1
ROLLBACK Trans2
```

**See also**                    COMMIT, STOP TRANSACTION, ROLLBACK

## STOP TRANSACTION Command

Stops a transaction started earlier by the START TRANSACTION command.

**Syntax**                    **STOP TRANSACTION** *Transaction\_Name*

**Notes**

1. The STOP TRANSACTION statement generates an implicit COMMIT for the transaction before freeing the resources.
2. The NCL SQL\_STOP instruction automatically stops all active transactions using the STOP TRANSACTION command.

**Example**

```
START TRANSACTION Trans1 ON MYDB , MYDB2 , MYDB3
START TRANSACTION Trans2 ON MYDB4
:
:
STOP TRANSACTION Trans1
STOP TRANSACTION Trans2
```

**See also**                    COMMIT, START TRANSACTION, ROLLBACK

---

## UPDATE Command

The UPDATE command updates one or more columns in a set of table rows selected by the WHERE clause.

**Syntax**                    **UPDATE** *Table\_List* **SET** *ColumnName= Data\_Value*  
                              [...*ColumnName=Data\_Value*] |  
                              [*ColumnName=ColumnName Operateur DataValue*] **WHERE\_Clause**

**Notes**

If no conditions are defined in the WHERE clause, all the rows in the table will be updated.

**Example**

```
UPDATE CLIENT SET ADDRESS=NULL WHERE CLINAME BETWEEN 'N' AND 'M'
```



## USE Command

Specifies the database to be used by the next SQL statement.

**Syntax**                *USE Database\_Name*

**Notes**

1. If a transaction is started on several databases, the default database used by the interpreter will be the first database opened.
2. A series of SQL statements that do not apply to the first database opened must be preceded by the USE command, which specifies the actual database used.
3. To return to the default database, the USE command must be called again.
4. The USE command has the same effect as the AT command.

**Example**

```
START TRANSACTION T1 ON MYDB1, MYDB2
; Mydb1 is the default database
USE MYDB2
SELECT .... FROM ... INTO.....
...
INSERT....
UPDATE....
USE MYDB1
```

**See also**                AT, START TRANSACTION

---



## Chapter 5

# A Simple Application



### ***This chapter explains***

- Updating the NSDB.INI Initialization File.
- Describing the Data.
- Manipulating the Data.
- Arranging the Results.
- Modifying the Database Structure.

## Contents

Introduction.....	5-3
Features of the Data Description Language	5-3
Updating the NSDB.INI Initialization File .....	5-4
Describing the Data .....	5-5
Creating a Database	5-5
Loading the Library	5-5
Connecting to the Database	5-6
Creating a Table and an Index	5-6
➤ Full example	
Manipulating the Data.....	5-9
Inserting a Row into a Table	5-9
➤ Full example	
Searching a Table	5-10
Projecting All Columns	5-10
➤ Example	
DISTINCT Clause and Functions	5-10
➤ Example	
Retrieving Data in Host Variables	5-11
➤ Example	
Scanning Table Rows	5-11
➤ Example	
Updating a Table Row	5-12
Joins	5-12
➤ Example	
Arranging the Results .....	5-14
ORDER BY...	5-14
Modifying the Database Structure .....	5-15
Adding a Column	5-15
Adding and Building an Index	5-15

# Introduction

## Features of the Data Description Language

This chapter provides a quick description of various features that will help you get started with NS-DBR with the aid of an NCL example.

You will develop an application designed to record bugs and details of their corrections in a database.

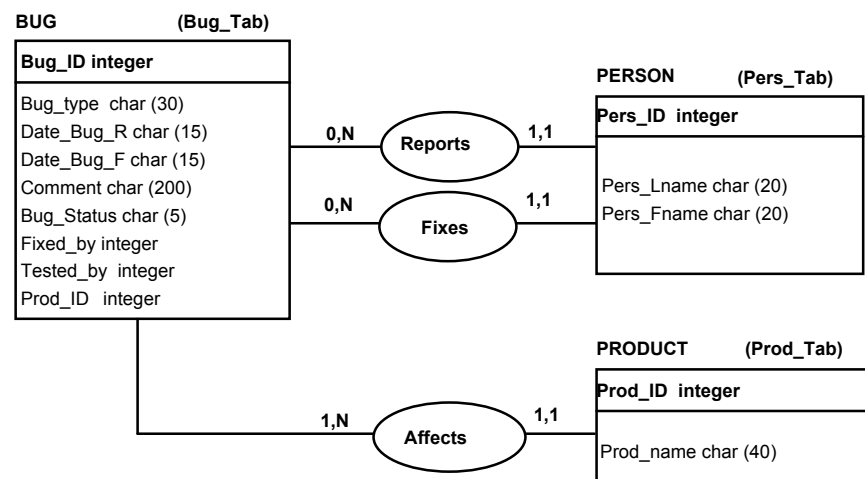
Each bug record will include the following details:

- Input date
- Correction date
- Associated comment
- Names of the people who reported and corrected the bug.

The database used by the application contains three tables which store details about each:

- Person
- Bug
- Product.

This database is illustrated by the following conceptual diagram:



## Updating the NSDB.INI Initialization File

The basic syntax for the NSDB.INI file is as follows:

```
[SYSTEM]
dbdir='C:\NSDBR\DB';           /* pathname of the NSDB working
    directory */
sysId='NAME';
journal=YES;

[USER]
DBTEST='C:\NSDBR\DB\DBTEST';   /*Bug manager database*/
SQLWKDB='C:\NSDBR\DB\TEMP',casesens;/*Temp NSDBR db*/

[END]
END_INIT;
```



SQLWKDB is a temporary database used to execute relational operations such as DISTINCT, ORDER BY, GROUP BY, and so on.

You must add it to the initialization file, otherwise the message -93 (INIT SQL WORK DB ERROR) will be displayed when NS-DBR initializes.

# Describing the Data

## Creating a Database

Creating a database involves:

- Using the data description language.
- Loading the library.
- Creating the actual database and its components (tables and indexes).

You can code these operations in the script for the INIT event associated with the application's main window.

## Loading the Library



The NSnnSQL libraries are similar to the NS-SQL interface libraries supplied by Nat System for Oracle, DBM and other DBMSs.

nn stands for the version number of the interface that you have installed.

Refer to the Nat System documentation for a detailed description of the functions in this library.

Load the NSnnSQL library by using the SQL\_INIT command as follows:

```
SQL_INIT "NS01SQL"
```

or

```
SQL_INIT "NS02SQL" ; NSDK Version 1.01 or higher
```

Error handling routines can be used to check that the load was successful. For example, you can test for errors using the following code:

```
IF SQL_ERROR% <> 0
  MESSAGE "SQL_INIT \ error",sql_ERRMSG$(SQL_ERROR%)
ELSEIF SQL_ERROR% = 0
  MESSAGE "", "SQL_INIT OK"
ENDIF
```

As a precaution, we recommend that you use a full error handling procedure after each SQL statement.


```
IF SQL_ERROR% < 0
  MESSAGE "Error:", SQL_ERRMSG$(SQL_ERROR%)
ELSEIF SQL_ERROR% > 0
  MESSAGE "Warning:", SQL_ERRMSG$(SQL_ERROR%)
ELSEIF SQL_ERROR% = 0
  MESSAGE "", "SQL_EXEC... statement OK"
ENDIF
```

## Connecting to the Database

The logical database name has already been specified in the initialization file:

```
DBTEST='C:\NSDBR\DBTEST' ;
```

The following two commands can be used to connect to a database :

- `SQL_OPEN "DBTEST","password"`  
Defines a transaction for DBM, SYBASE and ORACLE.  
 "password" is not used by NS-DB.
- `SQL_EXEC START TRANSACTION T1 ON DBTEST`  
Creates the database and starts a transaction on the DBTEST database specified in the initialization file identified by the SET NS-DB environment parameter.



If you try to connect to a database that does not exist, NS-DBR physically creates this database.

## Creating a Table and an Index

The CREATE TABLE command creates a table in the database.

The parser requires the CREATE TABLE command to define a PRIMARY KEY when the table is created.

You can create the PRIMARY KEY, immediately after creating the table, in two ways:

- ♦ By specifying PRIMARY KEY...

The bug table, BUG\_TAB, will be created as follows:

```
SQL EXEC CREATE TABLE BUG TAB \
(
  BUG_ID integer NOT NULL ,      \
  BUG_TYPE char (30) ,          \
  DATE_BUG_R char (15) ,        \
  DATE_BUG_F char (15) ,        \
  COMMENT char (200),           \
  BUG_STATUS char (5),           \
  FIXED BY integer,              \
  TESTED BY integer              \
) \
PRIMARY KEY IBUG (1 BUG_ID 1)
```

PRIMARY KEY is the bug ID.

The digits 1 ..... 1 are the CLUSTERING NUMBERS used to speed up data access (cf. *Clustering an Index*).

- ♦ Using the CREATE INDEX command



```
SQL EXEC CREATE INDEX IBUG \
      ON BUG_TAB (1 BUG_ID 1)
```

After being created, the tables will be physically built by the following command:

```
SQL_EXEC COMMIT TRANSACTION T1
```

### ➤ Full example

```
; Load the DLL

SQL_INIT "NS01SQL"

; Error handling used to check whether the
; library has been loaded

IF SQL_ERROR% <> 0
    MESSAGE "INIT error:", SQL_ERRMSG$(SQL_ERROR%)
ENDIF
IF SQL_ERROR% = 0
    MESSAGE "", "INIT OK"
ENDIF

; Create the database specified in the
; initialization file (ensuring
; compatibility with DBM, SYBASE and ORACLE).

SQL_OPEN "DBTEST", ""

SQL_EXEC START TRANSACTION T1 ON DBTEST

; Create the bug table

SQL EXEC CREATE TABLE BUG_TAB \
( \
    BUG_ID integer NOT NULL, \
    BUG_TYPE char (30), \
    DATE_BUG_R char (15), \
    DATE_BUG_F char (15), \
    COMMENT char (200), \
    BUG_STATUS char (5), \
    FIXED_BY integer, \
    TESTED_BY integer \
) \
PRIMARY KEY IBUG (1 BUG_ID 1)

; Or, as an alternative to "PRIMARY KEY...":
; SQL_EXEC CREATE INDEX IBUG ON BUG_TAB (1 BUG_ID 1)

; Error handling used to check whether the
; BUG_TAB table has been created

IF SQL_ERROR% <> 0
    MESSAGE "Error: ", SQL_ERRMSG$(SQL_ERROR%)
ENDIF
IF SQL_ERROR% = 0
    MESSAGE "Create Table:", "OK"
ENDIF

; Create the person table, PERS_TAB

SQL_EXEC CREATE TABLE PERS_TAB \
( \
    PERS_ID integer NOT NULL, \
    PERS_LNAME char (20), \
```

## 5-8      *A Simple Application*

---

```
        PERS FNAME char (20)  \
    )      \
PRIMARY KEY IPERS (1 PERS_ID 1)

; Error handling used to check whether the
; PERS_TAB table has been created

IF SQL_ERROR% <> 0
    MESSAGE "Error: ",SQL_ERRMSG$(SQL_ERROR%)
ENDIF
IF SQL_ERROR% = 0
    MESSAGE "Create Table:", "OK"
ENDIF

; Update the database
SQL_EXEC COMMIT
```

## Manipulating the Data

### Inserting a Row into a Table

The INSERT INTO command inserts a row into a table.

Suppose we have a DIALOG window containing Entry fields and Combo boxes.

First of all, we will place the value of each control in an NCL global variable.

```
MOVE ef_id TO pers_id%  
MOVE ef_lname TO pers$  
MOVE ef_fname TO pers_fn$
```

The command used to insert these values into the table uses the global variables, known as "host variables". They are used as follows:

```
SQL EXEC INSERT INTO PERS TAB \  
    (pers_id,pers_lname,pers_fname) \  
    VALUES (:pers_id%, :pers$, :pers_fn$)
```

If all the columns are specified, this command can also be coded as follows:

```
SQL EXEC INSERT INTO PERS TAB \  
    VALUES (:pers_id%, :pers$, :pers_fn$)
```

#### ➤ Full example

```
; Move the values of the Entry fields into  
; variables  
IF ef_lname <> "" AND ef_fname <> ""  
    MOVE ef_lname TO pers$  
    MOVE ef_fname TO pers_fn$  
ELSE  
    MOVE 1 TO pers_ID%  
ENDIF  
  
; Insert the ID, last name and first name  
; into the person table:  
SQL EXEC INSERT INTO Pers_Tab  
    (pers ID, pers lname, pers fname) \  
    VALUES (:pers ID,:pers$, :pers fn$)  
  
; Error handling used to check whether the  
; insertion was successful  
IF SQL_ERROR% <> 0  
    MESSAGE "Insert error:", SQL_ERRMSG$(SQL_ERROR%)  
ENDIF  
IF SQL_ERROR% = 0  
    MESSAGE "", "Insert OK"  
ENDIF  
  
; Commit the updates  
SQL EXEC COMMIT
```

## Searching a Table

Searching for items in a table involves:

- Using the Data Manipulation Language.
- Using any combination of the SELECT clause and restrictions defined by the WHERE clause. The SELECT clause positions a cursor on the first row in the series of rows that satisfy the search argument specified by WHERE.

## Projecting All Columns

The SELECT \* clause produces a result containing all the columns in the table.

The WHERE condition restricts the query.

### ➤ **Example**

```
; Retrieve all the rows in PERS_TAB (ID,  
; last name and first name) whose ID field  
; contains 1.  
  
SQL EXEC SELECT * FROM Pers_Tab WHERE \ pers ID = 1  
  
; Retrieve all the rows in PERS_TAB  
; whose last name field contains "Doe".  
  
SQL_EXEC SELECT * FROM Pers_Tab \  
WHERE pers lname = "Doe"  
  
; Retrieve all the rows in PERS_TAB  
; whose last name field matches the value  
; in the NCL global variable :pers$  
  
SQL_EXEC SELECT * FROM Pers_Tab \  
WHERE pers_lname = :pers$
```

## DISTINCT Clause and Functions

The DISTINCT, MAX, MIN and COUNT predicates are used to define searches whose results are subject to conditions.

### ➤ **Example**

```
; Retrieve the names of all the people in the  
; table but eliminate duplicates  
  
SQL EXEC SELECT DISTINCT pers lname \  
FROM Pers_Tab  
  
; Retrieve the highest ID number in the  
; PERS_ID column  
  
SQL_EXEC SELECT MAX (pers_ID) FROM Pers_Tab
```

```

; Retrieve the lowest ID number in the
; PERS_ID column
SQL_EXEC SELECT MIN (pers_ID) FROM Pers_Tab

; Retrieve the number of rows whose bug fix
; date (date bug f) is not blank

SQL_EXEC SELECT COUNT (date bug f) FROM \ BUG TAB \
      INTO :C% \
      WHERE date_bug_f <>" "

```

## Retrieving Data in Host Variables

The .....INTO..... clause is used to place the retrieved data items in host variables.

This clause is used with the SELECT and FETCH clauses.

### ➤ Example

```

; Use pers_ID to retrieve the highest
; personal ID found by SELECT

SQL_EXEC SELECT MAX (pers_ID) \
      FROM Pers_Tab \
      INTO :pers_ID%

; Retrieve the first row whose ID number
; is greater than 0 and move the values
; in each column into the corresponding
; host variables :pers_ID%, :pers$ &
; :pers_fn$

SQL_EXEC SELECT * \
      FROM Pers_Tab \
      INTO (:pers_ID%, :pers$, :pers_fn%) \
      WHERE pers_ID > 0

```

## Scanning Table Rows

The FETCH command scans the rows retrieved by a SELECT.

Suppose we want to retrieve the last name and first name of all the people in the Person table and insert them into a Combo box.

### ➤ Example

```

; Position the cursor in the first row in the table and use the host
; variables pers$ and pers fn$ to retrieve the items found

SQL_EXEC SELECT pers_lname, pers_fname \
      FROM Pers_Tab \
      INTO :pers$, :pers_fn$

; Loop while no errors or warnings are
; returned by the FETCH command

```

```
WHILE SQL_ERROR% = 0

    ; Insert data into the Combo box
    INSERT AT END pers$&&pers_fn$ TO Combo_box

    ; Read next row and retrieve its data
    SQL_EXEC FETCH INTO :pers$, :pers_fn$

ENDWHILE
```



A search loop is terminated normally by the error code which indicates that no (more) columns match the specified search criteria:

```
+100 : NO ROW FOUND OR RESULT OF QUERY IS EMPTY TABLE
```

## Updating a Table Row

A row is updated by the UPDATE ... SET ... command.

The WHERE clause used with the UPDATE command defines the set of occurrences to update.

```
IF bug_status$ <> "OK"

SQL EXEC UPDATE BUG TAB          \
    SET date_bug_f = :date_f$,    \
    bug_status = :bug_status$,    \
    fixed by = :fixed by%,        \
    comment = :com$              \
    WHERE bug_ID = :bug_ID%
```

This command updates:

- The row that contains the relevant bug.
- The bug fix date.
- The bug status.
- The ID of the person who fixed the bug.
- The comment about the bug fix.

## Joins

Joins are used to combine information from different tables. They are based on a WHERE clause that compares columns that are common to two tables.

For our example, a join can be used to search the PERS\_TAB table and retrieve the name of the tester who reported a bug to a developer.

The tested\_by column in the BUG\_TAB table retrieves the ID of the person stored in the PERS\_TAB table.

```
SQL EXEC SELECT pers ID \
          FROM Pers_Tab INTO :pers_ID% \
          WHERE pers_lname = :pers$

MOVE pers_ID% TO tested_by%

SQL EXEC INSERT INTO BUG_TAB bug ID,bug type,tested by) \
          VALUES ( :bug_ID%, :bug_type$, :tested_by%)
```

This code retrieves:

- The bug ID.
- The bug type.
- The tester ID which, when joined with the person's ID number (PERS\_ID) in the PERS\_TAB table will retrieve the first name and last name.

### ➤ Example

```
; Move the value of the Entry field ef_id
; into the global variable bug_ID%

MOVE ef_id TO bug_ID%

; Retrieve the bug ID, the person who tested
; it, the person's ID, last name and first
; name

SQL_EXEC SELECT bug_ID, tested_by, pers_ID, \
          pers_lname, pers_fname \
          FROM BUG_TAB A, Pers_Tab B \

; Place the retrieved values in the host
; variables :bug_ID%, :tested_by%, :pers_ID%,
; :pers$, and :pers_fn$

          INTO :bug_ID%, :tested_by%, \
          pers_ID%, :pers$, :pers_fn$ \

; Search for the bug ID which must be equal
; to the value entered in the Entry field
; EF ID (the tester's ID must be equal to the
; person in "Pers ID")

          WHERE A.Bug_ID=:bug_ID% \
          AND B.Pers_ID=A.Tested_by ; \

; Display the first and last name of the
; person who reported the bug ":bug_ID%":

MESSAGE "Tester's first and last name:" ,fname$&&lname$
```



A join between two unindexed columns slows down response time considerably. As a result, a warning (code +15) is displayed when the Select statement is compiled (i.e. before it is executed).

We will define an index for the PROD\_ID column in the BUG\_TAB table later on.

## Arranging the Results

The GROUP BY...HAVING and ORDER BY clauses arrange the results in the specified sort order.

Our example is not suitable for the GROUP BY clause, so we will only use the ORDER BY clause here.

### ORDER BY...

The ORDER BY clause defines how the result will be sorted.

The result, containing the bug ID, bug type, bug fix date and related product, will be sorted by the bug fix date. This is coded as follows:

```
SQL_EXEC SELECT      bug_ID,bug_type,date_bug_f, \
                    PROD_TAB.prod_id,PROD_TAB.prod_name \
FROM                BUG_TAB A, PROD_TAB B \
INTO                :bug_ID%,:type$, :date_f$, :prod$, :prod_id% \
WHERE               date_bug_f <> " " \
AND                 A.prod_id = B.prod_id \
ORDER BY            date_bug_f
```



The *Prod\_ID* column in the BUG\_TAB table was added after creating the table with the ALTER TABLE command. The index was added to the table by executing the CREATE INDEX command. These operations are described in the next section.



## Modifying the Database Structure

You can modify the databases created to suit your application's requirements by:

- Adding a column to a table.
- Creating an index for an existing table.
- And so on.

### Adding a Column

A column is added to an existing table by calling the ALTER TABLE command.

The following instruction will add the PROD\_ID column to the Bug table (BUG\_TAB):

```
SQL_EXEC ALTER TABLE BUG_TAB ADD (prod_id integer)
```

### Adding and Building an Index

An index is created for an existing table by executing the CREATE INDEX command followed by the BUILD INDEX command, which physically builds the index for the current occurrences in the table.

Our example uses the *Prod\_ID* column (added to the bug table) whenever we want to retrieve the name of the product (in the PROD\_TAB table) that is affected by a bug (in the BUG\_TAB table).

The product name is obtained joining the product IDs in the two tables: BUG\_TAB and PROD\_TAB.

The product number column in the product table (PROD\_TAB) is indexed whereas the one in the bug table (BUG\_TAB) is not.

Since a join between two unindexed columns slows down response time considerably, we need to index the product ID column in the bug table (BUG\_TAB) by coding the following:

```
SQL_EXEC CREATE INDEX IBUGP ON BUG_TAB (6 prod_id 6)
```

```
SQL_EXEC BUILD INDEX IBUGP
```

---



## Chapter 6

# SYSTEM CATALOG



### ***This chapter explains***

- Objects in the Catalog.
- Describing Objects in Another Language.
- Details on the Contents of the Catalog.

## Contents

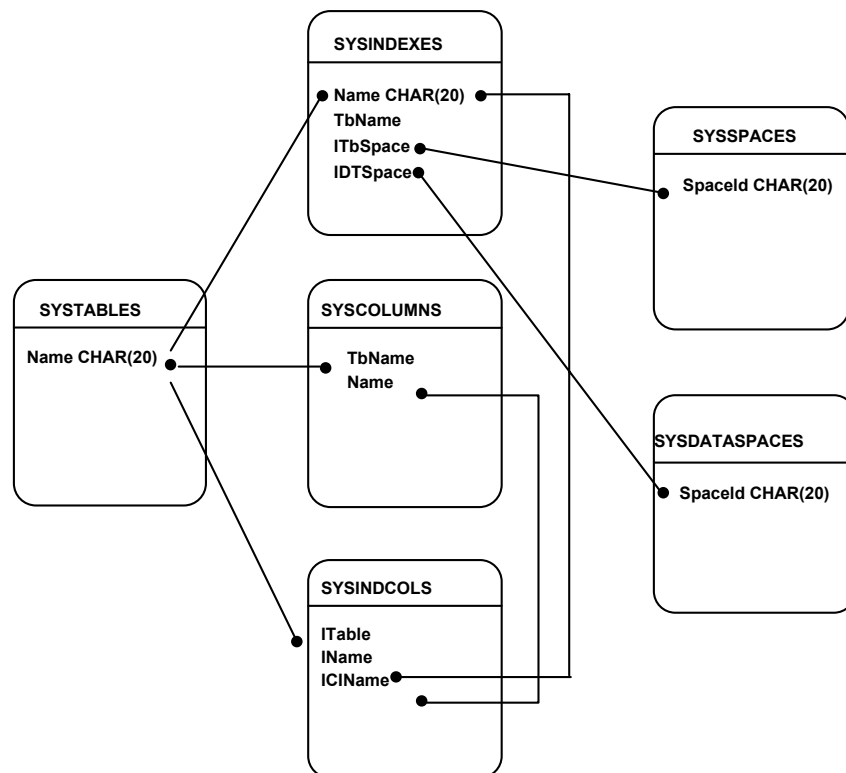
Introduction.....	6-3
Objects in the Catalog .....	6-5
Describing Objects in Another Language .....	6-6
Example      6-6	
Details on the Contents of the Catalog .....	6-8
SYSTABLES Structure    6-8	
SYSCOLUMNS Structure      6-8	
SYSINDCOLS Structure 6-9	
SYSINDEXES Structure 6-10	
SYSSPACES Structure    6-11	
SYSDATASPACES Structure    6-11	

## Introduction

All objects manipulated by the DBMS are stored in tables, in the same way as the objects manipulated by the user.

- These tables use reserved names which begin with the prefix 'SYS'.
- From now on, we will refer to this set of tables as the SYSTEM CATALOG.

Which objects are stored in the SYSTEM CATALOG?



The catalog contains tables made up of columns.

Each table is stored in a physical space known as the B\_TREE (all the tables in a given database can be located in the same physical area).

It is possible to calculate the physical area occupied by each table. These calculations are detailed in Chapter 4 under the syntax description for CREATE TABLESPACE.

A PRIMARY KEY is defined for each table. It allows an object to be identified uniquely.

Each B\_TREE space used to store data physically is known as a TABLESPACE

Each database is essentially defined by its catalog, which describes the structure of all the objects manipulated by the user together with all those manipulated by the system.

The SYSTEM CATALOG is:

- Generated when the Data Description Language (DDL) commands are compiled.
- Generated when commands such as CREATE TABLE and CREATE INDEX are executed.
- Stored in a resulting TABLESPACE, known as the PRIMARY TABLESPACE.

The PRIMARY TABLESPACE is built automatically when the database is created. It uses the filename defined in the initialization file for the created database.



The PRIMARY TABLESPACE for a database contains, at least, the database catalog and, at most, all the instances of all the other objects.

The CREATE TABLESPACE command allows the user to create additional physical storage areas.

When a table or index is created, the user can specify the TABLESPACE used to store the object. If this information is not specified, the object will be stored in the PRIMARY TABLESPACE.

## Objects in the Catalog

The SYSTEM CATALOG contains a basic set of reference tables known as SYSTEM TABLES. Their purpose is to store information about the USER TABLES, which the user will define in the database by executing commands in the Data Description Language (DDL).

The SYSTEM TABLES are automatically built when the database is initially created.

The SYSTEM CATALOGS used by relational database managers are all based on similar data models. We have tried to comply with the same object naming conventions as DB2 and DBM.

The following is a list of the NS-DB system tables:

- **SYSTABLES**  
Contains a list of tables defined in the database.
  - **SYSSPACES**  
Lists the TABLESPACES defined in the database.
  - **SYSDATASPACES (NSDB V.02)**  
Lists the DATASPACES defined in the database.
  - **SYSINDEXES**  
Lists the indexes defined in the database.
  - **SYSCOLUMNS**  
Lists the table columns defined in the database.
  - **SYSINDCOLS**  
Lists the index columns in the database.
-

## Describing Objects in Another Language

The catalog description can be easily generated in another language using the DCLGEN utility (described in Chapter 9).

### Example

```
DCLGEN MYDB NCL ALL INCLUDE.NCL
```

This command generates an NCL description of all the table structures in the INCLUDE.NCL source file.

If the database is empty (i.e. does not contain any user-defined tables), the generated INCLUDE.NCL file will contain the following:

```
; /* */
/* TABLE SYSCOLUMNS CATALOG STRUCTURE */
SEGMENT SYSCOLUMNS
  CSTRING  TBNAME(19);
  CSTRING  NAME(19);
  INT      COLNO(2);
  INT      COLTYPE(2);
  INT      LOG_POS(2);
  INT      LENGTH(2);
  CSTRING  NULLS(1);
ENDSEGMENT
; /* */
; /* TABLE SYSINDCOLS CATALOG STRUCTURE */
SEGMENT SYSINDCOLS
  CSTRING  INAME(19);
  CSTRING  ICLNAME(19);
  CSTRING  ITABLE(19);
  INT      ICOLNO(2);
  INT      ICLUTOP(2);
  INT      ICLUBOT(2);
  INT      IPKEY(2);
  CSTRING  DESCENDING(1);
ENDSEGMENT
; /* */
; /* TABLE SYSINDEXES CATALOG STRUCTURE */
SEGMENT SYSINDEXES
  CSTRING  NAME(19);
  INT      IID(2);
  CSTRING  ITBSPACE(19);
  INT      ITYPE(2);
  INT      KEY_END(2);
  INT      NB_COLS(2);
  INT      LRECL(2);
  INT      KEYLEN(2);
  CSTRING  SYSCATOBJ(1);
  CSTRING  TBNAME(19);
  INT      FKEY_IND_I(1); /* Null Value Indic */
  CSTRING  FKEY_IND(19);
  CSTRING  IDTSPACE (19); /* NSDB V.02)*/
```



```
ENDSEGMENT
; /* */
; /* TABLE SYSSPACES CATALOG STRUCTURE */
SEGMENT SYSSPACES
    CSTRING    SPACEID(19);
    CSTRING    PHYS_NAME(60);
    INT        TABLEUSAGE(2);
ENDSEGMENT
; /* */
; /* TABLE SYSTABLES CATALOG STRUCTURE */
SEGMENT SYSTABLES
    CSTRING    NAME(19);
    INT        LGSIZE(2);
    CSTRING    SYSCATOBJ(1);
ENDSEGMENT
; /* */
; /* TABLE SYSDATASPACES CATALOG STRUCTURE NSDB V.02) */
SEGMENT SYSDATASPACES
    CSTRING    SPACEID(19);
    CSTRING    PHYS_NAME(60);
    INT        TABLEUSAGE(2);
ENDSEGMENT
```

## Details on the Contents of the Catalog

This section provides a detailed description of the contents of each column in the SYSTEM CATALOG tables.

### SYSTABLES Structure

Table name:

```
NAME CHAR(18) NOT NULL
```

Maximum physical size of a table row in bytes:

```
LGSIZE SMALLINT NOT NULL
```

Table type ('Y': SYSTEM\_CATALOG\_TABLE; 'N': Table defined by the user with the CREATE TABLE command):

```
SYSCATOBJ CHAR(1) NOT NULL
```

### SYSCOLUMNS Structure

Name of the table that contains the column:

```
TBNAME CHAR(18) NOT NULL
```

Column name:

```
NAME CHAR(18) NOT NULL
```

Column's sequence number in the table:

```
COLNO SMALLINT NOT NULL
```

Column data type:

```
COLTYPE SMALLINT NOT NULL
  0 : CHAR
  1 : VARCHAR
  2 : SMALLINT
  3 : INTEGER
  4 : FLOAT
  5 : VARGRAPHIC
  6 : GRAPHIC
  7 : DECIMAL
  8 : SYSTIMESTAMP
```

Position of the data in a table row. The reference position is 0:

```
LOG_POS SMALLINT NOT NULL
```

Physical data length in bytes (including the NULL byte if NULLS='Y'):

```
LENGTH SMALLINT NOT NULL
```

Values accepted by a column (Y: the column accepts NULL values; N: the column does not accept NULL values):

```
NULLS CHAR(1) NOT NULL
```

## SYSINDCOLS Structure

Index name:

```
INAME CHAR(18) NOT NULL
```

Index column name:

```
ICLNAME CHAR(18) NOT NULL
```

Name of the table that contains this column:

```
ITABLE CHAR(18) NOT NULL
```

Sequence number of the column in the index:

```
ICOLNO SMALLINT NOT NULL
```

CLUSTERING NUMBER (ranging from 1 to 255) physically associated with the start of the column:

```
ICLUTOP SMALLINT NOT NULL
```

CLUSTERING NUMBER physically associated with the end of the column:

```
ICLUBOT SMALLINT NOT NULL
```

Index type:

```
IPKEY SMALLINT NOT NULL
0: PATH_INDEX
1: PKEY_INDEX
2: FKEY_INDEX
```

Descending or ascending sort order for the column index: ('Y': DESCENDING; 'N': ASCENDING):

```
DESCENDING CHAR(1) NOT NULL
```

---

## SYSINDEXES Structure

Index name:

```
NAME CHAR(18) NOT NULL
```

Internal index ID number:

```
IID SMALLINT NOT NULL
```

Name of the tablespace in the SYSSPACES table that stores the index:

```
ITBSpace CHAR(18) NOT NULL
```

Index type:

```
ITYPE SMALLINT NOT NULL
```

Physical position of the end of the key in the index:

```
0: PATH_INDEX  
1: PKEY_INDEX  
2: FKEY_INDEX  
KEY_END SMALLINT NOT NULL
```

Number of columns in the index:

```
NB_COLS SMALLINT NOT NULL
```

Physical length of the index record:

```
LRECL SMALLINT NOT NULL
```

Total physical length of the key in the index:

```
KEYLEN SMALLINT NOT NULL
```

Indicates whether the index is user-defined value ('N') or system-defined (value 'Y'):

```
SYSCATOBJ CHAR(1) NOT NULL
```

Name of the table associated with the index:

```
TBNAME CHAR(18) NOT NULL
```

Name of the FKEY index associated with the current index. This information is used to enforce referential integrity:

```
FKEY_IND CHAR(18)
```

Name of the DATASPACE in the SYSDATASPACES table that physically stores the data:

```
IDTSPACE CHAR(18) NOT NULL (NSDB V.02)
```

## SYSSPACES Structure

TABLESPACE ID:

```
SPACEID CHAR(19) NOT NULL
```

Physical pathname of the TABLESPACE (Drive:\Directory\File.Ext):

```
PHYS_NAME CHAR(60) NOT NULL
```

3-state indicator used to describe the current state of the TABLESPACE:

```
TABLEUSAGE SMALLINT NOT NULL  
0 = FREE (no tables stored)  
1 = UNCLUSTERED (only one index stored)  
n = CLUSTERED (n indexes stored)
```

## SYSDATASPACES Structure

DATASPACE ID:

```
SPACEID CHAR(19) NOT NULL
```

Physical pathname of the DATASPACE (Drive:\Directory\File.Ext):

```
PHYS_NAME CHAR(60) NOT NULL
```

2-state indicator used to describe the current state of the DATASPACE

```
TABLEUSAGE SMALLINT NOT NULL  
0 = FREE (no tables stored)  
1 = BUSY (one table stored)
```

---



## Chapter 7

# Interface with NCL



See the "Service Libraries volume 2" manual for details on how to access NS-DB using the NCL language.





## Chapter 8

# NS-DB Interface for 3GLs (C, Pascal)



***This chapter  
explains***

- First point.
- Second point.
- ...
- Last point.

## Contents

Overview of the Interface.....	8-4
NS-DB API Entry Point .....	8-5
Z_COM Communication Area.....	8-6
ZIO Input-Output Area .....	8-7
REQ Request Area .....	8-8
Fx Function Code .....	8-10
DB_INIT Function	8-10
DB_TERM Function	8-10
DB_STARTRN Function	8-10
Call Arguments	8-11
DB_OPENCURSOR Function	8-11
Allocating a Cursor Implicitly	8-12
Allocating a Cursor Explicitly	8-12
DB_CLOSECURSOR Function	8-12
DB_STOPTRN Function	8-12
DB_COMPILE Function	8-12
DB_COMMIT Function	8-13
DB_ROLLBACK Function	8-13
DB_ISRT Function	8-13
DB_DLET Function	8-13
DB_SEARCH Function	8-13
DB_NEXT Function	8-14
DB_PREV Function	8-14
DB_MODIF Function	8-14
DB_GETFIRSTDB Function	8-14
DB_GETNEXTDB Function	8-14
DB_DROPDB Function	8-14
Accessing the Catalog Tables .....	8-15
Transactional Journaling.....	8-16
Definition	8-16
Implementation	8-16
Technical Description	8-16
Definition of a transaction	8-16
Definition of COMMIT	8-16

Definition of ROLLBACK	8-16
Starting a Transaction	8-16
Stopping a Transaction	8-17
Isolating Transactions and Visibility of Updated Information	8-17
➤ <i>Rules regarding visibility and accessibility</i>	
➤ <i>Dealing with unstable, uncommitted updates</i>	

---

## **Overview of the Interface**

The data manipulation language (DML) defines a set of primitives used to manipulate the occurrences stored in tables.

The NS-DB interface does not provide any set functions based on relational algebra.

Indexes cannot be accessed by programming. The optimum access method is automatically selected by the database manager according to the arguments specified in each query.

Any referential integrity constraints defined for the tables are notified to the programmer by the semantic checks performed between the tables.

### **Example**

If referential integrity is defined between the CLIENT and INVOICE table, any program that attempts to create an invoice for a non-existent client will receive an integrity error message.

## NS-DB API Entry Point

The only programming interface available for manipulating data stored in a database is the DB\_CALL procedure described below.

```
PROCEDURE DB_CALL
(
    Fx : DB_FX      ; (* Requested Function *)
    VAR Z_COM : DB_COM ; (* Database communication AREA*)
    VAR ZIO    ; (* INPUT OUTPUT Area associated with Fx *)
    ZIOSIZE : WORD ; (* Useful size of Zio *)
    VAR REQ
    REQSIZE : WORD
);
```

## Z\_COM Communication Area

The Z\_COM communication area enables the user program to communicate with the database interface.

For the DBMS, each Z\_COM area will be associated with a transaction ("Unit of work on a database"). All database update operations will be associated with the transaction that performs them.

Each COMMIT (confirmation) or ROLLBACK (cancellation) will always apply to the update operations performed by a transaction, which will either be permanently written to the database (COMMIT) or permanently removed from the database (ROLLBACK).

Most of the Z\_COM area contains fields that return information to the caller.

TRANS and TRANSINST contain information about the transaction that should never be updated by the caller (transaction handle). The address stored in TRANSINST is not part of the caller's memory area: a TRAP CODE will occur if an attempt is made to access the contents of this address.

```
DB COM = RECORD
  NA Trans : TR_ID ; (* Transaction ID at OPEN time *)
  NA TransInst : C_TRANSACTION ; (* Transaction Addr *)
  R_Info RetCode : DATABASE_RETCODE; (* DB Return Code *)
  R_Info Msg : DATABASE_MESSAGE ; (* Assoc Ret Msg *)
  DB STARTRN--->DbName:DATABASE_FNAME;(*Accessed DB Name*)
  (A) --->TableName:SQL_NAME ; (* Accessed TABLE *)
  R_Info Last Fx :DB_FX ; (* Last Performed *)
  R_Info LineNo :WORD ; (* Compiler Info *)
  R_Info ColNo :WORD ; (* Compiler Info *)
  R_Info SpaceCode:WORD; (* SPACE MNGT ERROR CODE *)
  R_Info IoERR :WORD; (* Space Mngt OS2 IOERR *)
  R_Info Elapse :LONGINT;(*Calls Duration In Ms*)
  R_Info IoRead :LONGINT;(*Nb Of Phys Read on disk*)
  R_Info IoWrite :LONGINT;(* Nb Of Phys Write on disk *)
  R_Info Kcalls :WORD ;(* Number OF KSDS API CALLS *)
  R_Info MemAllo :LONGINT;(* Current memory Used *)
  R_Info Cat Size :LONGINT;(* Size of catlg in bytes *)
  R_Info Cursor :WORD; (* Transaction cur Cursor *)
  Reserved:ARRAY[0..MAX_RESERVED] OF BYTE
                                          ; (*SYSTEM USAGE*)
END;
```

The symbol ---> stands for:

Set by the caller for the specified function(s).

The symbol (A) stands for:

Must be supplied by the caller for all DML functions (DB\_ISRT, DB\_DLET, DB\_SEARCH, DB\_NEXT, DB\_PREV and DB\_MODIF).

The abbreviation **R\_Info** stands for:

Information returned by the interface and available to the caller.

## **ZIO Input-Output Area**

ZIO is the input/output area used to transfer data between the DBMS and the program. The calling program is responsible for allocating this area.



The caller should define a sufficiently large output area to avoid memory overflow.

## REQ Request Area

REQ is the request area. It is only used by two types of requests:

- Compiler requests (Fx = DB\_COMPILE) where REQ is assumed to be a C language character string containing an SQL statement.
- Search requests (Fx = DB\_SEARCH) where REQ is assumed to be of the type: `ARRAY[1..N] OF DB_REQ`.

```
DB_OPER = (DB_IN, DB_INF, DB_SUP, DB_SUPEQU, DB_INFEQU, DB_DIFFER, DB_EQUAL) ;
```



The DB\_DIFFER operator results in a full sequential scan of the relational table.

The only difference between the DB\_IN and DB\_EQUAL operators is the way that they handle non-numeric strings:

- DB\_EQUAL searches for an exact match with the argument.
- DB\_IN returns all the strings that contain the argument.

### Example

```
(* SEARCH ARGUMENT STRUCTURE *)

DB_REQ =
RECORD
    Column: SQL_NAME; (*Table Column name to query*)
    Db Op : DB_OPER ; (* matching operator *)
    Nullv : BYTE ; (* NULL BYTE on IF = NULL_VALUE *)
CASE
    Tp : DATA_TYPE OF (* value to match *)
        T_CHAR      : ( Cs : CSTRING ) ;
        T_VARCHAR   : ( Ps : STRING ) ;
        T_SMALLINT  : ( I : INTEGER ) ;
        T_INTEGER   : ( Li : LONGINT ) ;
        T_FLOAT     : ( R : DOUBLE ) ;
END ;
```



The first column found that contains an empty C string ( "" ) is interpreted by the DBMS as being the end of the request.

Column stands for a column name, as defined in the DBMS catalog.

A column can either be queried for the value that it contains (Db\_Op + Value To Match) or to determine whether or not it contains a NULL VALUE.

For all NULL column values: Nullv = NOT\_NULL\_VALUE

For all NOT NULL column values: Nullv = NULL\_VALUE

Db\_Op stands for a relational operator applied to the value during the comparison.



Depending on the column's data type, the value compared is placed in Cs, Ps, I, Li or R.

For character strings, if the operator is DB\_IN, the comparison algorithm searches the database for all strings beginning with the value in the Ps or Cs variable.

## Fx Function Code

The Fx code defines the required function, which can be one of the following:

```
DB_Fx = (DB_INIT, DB_TERM, DB_STARTRN, DB_STOPTRN, DB_COMPILE,
DB_COMMIT, DB_ROLLBACK, DB_ISRT,
DB_DLET, DB_SEARCH, DB_NEXT, DB_PREV, DB_MODIF, DB_GETFIRSTDB,
DB_GETNEXTDB, DB_DROPDB) ;
```

### DB\_INIT Function

The DB\_INIT function connects a client to the data server. This is the first compulsory call that a client program must make before it can use the services of a data server.



In multi-user mode (equivalent to DB\_INIT "NS02DBCL.SERVER"), the Z\_COM area must be set as follows:

```
DB_STARTRN--->DbName      : DLL name (NS02DBCL)
(A)      --->TableName    : Server name
```

### DB\_TERM Function

The DB\_TERM function terminates NS-DB.



None of the fields in the communication area need to be set before using this function.

### DB\_STARTRN Function

The DB\_STARTRN function starts a transaction on a database.

A very precise definition of a transaction has been given by C-J DATE: UNIT OF WORK ON A DATABASE.

Hence, a transaction is a set of logically related instructions sent to a specific database.

The notion of a "transaction" is distinct from the notion of a "program". The same program can execute several transactions on the same database. However, a transaction is the only entity recognized by the database integrity functions (COMMIT and ROLLBACK).

When a COMMIT statement is executed, all the update operations performed by the transaction that requested the COMMIT are applied to the database and are visible to all other concurrent transactions.

When the ROLLBACK statement is executed, it cancels all the update operations performed by the transaction that requested the ROLLBACK since the last COMMIT.

Therefore, in a multi-user context, the notion of a program opening a database does not apply. Each program starts one or more transactions on the databases that it wants to use.

## Call Arguments

The database name is Z\_COM.DbName

The transaction code is returned by Z\_COM.Trans, which contains a unique transaction number allocated by the interface.

Two data items must be passed back by the calling program in the communication area:

- The transaction code (DB\_COM.Trans).
- The cursor handle (used to maintain current read positions).

## DB\_OPENCURSOR Function

A cursor is a database resource used to maintain the current position reached by one of the following functions:

- DB\_SEARCH, which sets a current position in the database.  
If only one cursor resource is used, then only one current position can be maintained. This position is identified by a handle stored in DB\_COM.Cursor.  
N.B. each new DB\_SEARCH command will lose the position held by the previous DB\_SEARCH.
- DB\_NEXT, which uses the handle in DB\_COM.Cursor to maintain its position.
- DB\_PREV, which uses the handle in DB\_COM.Cursor to maintain its position.

You must define one cursor for each simultaneous current position required in the database.

To scan several tables concurrently, you need to define one cursor for each simultaneous current position.

There are two ways of allocating a cursor: **implicit allocation** and **explicit allocation**.

---

### **Allocating a Cursor Implicitly**

If you call the DB\_SEARCH function while the DB\_COM.Cursor field contains an undefined handle (i.e. not allocated earlier), the system allocates a new cursor and places its handle in the DB\_COM.Cursor field.

### **Allocating a Cursor Explicitly**

Before calling DB\_SEARCH, an explicit call to the DB\_OPENCURSOR function returns the handle of the allocated cursor in the DB\_COM.Cursor field.

This handle will then be used by the DB\_SEARCH, DB\_NEXT and DB\_PREV commands.

NSDB can then associate this handle with the position reached at the end of each DB\_SEARCH, DB\_NEXT or DB\_PREV command. This position is automatically restored at the beginning of the DB\_NEXT or DB\_PREV command that references this handle in the DB\_COM.Cursor field.

## **DB\_CLOSECURSOR Function**

The DB\_CLOSECURSOR function closes the cursor specified in DB\_COM.Cursor and frees the memory resources allocated to it.

## **DB\_STOPTRN Function**

The DB\_STOPTRN function stops a transaction started earlier by the DB\_STARTRN function. This function automatically commits the transaction.

When this function is called, the ZIO communication area should only contain a valid transaction code in DB\_COM.Trans (returned earlier by the DBMS after the DB\_STARTRN command).

## **DB\_COMPILE Function**

The DB\_COMPILE function compiles and executes an SQL statement for a given transaction.



This command only performs SQL compilations involving DATA DEFINITION LANGUAGE (DDL) commands (used to define tables and indexes).

Only one statement can be compiled each time this function is called (an SQL statement is a semantic unit (CREATE TABLE, CREATE INDEX, etc.) that ends in a semi-colon (;)).

- REQ: contains the SQL source to be compiled in a C character string (ending in \$00).
- Z\_COM: contains the return code (Retcode), full description and line and column number of any errors detected.
- ZIO: is an input/output area designed to hold retrieved or updated records. It is not used by the functions that compile the Data Description Language (DDL).

## DB\_COMMIT Function

The DB\_COMMIT function commits all the database update operations performed by a transaction.

## DB\_ROLLBACK Function

The DB\_ROLLBACK function rolls back all the database update operations performed by a transaction.

## DB\_ISRT Function

The DB\_ISRT function inserts an occurrence of a table into the database.

## DB\_DLET Function

The DB\_DLET function deletes an occurrence of a table from the database.

## DB\_SEARCH Function

The DB\_SEARCH function searches for the first occurrence of a table that matches a set of criteria defined by the calling program.



See the DB\_OPENCURSOR function for details on how to handle several searches simultaneously.

## DB\_NEXT Function

The DB\_NEXT function searches for the next occurrence (DB\_SEARCH required beforehand).

## DB\_PREV Function

The DB\_PREV function searches for the previous occurrence (DB\_SEARCH required beforehand).

## DB\_MODIF Function

The DB\_MODIF function modifies an existing occurrence (the PRIMARY KEY must exist).

## DB\_GETFIRSTDB Function

The DB\_GETFIRSTDB function returns the name of the first database that can be used in the system (a SYSDATABASE structure is returned).

If no databases have been defined, the error returned will be:  
DB\_RECORD\_NOT\_FOUND.

## DB\_GETNEXTDB Function

The DB\_GETNEXTDB function returns the next SYSDATABASE structure relative to the one stored in the ZIO area.

If the SYSDATABASE in the ZIO area is the last one in the list of databases available in the system, the error returned will be: DB\_RECORD\_NOT\_FOUND.

## DB\_DROPDB Function

The DB\_DROPDB function physically deletes the database together with all of its TABLESPACES.



All the data in the database will be deleted.

## **Accessing the Catalog Tables**

The system uses a number of internal relational tables to describe the information stored in the DBMS.

Cf. Chapter 6 *System Catalog*.

The information in these tables can be read using the standard navigation verbs: DB\_SEARCH , DB\_NEXT and DB\_PREV.

## Transactional Journalling

### Definition

Transactional journalling enables a transaction to cancel (UNDO or ROLLBACK) or confirm (DO or COMMIT) the updates that it has performed.

### Implementation

Use the DB\_COMMIT and DB\_ROLLBACK statements in the database's programming interface.

### Technical Description

#### Definition of a transaction

A transaction is a basic unit of database access operations performed by a user. It is the database equivalent of an OS/2 THREAD or TELECOM SESSION.

#### Definition of COMMIT

When a COMMIT statement is issued, all the updates performed by a transaction since the last COMMIT or ROLLBACK statement or since the start of the transaction are permanently applied to the database (DO).

#### Definition of ROLLBACK

When a ROLLBACK statement is issued, all the updates performed by a transaction since the last COMMIT or ROLLBACK statement or since the start of the transaction are removed from the database (UNDO).

#### Starting a Transaction

Transactions are started by the DB\_STARTRN statement, which allows a programmed entity to access a database object. (This statement is comparable with a 'database open'.)

The same process can start several transactions on one or more databases.



## Stopping a Transaction

When a transaction is stopped, an automatic COMMIT or ROLLBACK is performed, depending on how the transaction terminated. The following events stop a transaction:

- For a COMMIT statement:
  - Explicit STOP TRANSACTION request
  - Termination of the PROCESS that started the transaction: DOSEXIT (normal termination)
- For a ROLLBACK statement:
  - Explicit STOP TRANSACTION request
  - Termination of the PROCESS that started the transaction: TRAP CODE, CTRLC, etc. (abnormal termination)

## Isolating Transactions and Visibility of Updated Information

Transactions are totally isolated and protected from each other by the database manager.

### ➤ *Rules regarding visibility and accessibility*

The DB\_MODIF, DB\_ISRT and DB\_DLET statements indicate an intention to update data. As a result, the system locks the physical pages containing:

- The table rows
- The indexes affected by the update.

If a table row is being updated by a transaction, the page that contains it is reserved by the transaction until the next COMMIT or ROLLBACK requested by this transaction. After this, the resource will be freed.

### ➤ *Dealing with unstable, uncommitted updates*

- If a transaction updates information in the database and another transaction wants to read this information at the same time, there are three possible strategies:
    - Authorize access to the data, bearing in mind the risk that the update being performed by the first transaction may not be completed, in which case the information read will be incorrect.
    - Prohibit access to the information being updated by returning an error code.
    - Return the information stored on disk to the requester (opposite risk to first strategy).
  - Abnormal termination of the server
-

In addition to any problems regarding the physical stability of the journal, an abnormal termination by the server has the following consequences:

- All the server transactions in progress that have not been committed are declared as IN FLIGHT (unstable).
- All IN FLIGHT transactions will be rolled back and any update operations in progress will be lost when the system is rebooted.

## Chapter 9

# NS-DB Utilities



This chapter describes the utilities that complement the NS-DB interface.

These utilities make it easier to implement and operate NS-DB.

***This chapter  
explains:***

NS-DB Utilities:

- DBCHCK.
- DBSERV.
- DCLGEN.
- EXPORT.
- IMPORT.
- REMSRV.
- SHUTDOWN.
- SQLEXE.
- TCPSRVDB.

## Contents

DBCHCK Utility .....	9-4
Start-up syntax      9-4	
Description      9-4	
DBSERV Utility .....	9-7
Start-up syntax      9-7	
Description      9-7	
Details on network operation      9-8	
Concurrent access control      9-9	
Differences between network and standalone mode      9-10	
Automatic suffixing of internal names      9-10	
DCLGEN Utility .....	9-13
Start-up syntax      9-13	
Description      9-13	
Example      9-13	
EXPORT Utility .....	9-14
Start-up syntax      9-14	
Description      9-14	
Example      9-14	
IMPORT Utility .....	9-15
Start-up syntax      9-15	
Description      9-15	
Example      9-16	
REMSRV Utility .....	9-17
Start-up syntax      9-17	
Description      9-17	
Example      9-17	
SHUTDOWN Utility .....	9-18
Start-up syntax      9-18	
Description      9-18	
Examples      9-18	
SQLEXE Utility .....	9-19
Command syntax      9-19	
Description      9-19	

---

TCPSRVDB Utility .....	9-20
Start-up Syntax    9-20	
Description   9-20	
Example       9-20	

---

## DBCHCK Utility

### Start-up syntax

**DBCHCK** *Database-Name*

### Description

This utility checks an NS-DB database for physical errors.

**1. Database-Name :**

is the logical database name, as specified in the initialization file.

**2. DBCHCK :**

Checks an NS-DB database for physical errors.

Analyses the status of each physical space associated with the logical database passed as a parameter.

Can be run automatically by a system command file. If an error is detected, it returns the error code '+16'.

If any of the following error messages are sent to the screen, this means that the database has an invalid structure and needs to be restored from a backup copy.

```
** SQLCCK-003 PHYSICAL ERRORS on Drive:\Path\File
** Please contact Nat System for informations
:
:
:TOTAL BAD SPACES : X
Ns-Db Check ended ; Duration : XXXXX Ms
```

**Cause:** General message

```
** SQLCCK-004 EMPTY CI FOUND : Page-Nbr
```

**Cause:** A page containing no data has not been freed by SPACE MANAGER.

```
** SQLCCK-005 KEY NODE INVALID ORDER : Page-Nbr
```

**Cause:** The order of the keys in the tree is invalid (bad tree structure).

```
** SQLCCK-006 INVALID TREE LEVEL : Page-Nbr
```

Invalid level number for the page in the balanced tree(bad tree structure).

**\*\* SQLCCK-007 CI INTERNAL INV KEY SEQUENCE : Page-Nbr**

**Cause:** Invalid key sequence in the page.

**\*\* SQLCCK-008 CI BAD INTERNAL STRUCTURE : Page-Nbr**

**Cause:** Abnormal physical page structure.

UNUSED CI LIST(Must be empty) :

Page-Nbr

:

:

Page-Nbr

**Cause:** All the pages listed are not referenced in the tree. One or more branches have been lost.

The following message should be analyzed according to the context:

**\*\* SQLCCK-009 SPACE OS DEPENDENT ERROR : Os-ErrNum**

The following messages apply to the retrieval of free space:

**\*\* SQLCCK-010 INVALID FREE CI TYPE**

**\*\* SQLCCK-011 FREE CI ID INVALID**

**\*\* SQLCCK-012 INVALID FREE CI QUANTITY**

**\*\* SQLCCK-013 LAST CI HEAD MISMATCH**

**\*\* SQLCCK-014 INVALID GLOBAL KEY SEQUENCE**

The following negative error codes, returned by the utility, do not result from an invalid physical database structure.

**\*\* SQLCCK-001 PARAMETER MISSING**

**Cause:** The Database\_Name parameter has been omitted.

**\*\* SQLCCK-002 BASIC DATABASE CATALOG ACCESS ERROR :**

**Cause:** Error opening the database to be scanned.

The serious errors that occur when an NS-DB database is used may be due to one of the following reasons:

- A sudden power failure in the machine while a COMMIT statement is being executed on the database.

- A physical error on the storage medium.
- A bug in Space Manager.

The average time taken for the utility to run depends on the size of the database being checked.

The following results have been obtained on an IBM PS70 machine:

- 1 Mb database: 26 seconds.
- 15 Mb database: 7.8 minutes (For 7400 pages of 2048 Kb).



# DBSERV Utility

## Start-up syntax

**DBSERV** [[*ServId*] [**TRACE**|**NOTRACE**] [*Net-Message-Size*]]

## Description

The **DBSERV** utility is an NS-DB database server.

### DBSERV:

- Starts up by using the information in the NS-DB initialization file.
- Initializes the NS-DB engine.
- Listens to NS-DB clients by submitting requests to the NS-DB server *ServId*.

Client requests are routed by the NSnnDBCL libraries or by a remote NS-DB engine whose initialization file contains a database defined as follows:

- For NETBIOS :

```
Sysid='CLIENT' ,  
NETDLL='ACSNETB' ;  
:  
:  
Mydb='SERVER',REMOTE ;
```

- For TCP/IP :

```
Sysid='CLINAME:2000' ,  
NETDLL='TCPIPDLL' ;  
:  
:  
Mydb='SERVER:2200',TCPIP ;
```

The initialization file used by **DBSERV** must also contain the following definition for the Mydb logical database:

- For NETBIOS:

```
Sysid='SERVER',  
NETDLL='ACSNETB' ,  
NETLSTN='D:\NSDB\REMSRV.EXE' ;  
:  
:  
Mydb='D:\NSDB\MYDB.DAT',CASESENS ;
```

- For TCP/IP :
-

```
Sysid='SERVER:2200',  
NETDLL='TCPIP.DLL',  
NETLSTN='c:\NSDB\TCPSRVDB.EXE' ;  
:  
:  
Mydb='c:\NSDB\MYDB.DAT',CASESENS ;
```

The client/server link is established using the logical database name, which must be identical in both initialization files.

An NS-DB server redirects NS-DB calls using a stored procedure mechanism (STANDARD SYNCHRONOUS RPC (Remote Procedure Call)).

RPC is managed by a *NatStar / Communications Services* library required to implement NSnnSCDB on a network.



nn stands for the version number of the interface that you have installed.

The application code is totally independent of the database location.

### Details on network operation



NS-DB can be used on all networks with a NETBIOS session interface (ISO/OSI LEVEL 5, IEEE 802.2) or with TCP/IP.

This covers most commercially available networks, regardless of the physical layer that they use (TOKEN RING, ETHERNET, etc.).

Examples: NETBIOS interfaces are supplied by IBM with COMMUNICATION MANAGER and by MICROSOFT with LAN MANAGER.

Tasks performed by the drivers when the machine is initialized:

- IBM COM MANAGER or MICROSOFT LAN MAN V 2.1:  
The driver connects to the network. In the event of a fault (e.g. unplugged cable) an error is reported when the machine starts up.
- MICROSOFT LAN MAN V2.0:  
The network connection is established by the NET START Workstation utility, which must be run before using the MICROSOFT NETBIOS DLL and operating NS-DB in network mode.
- NOVELL NETWARE:

A driver supplied by NOVELL performs the NETBIOS emulation for the IPX/SPX protocol. The NETBIOS library (NBNETWR.DLL or NSnnNWR.DLL) is supplied by Nat System to ensure compatibility with the IBM APIs.

- **BEFORE INSTALLING NS-DB ON A NETWORK:**

You need to know the name of the NETBIOS or TCP/IP library used by the network supplier.

*Exemples :*

<i>IBM Communication Manager</i>	<i>-&gt; ACSNETB</i>
<i>Microsoft Lan Manager 2.0</i>	<i>-&gt; NB30</i>
<i>Microsoft Lan Manager 2.1</i>	<i>-&gt; ACSNETB</i>
<i>NETWARE NOVEL</i>	<i>-&gt; NBNETWR</i>
<i>IBM/TCP</i>	<i>-&gt; TCPIPDLL</i>



NBNETWR or NSnnNWR are supplied by Nat System.

## Concurrent access control

In multi-user mode, it is essential to control concurrent access to the same table row.

The CURSOR STABILITY or REPEATABLE READ locking mechanisms used by existing DBMSs (DBM, ORACLE, etc.) are unsatisfactory. They require the developer to code SQL in an unintuitive way (e.g. committing SELECT statements to free resources). Their locking strategies reduce data server accessibility and, ultimately, the fluidity of the system depends entirely on the SQL code in the applications.

NS-DB uses two locking strategies:

- **Logical: TIME-STAMPING**  
Time-stamping involves adding an extra SYSTIMESTAMP column to each table that requires this feature.
- **Physical: PAGE-LOCKING**  
Page-locking is applied to unstable information that has not been committed.

Until an update operation is confirmed by a commit, the physical page containing the data is protected against updates by concurrent transactions. The following error message is returned: PHYSICAL PAGE IN USE SQLEX-057



Updates must be performed in an interrupted stream terminated by a COMMIT or ROLLBACK.

When a transaction receives this message, it must execute a ROLLBACK and then re-execute the update.



TIME-STAMPING and PAGE-LOCKING are the two mechanisms that NS-DB uses to control concurrent access.

## Differences between network and standalone mode

The only major difference is the time taken to transmit data across the line. Even if high speeds are achieved (e.g. 4 Mb/s or 16 Mb/s on a TOKEN RING network), this will always be much slower than the transfer speeds obtained for memory and disk access on a machine in standalone mode.

Since there is no limit to the number of times data can be redirected on a network database, the only argument against this type of practice is the resulting deterioration in response time:

*Example:*

*CLIENT1 can define a client database as remote and redirect it to SERV1  
(CLIENT='SERV1',REMOTE;).*

*SERV1 can also redefine the client database as remote and redirect it to SERV2  
(CLIENT='SERV2',REMOTE;),*

*and so on until:*

*SERVN defines CLIENT locally  
(CLIENT='C:\CLIENT.DTA',CASEINSENS ;)*

*The OS/2 NS-DB servers are not dedicated.*

In the above example, a user can execute an application on the machine containing the SERVN server and connect locally to this server in CLIENT mode.

## Automatic suffixing of internal names

When the NETBIOS connection is established on the network, the following suffixes are automatically added by NS-DB to the SYSID defined in the initialization file. The suffix depends on the machine's function:

CLIENT	NC significant Network Client
SERVEUR REMOTE	NR significant Network Remote

These suffixes are used internally to ensure that the NETBIOS name is unique when a machine has a dual role: client of a remote database and data server for remote clients. In this case, two names are generated on the network: MACHINENC and MACHINENR.

All the NETBIOS error codes for the telecommunications interface start at 8000 and match the standard codes used by the NETBIOS driver manufacturer.

The codes returned by IBM drivers are listed below.

The other NETBIOS API suppliers (MICROSOFT, NOVELL, etc.) normally use the same error codes.

REMSRV OR NS-DB CLIENT ERROR CODES:

- 8001 \*\* ILLEGAL BUFFER LENGTH
  - 8003 \*\* INVALID COMMAND
  - 8005 \*\* COMMAND TIMED OUT
  - 8006 \*\* MESSAGE INCOMPLETE
  - 8007 \*\* DATA FOR ONE OR MORE SEND TYPE NO.ACK COMMANDS  
WAS NOT RECEIVED
  - 8008 \*\* ILLEGAL LOCAL SESSION NUMBER
  - 8009 \*\* NO RESOURCE AVAILABLE
  - 8010 \*\* SESSION CLOSED
  - 8011 \*\* COMMAND CANCELED
  - 8013 \*\* DUPLICATE NAME IN LOCAL NAME TABLE
  - 8014 \*\* NAME TABLE FULL
  - 8015 \*\* COMMAND COMPLETED-NAME HAS ACTIVE SESSION AND IS  
NOW DE-REGISTERED
  - 8017 \*\* LOCAL SESSION TABLE FULL
  - 8018 \*\* SESSION OPEN REJECTED
  - 8019 \*\* ILLEGAL NAME NUMBER
  - 8020 \*\* CANNOT FIND NAME CALLED
  - 8021 \*\* NAME NOT FOUND OR CANNOT SPECIFY "\*" OR NULL
  - 8022 \*\* NAME IN USE ON REMOTE NETBIOS
  - 8023 \*\* NAME DELETED
  - 8024 \*\* SESSION ENDED ABNORMALLY
  - 8025 \*\* NAME CONFLICT DETECTED
  - 8033 \*\* INTERFACE BUSY
  - 8034 \*\* TOO MANY COMMANDS OUTSTANDING
  - 8035 \*\* INVALID NUMBER IN NCB-LANA-NUM FIELD
  - 8036 \*\* COMMAND COMPLETED WHILE CANCEL OCCURRING
  - 8038 \*\* COMMAND NOT VALID TO CANCEL
  - 8048 \*\* NAME DEFINED BY ANOTHER ENVIRONMENT
  - 8052 \*\* ENVIRONMENT NOT DEFINED, RESET MUST BE ISSUED
-

8053 \*\* REQUIRED OPERATING SYSTEM RESOURCES EXHAUSTED,  
RETRY LATER

8054 \*\* MAXIMUM APPLICATIONS EXCEEDED

8055 \*\* NO SAPS AVAILABLE FOR NETBIOS

8056 \*\* REQUESTED RESOURCE(S) NOT AVAILABLE

8057 \*\* INVALID NCB ADDRESS OR LENGTH DOES NOT FIT IN  
SEGMENT

8058 \*\* RESET MAY NOT BE ISSUED FROM A NETBIOS ADAPTER  
APPENDAGE

8059 \*\* INVALID NCB-DD-ID VALUE

8060 \*\* NETBIOS ATTEMPTED TO LOCK USER STORAGE AND THE  
LOCK FAILED

8063 \*\* NETBIOS DEVICE DRIVER OPEN ERROR

8064 \*\* OS/2 ERROR DETECTED

8078 \*\* NETWORK STATUS-ONE OR MORE OF BITS 12,14,15 ON FOR  
LONGER THAN 60 SEC

8079 \*\* NETWORK STATUS-ONE OR MORE OF BITS 8-11 ON

8246 \*\* UNEXPECTED ERROR ON CCB COMPLETION

8247 \*\* ERROR ON IMPLICIT DIR.INITIALIZE

8248 \*\* ERROR ON IMPLICIT DIR.OPEN.ADAPTER

8249 \*\* ADAPTER SUPPORT SOFTWARE INTERNAL ERROR

8250 \*\* ADAPTER CHECK

8251 \*\* NETBIOS PROGRAM NOT LOADED IN PC

8252 \*\* DIR.OPEN.ADAPTER OR DLC.OPEN.SAP FAILED, CHECK  
PARAMETERS

8253 \*\* UNEXPECTED ADAPTER CLOSE

8254 \*\* NETBIOS NOT OPERATIONAL AND APPLICATION PROGRAM  
EXPLICITLY OPENED THE ADAPTER

8500 \*\* NETBIOS PROCEDURE OR NETBIOS DLL NOT FOUND

# DCLGEN Utility

## Start-up syntax

```
DCLGEN Database-Name LANGUAGE [[Database-Table | ALL]
[[Include-FileName] [ServerName]]]
```

## Description

The **DCLGEN** utility allows the programmer to convert the table structure definitions in the NS-DB catalog into declaratives in a specified programming language.

- *Database-Name*  
is the logical database name, as defined in the initialization file.
- **LANGUAGE**  
selects the language in which the declaratives will be generated. **DCLGEN** currently supports the following language parameters: C, PASCAL, PASCALAXA and NCL.  
If ALL is specified as the third parameter, declaratives will be generated for all the tables defined in the database, including the SYSTEM CATALOG tables.  
If the name of the file that will contain the declaratives produced by the utility is omitted, the declaratives will be generated in the standard output file.  
If *ServerName* is supplied, the database identified by *Database\_Name* will be searched for on the specified server using a client connection.

## Example

```
DCLGEN Mydb NCL ALL MYDB.NCL MYSERV
```

Generates the MYDB.NCL file containing all the NCL SEGMENTS that define the table structures in the MYDB database located on the MYSERV server.

## EXPORT Utility



### Start-up syntax

**EXPORT** *ExportFname DatabaseName NewTableName*

### Description

- *ExportFname*  
name of the file containing the table to be exported. This name has the following format: Drive:\Path\Fname.Ftype.
- *DatabaseName*  
name of the database containing the table to be exported. This is the logical name defined in the NSDB.INI file.
- *NewTableName*  
new name for the exported table.

**EXPORT** is a utility for exporting tables.

The current file format supported is IBM IXF PC (IBM and DB2-specific file format used for importing and exporting tables). This format is produced by the IBM DB2 and DBM database manager utilities.



Under Windows, EXPORT is part of the NSQBE utility

### Example

```
EXPORT C:\DBBASE.DAT MYDB MYTABLE
```





## IMPORT Utility



### Start-up syntax

```
IMPORT      ImportFname LogicalDb NewTableName PKeyCols  
            [TABLESPACE]
```

### Description

- *ImportFname*  
name of the file containing the exported database. This name has the following format: Drive:\Path\Fname.Ftype.
- *LogicalDB*  
logical name of the NSDB database that will import the file.  
 This name must be defined in the initialization file.
- *NewTableName*  
name of the table in the database identified by LOGICALDB.  
 If this name already exists in LOGICALDB, the user will be prompted to delete the existing table first.
- *PKeyCols*  
defines the primary key of the imported table. NS-DB always requires a PRIMARY KEY for each table. Consequently, a primary key must be defined for the new imported table, using the following syntax on the command line:  
`@ClusterNumber@Col, @ClusterNumber@Col.....`
- **TABLESPACE**  
by default, the table will be imported into the PRIMARY SPACE. If you want to import the table into another TABLESPACE, you must specify its logical name using this parameter on the command line.



Under Windows, IMPORT is part of the NSQBE utility.

The **IMPORT** utility allows you to import tables from commercially available DBMSs.

The current file format supported is IBM IXF PC (IBM and DB2-specific file format used for importing and exporting tables). This format is produced by the IBM DB2 and DBM database manager utilities.

## Example

```
IMPORT C:\DBBASE.IXF MYDB MYTABLE @12@PRIMARYCOL SPACE4
```

## REMSRV Utility



### Start-up syntax

```
REMSRV [TRACE | NOTRACE] [DllName] [Node] [LocSrv]  
[MaxCmds] [MaxSess] [MaxNames]
```

### Description

The **REMSRV** utility is a NetBIOS listener program for database servers.

**REMSRV** is automatically run when DBSERV is started up, using the parameters specified in the initialization file ([System] group, SYSID verb, NETLSTN variable).

It is rarely started by the user.

### Example

```
[System]  
dbdir='C:\NSDB;  
Sysid='SRVNAME',  
NETDLL='NB30',/* for Microsoft LAN MAN 2.0 */  
NETLSTN='C:\NSDB\REMSRV.EXE';
```

## SHUTDOWN Utility

### Start-up syntax

**SHUTDOWN** *ServerName* [**NETWORKING** *NetBiosDllName*]

### Description

The **SHUTDOWN** utility allows you to shut down an NS-DB server started earlier by the DBSERV command.

**SHUTDOWN** can either be used for a local machine (default) or a remote machine. In the latter case, the **NETWORKING** option must be added to specify the name of the NETBIOS DLL used in the communication process.

### Examples

```
; Shut down local server:
SHUTDOWN SERVLOC

; Shut down a remote server
; on an IBM node

COMMUNICATION MANAGER:
SHUTDOWN SERVREM NETWORKING ACSNETB
```

# SQLEXE Utility

## Command syntax

**SQLEXE** *SourceFileName DatabaseName*

## Description

The **SQLEXE** utility allows you to submit a stream of SQL commands stored in a source file.

Only statements belonging to the Data Description Language can appear in the source file.

*DatabaseName* is the logical database name, as defined in the initialization file.

The SQL statements in the source file will be sent to the database identified by *DatabaseName*.



Each SQL statement must end with a semi-colon.

---

## TCPSRVDB Utility

### Start-up Syntax

TCPSRVDB [TRACE | NOTRACE] [DllName] [Node] [LocSrv] [MaxSess]

### Description

The **TCPSRVDB** utility is a TCP/IP listener program for database servers.

Using the parameters specified in the initialization file, **TCPSRVDB** is automatically started up after starting up the DBSERV utility ([System] group, SysIdverb, NETLSTN variable).

It is rarely started up by the user.

### Example

```
[System]
dbdir='C:\NSDB;
Sysid='SRVNAME:2200',
NETDLL='TCPIPDL', /* for IBM/TCP */
NETLSTN='C:\NSDB\TCPSRVDB.EXE';
```

## Chapter 10

# NSQBE Utility



***This chapter  
explains***

- Components of NSQBE.
- Using NSQBE.

## Contents

Overview.....	10-7
Components of NSQBE.....	10-8
Main Window            10-8	
Title Bar    10-8	
Menu Bar    10-8	
➤ <i>File Menu</i>	
➤ <i>Exit Menu</i>	
Icon Bar    10-9	
Data Base List Dialog Box            10-9	
Utilities Menu            10-10	
➤ <i>Check</i>	
➤ <i>Start transaction</i>	
➤ <i>sTats...</i>	
Exit Menu    10-11	
<i>DatabaseName Transaction N</i> Dialog Box    10-11	
Menu Bar    10-12	
Rows Menu 10-12	
➤ <i>New...</i>	
➤ <i>Commit transaction</i>	
➤ <i>Rollback transaction</i>	
➤ <i>Search</i>	
Tables Menu            10-13	
➤ <i>Info</i>	
➤ <i>Dclgen</i>	
➤ <i>Import</i>	
➤ <i>Export</i>	
➤ <i>New</i>	
➤ <i>Reorg</i>	
Exit Menu    10-14	
<i>Store Rows on DatabaseName.TableName</i> Dialog Box    10-14	
Columns    10-15	
Left/Right    10-15	
Store        10-15	
Cancel       10-15	
<i>Selection on DatabaseName.TableName</i> Dialog Box            10-15	
Table columns            10-16	
Projections    10-16	
Selections...            10-16	
Selection predicate    10-16	
Remove        10-17	



Projected columns 10-17

➤ *Remove*

Selection 10-17

➤ *Result window size*

➤ *Start*

➤ *Cancel*

### ***Selection Result*** Dialog Box 10-17

Selection Information 10-18

➤ *Used Index*

➤ *Max*

➤ *Min*

➤ *Average*

➤ *Total time*

➤ *Search Time*

Selected list 10-19

➤ *Next*

➤ *Next*

➤ *Cancel*

### ***DatabaseName.TableName*** Dialog Box 10-19

Table Name 10-20

Data storage 10-20

Column Info 10-20

➤ *Column Name*

➤ *Data Type*

➤ *Size*

➤ *Null value*

➤ *Add column*

Indexes 10-21

➤ *Add index...*

Column List 10-21

Create 10-21

Alter 10-21

Drop 10-21

Cancel 10-22

### ***TableName.IndexName*** Dialog Box 10-22

Index Name 10-22

Index Info 10-23

➤ *Table Space*

➤ *Primary Key*

➤ *Unique*

Column List / Indexed Columns 10-23

➤ *Index Column*

➤ *Indexed Columns*

Foreign Key 10-24

#### 10-4 NSQBE Utility

---

Create 10-24  
Drop 10-24  
Cancel 10-24

#### *New Index definition on TableName* Dialog Box 10-24

Index Name 10-25  
Index Info 10-25  
    ➤ *Table Space*  
    ➤ *Primary Key*  
    ➤ *Unique*  
Column List / Indexed Columns 10-26  
    ➤ *Index Column*  
Indexed Columns 10-26  
Foreign Key 10-26  
    ➤ *References*  
    ➤ *Delete Rule*  
    ➤ *Update rule*  
Create 10-27  
Drop 10-27  
Cancel 10-27

#### *DatabaseName.NewTable* Dialog Box 10-27

Table Name 10-28  
Data storage 10-28  
    ➤ *Data Space (NS-DB V.02)*  
Column Info 10-28  
    ➤ *Column Name*  
    ➤ *Data Type*  
    ➤ *Size*  
    ➤ *Null value*  
    ➤ *Add column*  
Indexes 10-29  
    ➤ *Add index*  
Column List 10-29  
Create 10-29  
Alter 10-30  
Drop 10-30  
Cancel 10-30

#### *DCLRES* Dialog Box 10-30

NCL Language 10-30  
C Language 10-30  
Pascal Language 10-30  
Save as 10-31  
Cancel 10-31

#### *Import Tables from IXF Format* Dialog Box 10-31

File Name 10-32

---

Current Directory	10-32
Files	10-32
Directories	10-32
Import	10-32
Cancel	10-32
<b>Import from Drive:\Path\Fname.IXF Dialog Box</b>	<b>10-32</b>
Table Name	10-33
Storage location	10-33
➤ Destination TABLESPACE	
➤ Destination DATASPACE (NS-DB V.02)	
Primary Key Election	10-33
OK	10-34
Cancel	10-34
<b>Export Table Using IXF/PC Format Dialog Box</b>	<b>10-34</b>
File Name	10-35
Current Directory	10-35
Files	10-35
Directories	10-35
Export	10-35
Cancel	10-35
<b>Monitor Information Dialog Box</b>	<b>10-35</b>
Monitor information	10-36
➤ Monitor name	
OK	10-36
Cancel	10-36
<b>Monitor Activity Dialog Box</b>	<b>10-36</b>
Global Activity	10-37
➤ Total transactions	
➤ Total Databases	
Active transactions	10-37
➤ TransactionId	
➤ DataBase	
➤ Active cursors	
➤ Cursors memory	
Server activity	10-38
Refresh	10-38
Cancel	10-38
<b>Transaction N on DatabaseName Dialog Box</b>	<b>10-38</b>
Counts	10-39
Total time	10-39
Avg time	10-39
Max time	10-39
Cancel	10-39
<b>Server Activity Dialog Box</b>	<b>10-39</b>

---

## 10-6    *NSQBE Utility*

---

Total calls to server	10-40
Total calls to database	10-40
Total Elapse in database	10-41
Max Elapse in database	10-41
Avg Elapse in database	10-41
Total Elapse in server	10-41
Max Elapse in server	10-41
Avg Elapse in server	10-41
Current number of clients	10-41
Max number of clients in session	10-41

## Using NSQBE..... 10-42

Opening or Creating a Database	10-42
Checking a Database for Physical Errors	10-42
Checking a Database for Logical Errors	10-43
Starting a Transaction	10-43
Adding a Row to a Table	10-44
Commit and Rollback	10-44
Executing a Query	10-45
Projecting Columns	10-45
Selecting a Search Predicate	10-46
Creating a Table	10-47
Creating a DATASPACE (NS-DB V.02)	10-49
Creating a TABLESPACE	10-49
Dropping a Table	10-49
Adding a New Column to a Table	10-50
Adding an Index	10-51
Generating Data Structures in NCL, C and Pascal	10-52
Importing a Table	10-52
Exporting a Table	10-53
Displaying Monitor Activity	10-54
Displaying Server Activity	10-55

## Overview

NSQBE provides a user-friendly and standard way of accessing and manipulating NS-DB databases, thanks to its graphical interface and the fact that it requires no prior knowledge of SQL.

NSQBE combines all the features provided by the utilities (see chapter 9), supplemented by functions for accessing, creating and updating databases.

NSQBE is a particularly effective tool for producing and analyzing test data for database applications.

It provides all the following functions:

- Opening and creating databases.
- Checking databases for physical errors.
- Starting transactions.
- Creating and deleting tables.
- Updating the data in a table.
- Set constraints.
- Adding indexes.
- Executing queries.
- Importing and exporting tables.
- Generating NCL, C or Pascal data structures.
- Monitoring activity.
- Analyzing response times for database applications.

NSQBE comprises a single executable file (NSQBE.EXE), which uses the NSnnQBE DLL together with all the NS-DB libraries.



nn stands for the version number of the interface that you have installed.

NSQBE can be started up from a full-screen session or program group (NS-DK Version 1.01B or higher, and for NatStar versions 2.0 or higher).

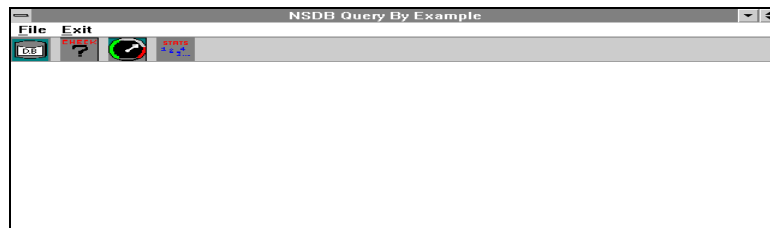
You start it by launching NSQBE.EXE.

---

## Components of NSQBE

### Main Window

After loading, which takes a few moments, the *NSDB Query By Example* window is displayed on the screen.



The workspace in the *NS-DB Query By Example* window consists of four parts:

- Title bar.
- Menu bar.
- Icon bar, situated under the menu bar.
- Work area.

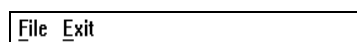
### Title Bar



This bar contains the following (from left to right):

- System menu icon.
- Product name.
- Minimize and maximize icons for the window.

### Menu Bar



The menu bar is situated under the title bar and provides access to all NSQBE's functions.

### ➤ File Menu



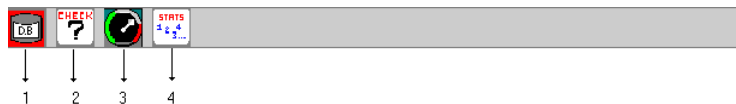
The File menu allows you to:

- Access the databases whose names have been defined in the initialization file.
- Display the activity monitor.

### ➤ Exit Menu

The Exit menu contains only one option, which closes the application.

## Icon Bar



This line is situated under the menu bar and contains four icons.

1. DB: opens the *Data Base List* dialog box, which lists all the databases defined in the initialization file. When a database is selected from the List box in the *Data Base List* dialog box, the CHECK icon is enabled.
2. CHECK: checks the physical space occupied by the database selected from the List box in the *Data Base List* dialog box.
3. MONITOR: opens the *Monitor Information* dialog box, which displays the activity monitor declared in the initialization file.
4. STATS: checks the logical validity of the database selected from the List box.

## Data Base List Dialog Box

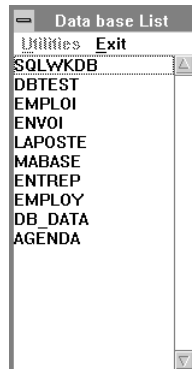
The *Data Base List* dialog box:

- Lists all the databases defined in the initialization file.
- Checks the physical space occupied by the database or starts a transaction on the database selected from the list.

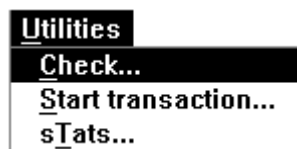
This dialog box is displayed by:

---

- ◆ Selecting the *Databases* option from the *File* menu.
- ◆ Clicking the DB icon in the main window.



## Utilities Menu



The *Utilities* menu is enabled if a database has been selected from the list. It allows you to:

- ◆ Check the physical space occupied by a database.
- ◆ Start a transaction on a database.
- ◆ Display the *DataBase stats Result* dialog box.

### ➤ Check

Checks the physical space occupied by a database.

Displays the result in the Database Check Result dialog box.

This box can also be displayed by double-clicking the CHECK icon in the main window.

This feature is equivalent to the DBCHCK utility.

### ➤ Start transaction

Opens the *DatabaseName Transaction N* dialog box, which allows you to start a transaction on the database selected from the list.

This dialog box can also be displayed by double-clicking the name of the database in the *Data Base List* dialog box.



The *Start transaction* function will either:

- Create the SYSTEM CATALOG for a database defined in the initialization file.
- Open an existing database.

### ➤ *sTats...*

Opens the *DataBase stats Result* dialog box, which lists the occurrences in each index for the selected database.

It also displays the pathname of the database selected from the List box and the pathname of each TABLESPACE in the database. These are preceded by  
\*\*SQLSTA+003 PROCEED WITH.

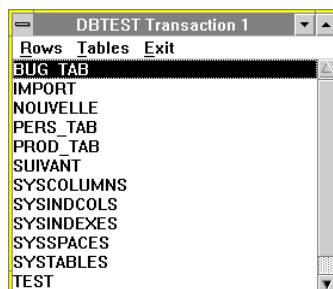
## Exit Menu

The *Exit* menu closes the *Data Base List* dialog box.

## DatabaseName Transaction N Dialog Box

The *DatabaseName Transaction N* dialog box is displayed by:

- ♦ Selecting the *Databases* option from the *File* menu.
- ♦ Double-clicking the name of the required database.



This dialog box allows you to perform the following operations:

- Add data to a table.
  - Execute queries.
  - Create tables.
  - Import tables.
  - Export tables.
-

## Menu Bar



This bar displays three menus that enable you to perform most of the operations that can be applied to database tables and data.

## Rows Menu



The *Rows* menu allows you to:

- Add one or more occurrences to a table.
- Commit the changes made to a database.
- Rollback the changes about to be made to a database.
- Execute queries on the data in the database.

### ➤ **New...**

Displays the *Store rows on DatabaseName.TableName* dialog box, which allows you to add occurrences to a client table.

### ➤ **Commit transaction**

Physically updates the database with any changes made since the transaction was started or since the last COMMIT or ROLLBACK.

This option is disabled while no updates have been made to the database.

### ➤ **Rollback transaction**

Cancels any updates made since the transaction was started or since the last COMMIT or ROLLBACK.

This option is disabled while no updates have been made to the database.

**➤ Search**

Opens the *DatabaseName.TableName* dialog box, which allows you to:

- Project columns.
- Select predicates.

**Tables Menu**

The *Tables* menu allows you to:

- Display information about the structure of a table.
- Add an index to a table.
- Generate data structures.
- Import tables.
- Export tables.
- Create new tables.

**➤ Info**

Opens the *DatabaseName.TableName* dialog box, which:

- Displays the characteristics of the table selected from the list.
- Allows you to add indexes and delete tables.

**➤ Dclgen**

Displays the *DCLRES* dialog box which generates declaratives for the selected table in NCL, C or Pascal.

**➤ Import**

Opens the *Import tables from IXF format* dialog box, which allows you to import tables exported in IBM IXF/PC format.

---

**➤ Export**

Opens the *Export TableName using IXF/PC format* dialog box, which enables you to export the selected table.

**➤ New**

Opens the *DatabaseName.NEWTABLE* dialog box, which enables you to create a new table in the database.

**➤ Reorg**

Retrieves the physical space freed after deleting a table.

**Exit Menu**

This menu generates an automatic COMMIT operation and closes the transaction.

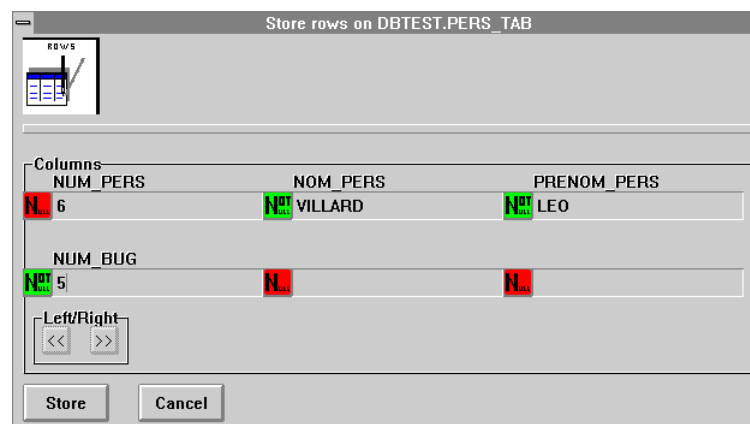
**Store Rows on DatabaseName.TableName Dialog Box**

The *Store Rows on DatabaseName.TableName* dialog box allows you to store new occurrences in the database.

The name of the database appears in the title bar.

This dialog box is displayed by:

- ◆ Selecting the *New...* option from the *Rows* menu in the *DatabaseName.Transaction N* dialog box.
- ◆ Double-clicking a result row in the *Selection Result Box*.



**Columns**

Displays the various columns in the table.

**Left/Right**

Allows you to access fields that are not visible.



Shifts the Columns group to the left so that you can access previous fields that are not visible.



Shifts the Columns group to the right so that you can access subsequent fields that are not visible.

**Store**

Updates the database with the information entered in the various fields.

Depending on the value in the primary key field, this will either generate the NS-DB statement DB\_INSERT or DB\_UPDATE.

**Cancel**

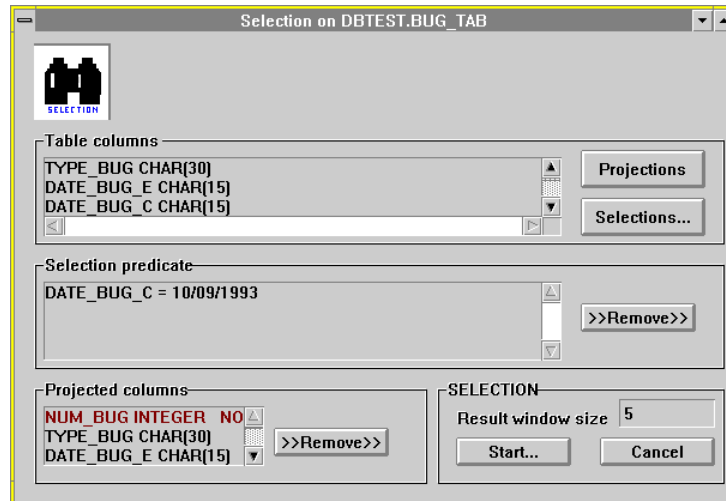
Closes the dialog box and ignores any changes made to the various fields.

***Selection on DatabaseName.TableName* Dialog Box**

The *Selection on DatabaseName.TableName* dialog box allows you to perform database search queries using column projections and selection predicates.

This dialog box is displayed by selecting the *Search...* option from the *Rows* menu in the *DatabaseName Transaction N* dialog box.

---



## Table columns

Lists the columns in the selected table (whose name is displayed in the title bar).

Indexes are differentiated by three colors:

- Red: the column is part of the PRIMARY KEY.
- Green: the column is part of the FOREIGN KEY.
- Blue: the column is part of an access key.

## Projections

Projects the columns selected from the *Table columns* List box.

Displays projected columns in the *Projected columns* List box.

Projected columns are a sub-set of the table's columns.

## Selections...

Displays the *Select Predicate* dialog box, which allows you to select a predicate for the column selected from the table.

## Selection predicate

Lists selected predicates.

## Remove

Removes the selected predicate from the *Predicate Selection* List box.

## Projected columns

Lists projected columns

### ➤ **Remove**

Deletes the projected column from the *Projected Columns* List box.

## Selection

### ➤ **Result window size**

Defines the layout of the query result. If 'n' is entered in this field, the result will be displayed in groups of n rows.

### ➤ **Start**

Executes the query and displays the result in the *Selection result box*.

### ➤ **Cancel**

Closes the dialog box and ignores any changes made.

## Selection Result Dialog Box

The *Selection Result Box* displays the results of the query defined in the *Selection on DatabaseName.TableName* dialog box.

---

Selection result box

Selection information

Used Index

BUG

(Times are in milliseconds and do not include graphical overhead)

Max	Min	Average	Total time	SearchTime
31	0	12	125	63

Selected list

NUM BUG	TYPE BUG	DATE
1	Question	15/02/92
2	Anomalie	25/03/93
3	Question	28/07/93

\*\* END OF SELECTION LIST \*\*

Next Delete current Cancel

## Selection Information

### ➤ Used Index

Displays the name of the index used by the NS-DB optimizer to execute the requested query.

### ➤ Max

Displays the maximum time taken (in milliseconds) for NS-DB to read one of the rows displayed in the List box.

### ➤ Min

Displays the minimum time taken (in milliseconds) for NS-DB to read one of the rows displayed in the List box.

### ➤ Average

Displays the average time taken (in milliseconds) to display the occurrences in the List box.

### ➤ Total time

Displays the total time taken (in milliseconds) to display the occurrences in the List box.



**➤ Search Time**

Displays the actual time taken (in milliseconds) to execute the initial search query (DB\_SEARCH).

**Selected list**

Displays, in column form, the result of the query entered in the *Selection on DatabaseName.TableName* box. Double-clicking a result row opens the *Store Rows on DatabaseName.TableName* dialog box.

**➤ Next**

Displays the next group of result rows that are not visible.

**➤ Next**

Deletes the row chosen from the *Selected List*.

**➤ Cancel**

Closes the dialog box and ignores any changes made.

***DatabaseName.TableName* Dialog Box**

The *DatabaseName.TableName* dialog box allows you to:

- Display the characteristics of a table.
- Add a new column.
- Add a new index.

This dialog box is displayed by:

- ♦ Selecting the *Info...* option from the *Tables...* menu in the *DatabaseName Transaction N* dialog box.
- ♦ Double-clicking a table name in the *DatabaseName Transaction N* dialog box.

## Table Name

Displays the name of the selected table.

## Data storage

DATA SPACE (NS-DB V.02)

Displays the name and state of each DATASPACE.

A DATASPACE is a two-state object: FREE or BUSY.

## Column Info

### ➤ Column Name

Displays the name of the new column to add to the table.

### ➤ Data Type

Selects the column type (CHAR, INTEGER, STRING, etc.).

### ➤ Size

Displays the size of the data type: CHAR, VARCHAR, VARGRAPHIC or GRAPHIC.

**➤ Null value**

Sets a constraint on the column value.

If this box is unchecked, the value of the column will be NOT NULL and must contain a value.

**➤ Add column**

Adds the new column to the *Column List*.

**Indexes**

Lists the various indexes defined for the table.

Double-clicking an index opens the *TableName.IndexName* dialog box, which allows you to display information about the selected index.

**➤ Add index...**

Displays the *New index definition on TableName* dialog box, used to add an index to the table.

**Column List**

Lists the various columns in the table.



Removes the selected column from the *Column List*.

**Create**

This button is disabled when the *DatabaseName.TableName* dialog box has been opened to display information about a table.

**Alter**

Creates a new column in the table *TableName*.

**Drop**

Deletes the table.

---



the table will be permanently deleted.

## Cancel

Closes the dialog box and ignores any changes made.

## TableName.IndexName Dialog Box

The *TableName.IndexName* dialog box:

- Displays information about the selected index.
- Allows you to delete the selected index.

The title bar in this window displays the corresponding table and index name.

This dialog box is displayed by double-clicking one of the indexes in the *Indexes* list in the *DatabaseName.TableName* dialog box.

## Index Name

Displays the name of the selected index.

## Index Info

### ➤ **Table Space**

Displays the name and state of the TABLESPACE used to store the new index.

A TABLESPACE is a three-state object: FREE, CLUSTERED or UNCLUSTERED.

### ➤ **Primary Key**

Check this box if the selected index is the PRIMARY KEY.

### ➤ **Unique**

Check this box if the index is UNIQUE.

## Column List / Indexed Columns

This group contains two List boxes:

- *Column List* displays the various columns in the table.
- *Indexed Columns* displays the result of the indexed column.

### ➤ **Index Column**

#### 1. Cluster1

Displays the first CLUSTERING NUMBER.

#### 2. Cluster2

Displays the second CLUSTERING NUMBER.

This group of controls is disabled when the *TableName.IndexName* dialog box is used to consult index information.

#### 3. Descending

Sorts the keys of an index column in ascending or descending order.

If this box is checked, the sort order will be descending.

#### 4. Displays the description of the index column in the *Indexed Columns* List box.

This button is disabled when the *TableName.IndexName* dialog box is used to consult index information.

#### 5. Deletes the index selected from the *Indexed Columns* List box.

This button is disabled when the *TableName.IndexName* dialog box is used to consult index information.

► **Indexed Columns**

Displays the name of the indexed column together with its associated CLUSTERING NUMBERS.

## Foreign Key

This group of controls is disabled when the *TableName.IndexName* dialog box is used to display index information.

## Create

This button is disabled when the *TableName.IndexName* dialog box is used to consult index information.

## Drop

Permanently deletes the index.

This button is disabled if you have selected a primary key index that cannot be deleted.

## Cancel

Closes the dialog box and ignores any changes made.

## ***New Index definition on TableName Dialog Box***

The *New index definition on TableName* dialog box is displayed by clicking the *Add Index...* button in the *DatabaseName.TableName* dialog box.

This dialog box displays index information and is similar to the *TableName.IndexName* dialog box. It is used here to input a new index.

The screenshot shows a dialog box titled "New index definition on BUG\_TAB". It contains the following fields and controls:

- Index Name:** A text field containing "INEWDATEC".
- Table Space:** A dropdown menu showing "SYSTEM CLUSTERED".
- Primary Key:** An unchecked checkbox.
- Unique:** An unchecked checkbox.
- Column List / Indexed Columns:** A list box containing three entries: "TYPE\_BUG CHAR(30)", "DATE\_BUG\_E CHAR(15)", and "DATE\_BUG\_C CHAR(15)".
- Index Column:** A section with two input fields for "Cluster 1" and "Cluster 2", both containing the value "8". There is a "GET CLUSTER" button and navigation arrows (>>, <<).
- Descending:** An unchecked checkbox.
- Foreign Key:** A section with a "References" dropdown menu.
- Delete Rule:** A section with three radio buttons: "Restrict" (selected), "Cascade", and "Set NULL".
- Update Rule:** A section with one radio button: "Restrict" (selected).
- Buttons:** "Create", "Drop", and "Cancel" buttons are located at the bottom left.

## Index Name

Displays the name of the index to create.

## Index Info

### ➤ Table Space

Allows you to select the TABLESPACE used to store the new index. A new TABLESPACE can be specified in this field; it will be created when you click the *Create* button in the *TBSPACE* dialog box.

### ➤ Primary Key

Sets a constraint specifying that the new index is the PRIMARY KEY INDEX and UNIQUE.

Disabled if the PRIMARY KEY column already exists.

### ➤ Unique

Sets a constraint specifying that the value associated with the column must be UNIQUE.

If this box is checked, the index will be UNIQUE.

---

## Column List / Indexed Columns

This group contains two List boxes:

- *Column List* displays the various columns in the table.
- *Indexed Columns* displays the result of the indexed column.

### ➤ Index Column

#### 1. Cluster1

Displays the first CLUSTERING NUMBER.

#### 2. Cluster2

Displays the second CLUSTERING NUMBER.



3. Automatically displays the first unused CLUSTERING NUMBER in the database.

The purpose of CLUSTERING NUMBERS is described in the section on CREATE INDEX INDEX COLUMN in Chapter 4.

#### 4. Descending

Sorts the keys of an index column in ascending or descending order.

If this box is checked, the sort order will be descending.



5. Displays the description of the index column in the *Indexed Columns* List box.



6. Deletes the index selected from the *Indexed Columns* List box.

## Indexed Columns

Lists the columns that form the index.

## Foreign Key

### ➤ References

Specifies the table associated with the FOREIGN KEY.

### ➤ Delete Rule

#### 1. Restrict



Used to enforce referential integrity.

An instance cannot be deleted from a table if it has a dependent table.

## 2. Cascade

Used to enforce referential integrity.

If an instance is deleted from a table, all the related instances in the dependent table will be automatically deleted.

## 3. Set NULL

Used to enforce referential integrity.

The table's FOREIGN KEY may be set to NULL.

### ➤ Update rule

#### Restrict

Used to enforce referential integrity.

The FOREIGN KEY cannot be modified.

Only *Restrict Rules* are functionally implemented in the current version of NS-DB.

### Create

Creates the new index.

If a new TABLESPACE has been specified in the TABLESPACE field, the *TBSPACE* dialog box will be displayed.

### Drop

Permanently deletes the index.

### Cancel

Closes the dialog box.

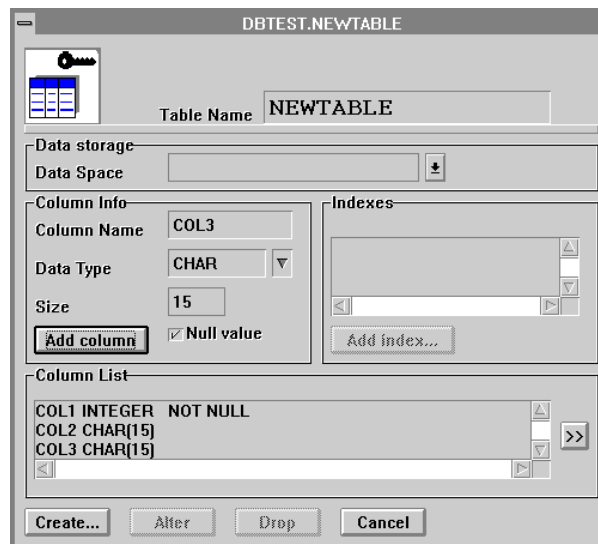
## ***DatabaseName.NewTable* Dialog Box**

The *DatabaseName.Newtable* dialog box allows you to create a new table together with its indexes.

The title bar in this dialog box displays the name of the database and the new table.

---

This dialog box is displayed by selecting the *New...* option from the *Tables* menu in the *Database\_Name Transaction N* dialog box.



### Table Name

Displays the name of the new table to create.

### Data storage

#### ➤ Data Space (NS-DB V.02)

Displays the name and state of each DATASPACE.

A DATASPACE is a two-state object: FREE or BUSY.

### Column Info

#### ➤ Column Name

Displays the name of the column to create in the table.

#### ➤ Data Type

Allows you to select or enter the column type.

**➤ Size**

Displays the size of the data type: CHAR, VARCHAR, VARGRAPHIC or GRAPHIC.

**➤ Null value**

Sets a constraint on the column value.

If this box is unchecked, the value of the column will be NOT NULL and must be input.

**➤ Add column**

Adds a new column to the *Column List*.

**Indexes**

Lists the various indexes defined in the table.

Double-clicking an index opens the *TableName.IndexName* dialog box, which displays information about the selected index.

**➤ Add index**

Opens the *New index definition on TableName* dialog box, which allows you to add an index to the table.

**Column List**

Lists the various columns in the table.

Removes the selected column from the *Column List*.

**Create**

Displays the *New index definition on TableName* dialog box, which allows you to create the table and the PRIMARY KEY INDEX.

(NS-DB V.02) If a new DATASPACE has been specified in the DATASPACE field, the *TBSPACE* dialog box will be displayed on the screen. This will be followed by the *New index definition on TableName* dialog box.

---

### **Alter**

This button is disabled while the *DatabaseName.NewTable* dialog box is used to create a table.

### **Drop**

This button is disabled while the *DatabaseName.NewTable* dialog box is used to create a table.

### **Cancel**

Closes the dialog box and ignores any changes made.

## **DCLRES Dialog Box**

The *DCLRES* dialog box is displayed by selecting the *Dclgen...* option from the *Tables* menu in the *DatabaseName Transaction N* dialog box.

The contents of this dialog box vary according to the choice displayed by the *Dclgen* option.

### **NCL Language**

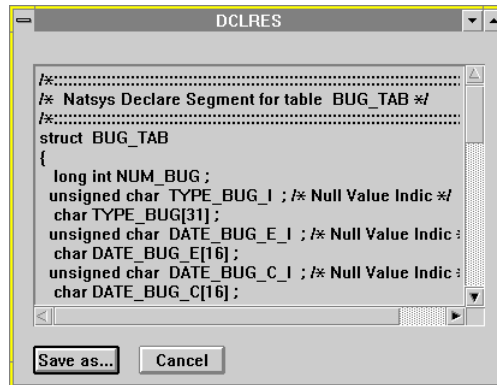
Generates an NCL data segment based on the table selected from the List box in the *DatabaseName Transaction N* dialog box.

### **C Language**

Generates an C data structure based on the table selected from the List box in the *DatabaseName Transaction N* dialog box.

### **Pascal Language**

Generates a Pascal record based on the table selected from the List box in the *DatabaseName Transaction N* dialog box.



### Save as

Displays the *Save SQL Language Declaratives* dialog box, which saves the list of declaratives to a source file. This file can be reused later by an NCL, C or Pascal application.

### Cancel

Closes the dialog box and ignores any changes made.

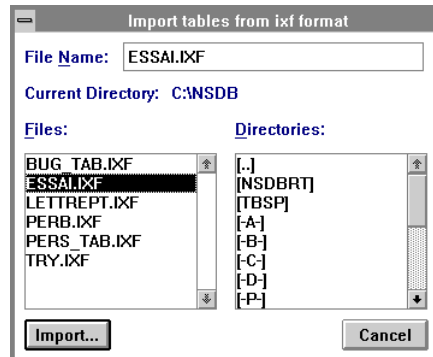
## Import Tables from IXF Format Dialog Box

The *Import Tables from IXF format* dialog box allows you to select a file containing data in IXF format and import it into an NS-DB database table.

This dialog box is displayed by selecting the *Import...* option from the *Tables* menu in the *DatabaseName Transaction N* dialog box.



A file in IXF/PC format can be produced using a DB2, DBM or NS-DB EXPORT utility.



### File Name

Displays the name of the IXF file selected from the *Files* List box.

### Current Directory

Displays the current directory name.

### Files

Lists all existing IXF files in the current directory.

### Directories

Allows you to select a new path.

### Import

Displays the *Import from Drive:\Path\Fname.IXF* dialog box, which allows you to import a table.

### Cancel

Closes the dialog box and ignores any changes made.

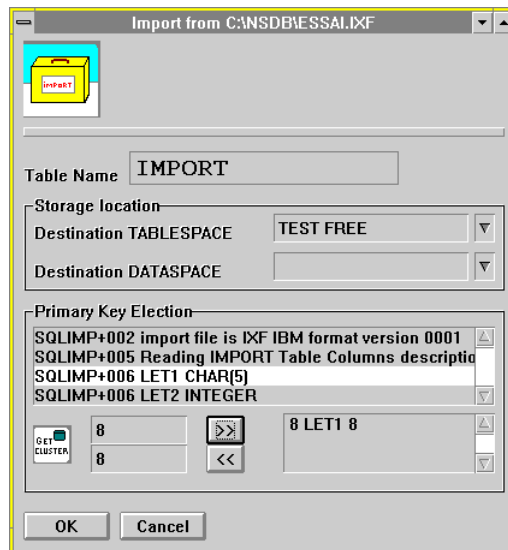
## ***Import from Drive:\Path\Fname.IXF* Dialog Box**

The *Import from Drive:\Path\Fname.IXF* dialog box allows you to name the table that will receive the data imported from the .IXF file.

This dialog box is displayed by clicking the *Import* button in the *Import Tables from IXF format* dialog box.



The CREATE TABLE command will be executed by the import module using the data descriptions in a .IXF file.



## Table Name

Displays the name of the table that will be imported into the current database.

## Storage location

### ➤ **Destination TABLESPACE**

Selects the destination TABLESPACE.

### ➤ **Destination DATASPACE (NS-DB V.02)**




Selects the destination DATASPACE.

## Primary Key Election

Lists:

1. Information about the import.
2. The columns in the table.

Only lines beginning with SQLIMP+006 can be accessed. They can be used to build the PRIMARY KEY INDEX required to create an NS-DB table.

3.  Automatically displays the first unused CLUSTERING NUMBER in the database.
4.  Displays the description of the index column in the List box.
5.  Deletes the index selected from the List box.

### OK

Imports the table into the database.

If the destination table already exists, a message will prompt you to confirm the replacement.

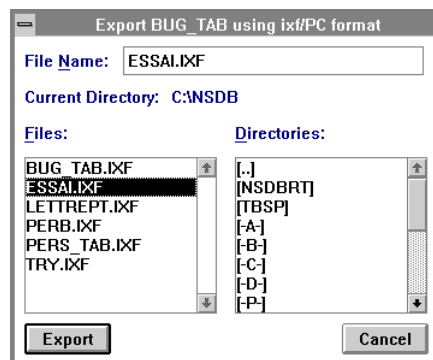
### Cancel

Closes the dialog box and ignores any changes made.

## Export Table Using IXF/PC Format Dialog Box

The *Export TableName using IXF/PC format* dialog box is displayed by selecting the *Export...* option from the *Tables* menu in the *DatabaseName Transaction N* dialog box.

To enable the *Export...* option, you need to select the table that you want to export from the *DatabaseName Transaction N* dialog box.





**File Name**

Automatically displays the name of the .IXF file to create.

This filename defaults to the name of the exported table

**Current Directory**

Displays the current directory name.

**Files**

Lists all existing IXF files in the current directory.

**Directories**

Allows you to select the directory used to store the .IXF file.

**Export**

Exports the .IXF file to the selected directory.

**Cancel**

Closes the dialog box and ignores any changes made.

***Monitor Information Dialog Box***

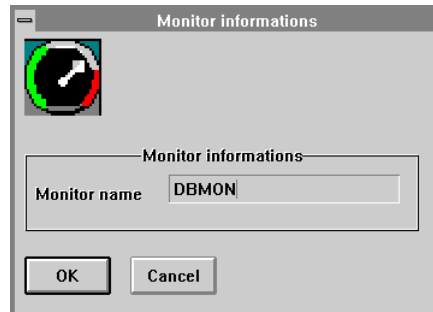
The *Monitor Information* dialog box is displayed by selecting the *Monitor* icon in the main window.



The monitor is activated in the initialization file as follows:

```
[System]
DBDIR='C:\NS-DB',TRACE='NO',MONITOR='DBMON';
SYSID='NAME';
JOURNAL=YES;
```

---



## Monitor information

### ➤ **Monitor name**

Displays the name of the monitor defined in the initialization file.

### **OK**

Displays the *Monitor Activity* dialog box, which displays monitor information.

### **Cancel**

Closes the dialog box and ignores any changes made.

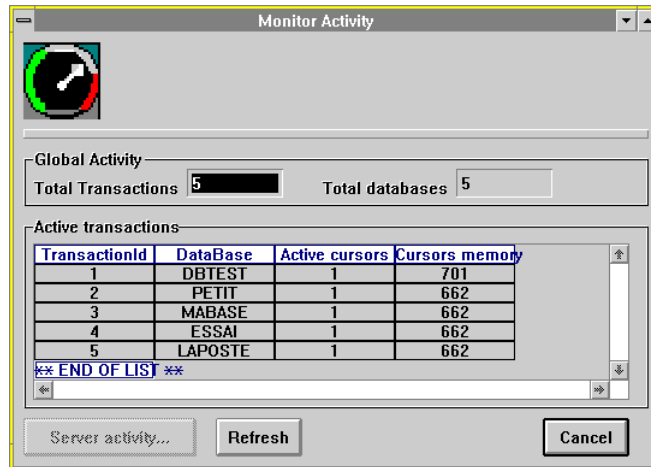
## **Monitor Activity Dialog Box**

The *Monitor Activity* dialog box displays information about the monitor's global activity and the number of transactions.

This dialog box is displayed by clicking *OK* in the *Monitor Information* dialog box.

The activity monitor allows you to:

- Analyze the current database's resources.
- Display access times for NS-DB applications running on a given machine.



## Global Activity

### ➤ **Total transactions**

Displays the total number of transactions started.

### ➤ **Total Databases**

Displays the total number of databases opened.

## Active transactions

Lists all active transactions.

Double-clicking a line in the list opens the *Transaction N on DatabaseName* dialog box, which displays the number of statements executed together with the time taken.

### ➤ **TransactionId**

Transaction ID, ranging from 1 - N.

### ➤ **DataBase**

Displays the name of the database on which transaction N was started.

### ➤ **Active cursors**

Displays the number of active cursors.

**➤ Cursors memory**

Displays the amount of memory used by a cursor.

**Server activity**

This button is enabled if a server has been started (by the DBSERV utility).

It opens the *Server Activity* dialog box, which displays information about the server.

**Refresh**

Refreshes the list of active transactions and includes any transactions that were started while the monitor was displayed (only possible in client/server mode).

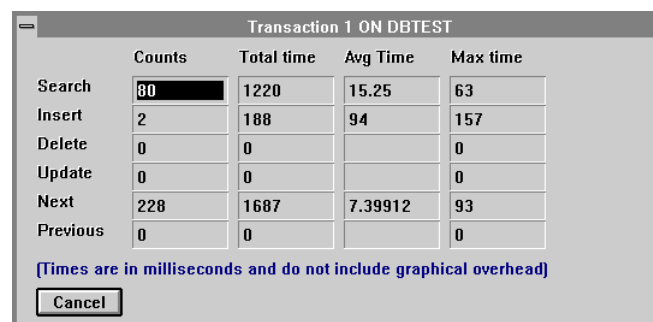
**Cancel**

Closes the dialog box.

***Transaction N on DatabaseName* Dialog Box**

The *Transaction N on DatabaseName* dialog box displays the number of statements executed together with their duration.

This dialog box is displayed by double-clicking an active transaction in the *Active transactions* group in the *Monitor Activity* dialog box.



	Counts	Total time	Avg Time	Max time
Search	80	1220	15.25	63
Insert	2	188	94	157
Delete	0	0		0
Update	0	0		0
Next	228	1687	7.39912	93
Previous	0	0		0

[Times are in milliseconds and do not include graphical overhead]

Cancel

Information about the transaction is displayed in a table showing:

- Horizontally: statistics relating to the transaction (number of statements executed by the transaction and their duration).

- Vertically: a list of NS-DB statements - SEARCH, INSERT, DELETE, UPDATE, NEXT, PREVIOUS.

**Counts**

For each NS-DB statement, displays the number of calls executed during the transaction.

**Total time**

Displays the total execution time (in milliseconds) for each NS-DB statement listed vertically.

**Avg time**

Displays the average execution time ([Total time / Counts] milliseconds) for each NS-DB statement listed vertically.

**Max time**

Displays the maximum execution time (in milliseconds) for each NS-DB statement.

**Cancel**

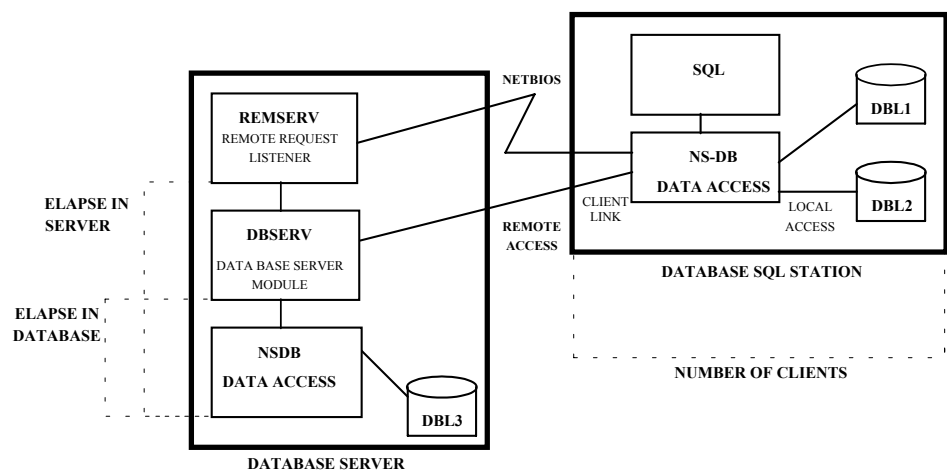
Closes the dialog box.

**Server Activity Dialog Box**

The *Server Activity* dialog box is displayed by clicking the *Server Activity...* button in the *Monitor Activity* dialog box. It displays information about the NS-DB server.

---

Server activity	
Total number of call served	0
Total calls to database	0
Total Elapse in database	0
Max Elapse in database	0
Avg Elapse in database	0
Total Elapse in server	0
Max Elapse in server	0
Avg Elapse in server	0
Current number of clients	0
Max number of clients in session	0
(Times are in milliseconds and do not include graphic)	
<input type="button" value="Refresh"/> <input type="button" value="Cancel"/>	



General NS-DBR schema, in dotted boxes: Elapse In Server, Elapse In Database, Number Of Clients.

### Total calls to server

Displays the total number of server access operations.

### Total calls to database

Displays the total number of database access operations.

**Total Elapse in database**

Displays the total time taken in the NS-DB engine (this total includes the time taken to access the server module (DBSERV)).

**Max Elapse in database**

Displays the maximum time taken in the NS-DB engine (this maximum includes the time taken to access the server module).

**Avg Elapse in database**

Displays the average time taken in the NS-DB engine (this average includes the time taken to access the server module).

**Total Elapse in server**

Displays the total time taken in the NS-DB engine and server (this total includes the time taken to access the listener module (REMSERV)).

**Max Elapse in server**

Displays the maximum time taken in the NS-DB engine and server (this total includes the time taken to access the listener module (REMSERV)).

**Avg Elapse in server**

Displays the average time taken in the NS-DB engine and server (this total includes the time taken to access the listener module (REMSERV)).

**Current number of clients**

Displays the number of clients connected to the server.

**Max number of clients in session**

Displays the maximum number of clients connected to the server in the session.

---

## Using NSQBE

### Opening or Creating a Database

The *Databases List* dialog box lists the databases defined in the initialization file.

This dialog box is displayed by:

- ◆ Selecting the *Databases...* option from the *File* menu in the main window.
- ◆ Clicking the DB icon in the main window.

A database is created or opened by:

- ◆ Double-clicking or entering the name of an existing database.
- ◆ Selecting the *Start transaction...* option from the *Utilities* menu.

If the database had to be created, the *DatabaseName Transaction N* window will display the SYSTEM CATALOG tables that have just been created. The new tables will be created after selecting the *New...* option from the *Tables* menu.

If the database already existed, all the tables, including the system tables, appear in the List box displayed by the *DatabaseName Transaction N* dialog box.

Several simultaneous transactions can be started on the same database or on different databases. In this case, the transactions will be automatically numbered from 1 to N in the title bar of the *DatabaseName Transaction N* dialog boxes. (For convenience, each window can be minimized to avoid cluttering up the screen).

The *DatabaseName Transaction N* window is the main part of the utility since its menus contain all the NSQBE functions.

### Checking a Database for Physical Errors

You can check the physical space occupied by a database by:

- ◆ Choosing the *Check...* option from the *Utilities* menu.
- ◆ Clicking the CHECK icon in the main window.

The database must previously have been selected from the *Database List* dialog box. While the check runs, the message "Please wait... Database Check in Progress..." is displayed on the screen. This is followed by the *Database Check Result* window, which displays the result of the check.



See the description of the DBCHCK utility in Chapter 9 for details on the error messages returned in the *Database Check Result* dialog box.

## Checking a Database for Logical Errors

You can check a database for logical errors by:

- ♦ Choosing the *sTats...* option from the *Utilities* menu.
- ♦ Clicking the STATS icon in the main window.

The database must previously have been selected from the *Database List* dialog box. While the check runs, the message "Please wait... Database Statistics in Progress..." is displayed on the screen. This is followed by the *Database Stats Result* window, which displays the result of the check.

The list of occurrences in each index is displayed as follows:

```
Table : TABLENAME  
PRIMARY KEY INDNAME1 Occurrences : 6 on TBSPACE1  
INDEX INDNAME2 Occurrences : 6 on TBSPACE2  
:  
:
```

If, for a given table, the number of occurrences of each index is not identical, this means that the database is **LOGICALLY disorganized**. You will then need to DROP any secondary indexes and rebuild them (the primary key cannot be deleted).

The message:

```
SQLSTA-003 INDEX SYNCHRONIZATION NEEDED ON TABLENAME
```

is displayed for each disorganized table.

To delete an index:

- Select the *Info* option from the *Table* menu in the *Start transaction* dialog box.
- Double-click the secondary index required.
- Click the *Drop* button in the *TableName.IndexName* dialog box.

To rebuild the index, click the *Add Index* button in the *DatabaseName.TableName* dialog box displayed by selecting the *Info* option from the *Tables* menu.

## Starting a Transaction

To start a transaction:

- ♦ Double-click the database name displayed in the *DataBase List* dialog box.

- ◆ Select the *Start Transaction...* option from the *Utilities* menu.

Several simultaneous transactions can be started on the same database or on different databases.

The fact that the transaction has been started is shown in the *DatabaseName Transaction N* dialog box.

This dialog box displays the various tables in the corresponding database.

It contains the *Rows* and *Tables* menus, which display the various functions that can be requested during the transaction:

- Adding a row of data to a table.
- COMMIT and ROLLBACK
- And so on.

## Adding a Row to a Table

To add a new row of data to a table, select the *New...* option from the *Rows* menu in the *Store Rows on DatabaseName.TableName* dialog box.

Only client tables can be modified (as opposed to SYSTEM CATALOG tables).

Each of the table columns in this window are preceded by an icon that indicates the value allowed for the column.

Clicking the *Store* button stores the data in the database. The transaction will physically write the data after you select the *Commit* option from the *Rows* menu.



Several instances can be input before closing the dialog box.

## Commit and Rollback

The COMMIT function physically updates the database with any changes made since the last COMMIT or ROLLBACK, while the ROLLBACK function cancels these changes.

The COMMIT function is triggered by selecting the *Commit* option from the *Rows* menu.

The ROLLBACK function is triggered by selecting the *Rollback* option from the *Rows* menu.

To enable these functions, transactional journalling must be activated (JOURNAL=YES in the initialization file).

Example of transactional journalling in the initialization file :

```
[System]
DBDIR='C:\NS-DB';
SYSID='NAME';
JOURNAL='YES';
```

These two commands are disabled while no updates have been made. As soon as an update is made in the transaction, both options are automatically enabled. The selection of one of them triggers the associated action and they are both disabled again.

## Executing a Query

The *Selection on DatabaseName.TableName* dialog box is displayed by selecting the *Search...* option from the Rows menu. It allows you to execute queries on the selected table.

Two operations are available:

- Projecting table columns.
- Selecting search predicates.



These two operations can be combined.

## Projecting Columns

The *Selection DatabaseName.TableName* dialog box allows you to project columns and define a sub-set of the selected table.

To project one or more columns, simply select each one from the *Table Columns* List box and click the *Projections* button. The projected columns appear in the *Projected Columns* List box.



The maximum number of columns that can be projected depends on the total width of the columns. Since a List box cannot contain more than 255 characters, the total width of the projected columns cannot exceed this number. If this width is exceeded the message: "Selection Error Buffer overflow => Too many projected columns" will be displayed when the <Start...> button is pressed. You will then need to reduce the number of columns included in the operation and try again.

---

You can use the *Result Window Size* field to set the number of rows displayed in the query result. If you enter N in this field, the result will be displayed in groups of N rows.

The result is calculated when you click the *Start...* button, which displays the *Selection Result Box* dialog box.

The *Selected List* in this window displays the query's result for all selected columns.

The name of the index used by the optimizer is displayed in the *Used Index* field. The various response times displayed are explained in the section *Selection Result Box*.

If you have configured the result to display a group of rows, use the *Next* button to scroll through each group of rows.

To delete an instance, select the required row from the *Selected List* and click the *Delete current* button.



When you stop the transaction, deleted rows are permanently removed as a result of an implicit COMMIT statement. To cancel the deletion, select the ROLLBACK function before stopping the transaction.

## Selecting a Search Predicate

The *Selection DatabaseName.TableName* dialog box allows you to restrict the query by applying a restrictive predicate to one or more columns.

Select the columns that you want to project from the *Projected Columns* List box. Regardless of the number of projected columns, you can select a predicate by:

- Choosing a column from the *Table columns* List box.
- Clicking the *Selections...* button.

The *Select Predicate* dialog box displays the name of the selected column and allows you to choose from the following operators: =, >, <, >=, <=, <>, ==.

The restriction will take effect once you have:

- Closed the *Select Predicate* dialog box.
- Clicked OK.

The predicate will be displayed in the *Selection predicate* List box in the *Selection on DatabaseName.TableName* dialog box.

You can repeat this operation to define several predicates for different columns (projected or otherwise).

The result of the query will be displayed in the *Selection Result Box*.

## Creating a Table

To create a table in the current database, select the *New...* option from the *Tables* menu in the *DatabaseName.TableName* dialog box.

Creating a table consists of two steps:

1. Defining the various columns in the table.
2. Defining the PRIMARY KEY INDEX required to create an NS-DB table.

The table name must be specified in the *Table Name* field of the *DatabaseName.TableName* dialog box.

In NS-DB Version V.02, a DATASPACE can be selected from the DATA STORAGE group. If you want to define a new DATASPACE, you simply enter its name in the DATASPACE field. After clicking the *Create...* button, the *TBSPACE* dialog box will be displayed, allowing you to specify the location of the DATASPACE.

The columns are defined using the *Column Name*, *DataType* and *Size fields*. The *Null value* Check box sets the value for the whole column. If it is unchecked, the column will have the value NOT NULL.

Once a column has been defined, its details will be displayed in the *Column List* when you click the *Add column* button.

The *Create...* button will then be enabled.

All the columns created for the table will appear in this List box. If a column definition is incorrect, you can delete the column by selecting it and clicking the >> button.

Once all the columns in the table have been defined, click the *Create...* button to successively display:

- The *TBSPACE* dialog box (if a new DATASPACE has been selected).
- The *New index definition on TableName* dialog box, which allows you to create the PRIMARY KEY INDEX.

The *Primary Key* Check box is automatically checked and cannot be modified.

---



When a table is created in NS-DB, its PRIMARY KEY INDEX must also be created.

You must enter the name of the PRIMARY KEY in the *Index Name* field.

Specify the characteristics of the columns that make up the primary index in the *Column List/Indexed Columns* group (*Column List* is the first List box and *Indexed Columns* is the second List box).

Select a column from the first List box, *Column List*, which displays all the columns in the table.

Once the column has been selected, you must choose the CLUSTERING NUMBERS associated with it. Click the *Get Cluster* button, which automatically displays the last CLUSTERING NUMBERS that have yet been used by NS-DB. You can also choose your own CLUSTERING NUMBERS by entering them directly in the *Cluster1* and *Cluster2* fields.

If you check the *Descending* Check box, the column will be sorted in descending order.

Next click the >> button to display the indexed column, together with its CLUSTERING NUMBERS, into the *Indexed Columns* List box.

You can repeat this operation to index more columns and build a complex primary key.

The *Foreign Key* group allows you to provide more information about the indexed column. If the index is a FOREIGN KEY, you can specify:

- The table it is associated with.
- The clauses used to enforce referential integrity.

Click the *Create* button to create the table and the PRIMARY KEY index simultaneously. This will close the following dialog boxes:

- *New index definition on TableName.*
- *DatabaseName.TableName.*

The name of the new table will be added immediately to the list of tables in the *DatabaseName Transaction N* dialog box.

## Creating a DATASPACE (NS-DB V.02)

You can create a DataSpace when you create a new table by entering a name in the DATASPACE field in the *Database.Newtable* dialog box.

The location of the DATASPACE must be specified in the *TBSPACE* dialog box, which is displayed by clicking the *Create...* button. The *Operating system file name and location* field must contain the full pathname of the DATASPACE:  
`Drive:\Path\Fname.`

The DATASPACE will be created when you click the *Create* button in the *TBSPACE* dialog box.

## Creating a TABLESPACE

You can create a TABLESPACE when you create a new index.

Select the *Info* option from the *Tables* menu to display the *DatabaseName.TableName* dialog box. Then, click the *Add Index...* button to display the *New Index definition on TableName* dialog box.

To create the TABLESPACE, enter a name in the TABLESPACE field.

You must specify the location of the TABLESPACE in the *TBSPACE* dialog box, which is displayed by clicking the *Create* button.

The *Champ Operating system File Name and location* field must contain the full pathname of the TABLESPACE: `Drive:\Path\Fname.`

The TABLESPACE will be created when you click the *Create* button in the *TBSPACE* dialog box.

## Dropping a Table

The *DatabaseName.TableName* dialog box allows you to drop a table.

This dialog box is displayed by:

- ◆ Selecting the *Info...* option from the *Tables* menu.
- ◆ Double-clicking the name of a table in the *DatabaseName Transaction N* dialog box.

The name of the table to be dropped is displayed in the title bar of the *DatabaseName.TableName* window.

---

Click the *Drop* button to drop the selected table. This will display a message on the screen prompting you to confirm the deletion: "TableName Do you really want to drop this table ?". Click *Yes* to permanently delete the table.



The Drop operation is irreversible. The storage area formerly occupied by the table is retrieved immediately by NS-DB (REORG DATABASE command).

The table drop operation is accompanied by the message "TableName Drop in progress". The dialog box is then closed.

The name of the dropped table will no longer appear in the list of tables displayed by the *DatabaseName Transaction N* window.

## Adding a New Column to a Table

The *DatabaseName.TableName* dialog box allows you to add a new column to the selected table.

This dialog box is displayed by:

- ◆ Selecting the *Info...* option from the *Tables* menu.
- ◆ Double-clicking the name of a table displayed in the *DatabaseName Transaction N* dialog box.

The name of the table will be displayed in the title bar of the *DatabaseName.TableName* window.

Define the new column using the fields in the *Column Info* group, which specify:

- The column name.
- Its type.
- Its size.
- Its value.

If the *Null value* Check box is checked, the value of the column will be NULL, otherwise it will be NOT NULL.

Once the various fields have been defined, click the *Add column* button to display the definition of the new column in the *Column List*. The *Alter* Push button will then be enabled.

If the column definition is invalid, you can delete it by selecting it from the List box and clicking the >> button.



If the column definition is correct, it will be added when you click the *Alter* button (which generates an ALTER TABLE command).

After displaying the message "TableName Add column..." on the screen, the dialog box will be closed.

## Adding an Index

The *DatabaseName.TableName* dialog box allows you to add a new index to the selected table.

This dialog box is displayed by selecting the *Info...* option from the *Tables* menu.

Click the *Add index...* button to display the *New index definition on TableName* dialog box, which allows you to define a new index.

Enter the index name in the *Index Name* field. Since the PRIMARY KEY index was created earlier when the table was created, the *Primary key* Check box is disabled.

To ensure that the values in the column are all unique, you can assign the UNIQUE keyword to the index by checking the *Unique* Check box.

Select the characteristics of the index from the *Column List/Indexed Columns* group (*Column List* is the first List box and *Indexed List* is the second List box).

Select a column from the first List box, *Column List*, which displays all the columns in the table.

Once the column has been selected, you must choose the CLUSTERING NUMBERS associated with it. Click the *Get Cluster* button, which automatically displays the last clustering numbers that have yet been used by NS-DB. You can also choose your own clustering numbers by entering them directly in the *Cluster1* and *Cluster2* fields.

If you check the *Descending* Check box, the column will be sorted in descending order.

Next, click the >> button to insert the indexed column, together with its CLUSTERING NUMBERS, into the *Indexed Columns* List box .

You can repeat this operation to index more columns and build a complex primary key.

The *Foreign Key* group allows you to provide more information about the indexed column. If the index is a FOREIGN KEY, you can specify:

- The table it is associated with.

- The clauses used to enforce referential integrity.

Click the *Create* button to create the index and close the dialog box.

The name of the new index will appear in the *Indexes* List box displayed by the *DatabaseName TableName* window.

The PATH INDEX keyword, displayed in the List box, is used to differentiate the displayed index from the PRIMARY KEY INDEX or FOREIGN KEY INDEX.

## Generating Data Structures in NCL, C and Pascal

The *Dclgen...* option on the *Tables* menu displays the *DatabaseName.TableName* dialog box, which allows you to generate data structures for the NCL, C and Pascal languages.

To generate a data structure, use the *DatabaseName Transaction N* List box to select the table(s) from which you want to extract an NCL, C or Pascal structure.

Select NCL, C or Pascal from the *Dclgen* option to generate the corresponding data structure in the selected language.

The result of the generation is displayed in the *DCLRES* dialog box.

This result can be saved to a file with an extension of your choice so that it can be included in an application written in the generated language.

Click the *Save as...* button to display the *Save SQL Language Declaratives* dialog box, which allows you to select the pathname of the save file.

Click the *Save File* button to create a file after closing the *Save SQL language declaratives* dialog box.

Click the *Cancel* button to close the *DCLRES* dialog box.

## Importing a Table

The *Import tables from IXF format* dialog box allows you to:

- Import a file in IXF format and create an NS-DB table in the current database.
- Select the directory that contains the .IXF file that you want to import.

This *Import tables from IXF format* dialog box is displayed by selecting the *Import* option from the *Tables* menu.

Click the *Import...* button to open the *Import from Drive:\Path\Fname.IXF* dialog box, which displays the name and characteristics of the table that you want to import.

The functions in the *Storage Location* group have not yet been implemented. They will be implemented in NS-DB Version 2.

The PRIMARY KEY ELECTION group contains full details about the import. These details are preceded by the code SQLIMP+000X. The first two lines, which begin with SQLIMP+0002 and +0005, describe the file format and presentation of the various columns being imported.

You can only access lines that begin with SQLIMP+0006, which display the various columns in the table being imported. They will be used to define the PRIMARY KEY INDEX.

You must select the column used for the primary index from the List box. Click the *Get Cluster* Push button to display the CLUSTERING NUMBERS that have not yet been used in the NS-DB database. CLUSTERING NUMBERS can also be entered directly in the corresponding fields. To associate them with the selected column, you must click the >> button.

To build a complex primary index, simply repeat this operation for each additional column that you want to include.

Click *OK* to start the import into the current database. The import operation is accompanied by the message: "TableName import in progress...".

After closing the dialog box, the name of the imported table will be displayed in the *DatabaseName Transaction N* dialog box.

## Exporting a Table

Selecting the *Export...* option from the *Tables* menu displays the *Export TableName using IXF/PC format* dialog box, which allows you to export an NS-DB table into a file in IXF format.

This dialog box displays the current directory by default.

The table that you want to export must first be selected from the List box in the *DatabaseName Transaction N* dialog box.

You can modify this path by selecting a new drive and directory. The new file will be stored in the specified directory.

---

The name of the table to be exported is automatically displayed with the .IXF extension in the *File Name* field. You can modify this name by entering another name of your choice in this field.

Click the *Export* button to close the dialog box and start exporting the table into the specified pathname. The export is accompanied by the message: "TableName Export in Progress...".



The exported table will only contain the primary key index. Any secondary keys will need to be recreated via the *New index definition on TableName* dialog box. This box is displayed by clicking the *Add Index...* button in the *DatabaseName.TableName* window.

## Displaying Monitor Activity

Select the Monitor icon in the main window to display the *Monitor information* dialog box, which allows you to enter the name of the monitor specified in the initialization file.

The Monitor must be defined in the initialization file as follows:

```
[System]
DBDIR='C:\NS-DB', TRACE='NO', MONITOR='DBMON';
Sysid='NAME';
Journal =YES;
```

The activity monitor allows you to track and analyze database resources and access times for NS-DB applications running on a given machine.

After inputting the Entry field, click the button to open the *Monitor Activity* dialog box, which displays:

- The number of transactions started.
- The total number of databases opened.
- The list of transactions started.

The fields in this window are explained in the section *Monitor Activity Dialog Box*.

To obtain statistics on an active transaction, simply double-click the relevant line in the *Active Transactions* List box.

The *Transaction N on DatabaseName* dialog box displayed on the screen contains run-time statistics for the transaction (number of statements executed and their duration).

The fields in this dialog box are explained in the section *Transaction N on DatabaseName Dialog Box*.

Click *Cancel* to close the *Transaction N on DatabaseName* dialog box. This will return you to the *Monitor activity* window where you can display performance statistics for other transactions.

In client/server mode, other transactions may have been started while the monitor was displayed. As a result, these transactions will not appear in the list of active transactions. Click the *Refresh* button to update the List box with the most recent transactions.

## Displaying Server Activity

In client/server mode, you can run NSQBE on the server station to obtain statistics about the server's activity.

You can display the *Server Activity* dialog box from the *Monitor Activity* dialog box.

To obtain this dialog box, select the monitor icon in the main window and enter the name of the monitor (as specified in the initialization file) in the *Monitor Information* box.

Once the *Monitor Activity* dialog box is displayed, select the *Server Activity...* button to display the *Server Activity* window.

The various details displayed are explained in the section *Server Activity Dialog Box*.

While displaying this information, other events may occur on the server. Consequently, the information displayed in the various fields will not include these events. To update this information, click *Refresh*.

---

## Appendix A

# NS-DBR Error Messages



This appendix describes all the error codes and messages for NS-DBR.

***This chapter  
explains***

- General Usage Guide
- Description.

## Contents

General Usage Guide .....	A-3
Description .....	A-4

## General Usage Guide

The error codes returned by NS-DB comply with SQL conventions:

- A serious error generates a negative error code, indicating that the system was unable to execute the command.
- A warning results in a positive error code, indicating that the DBMS was able to execute the command but obtained a non-standard result, as described by the message associated with the warning code.

*Example*

```
+100 NO ROW FOUND.
```

*Indicates that the table was read but no rows were found.*

A zero error code indicates that the SQL command was executed successfully.

---



## Description



The error codes described below are arranged in descending negative order followed by ascending positive order.

-116

**Cause:** The maximum number of authorized sessions, namely the number of CLIENTS NETBIOS connected simultaneously to the database server, has been reached. By default, this value is 16.

**Corrective Action:** Increase the number of the authorized sessions in the NSDB.INI file.

SQLLEX-110 \*\* ESDSERR (NSDB V.02)

**Cause:** The SPACE MANAGEMENT module encountered an error while attempting to access the database.

**Corrective action:** Check the physical disk space used to store the DATASPACEs defined for the database or contact Nat System.

SQLLEX-109 \*\* TIMESTAMP CONFLICT OCCURS ON UPDATE

SQLLEX-108 \*\* CURSOR POSITION LOST (ZCOM.RESERVED)

SQLLEX-107 \*\* COLUMN NOT GROUPED OR NOT AGGREGATE FX

**Cause:** The result produced by the GROUP BY verb is an aggregate of the projected columns, arranged by group. The SELECT statement can only project columns that appear in the GROUP BY clause and can only apply aggregate functions to columns that do not appear in the GROUP BY clause.

**Corrective action:** Check the projected columns and the GROUP BY clause.

SQLLEX-106 \*\* INVALID OR UNPREPARED STATEMENT

**Cause:** An attempt was made to execute a FETCH statement after executing a SELECT statement that has not been compiled by SQL\_EXEC or after executing a SELECT statement that generated a negative compiler error code.

**Corrective action:** Check the way return codes are tested by the application.

SQLLEX-105 \*\* MIXING FUNCTIONS AND PROJECTIONS

**Cause:** The SELECT clause of an SQL statement contains a mixture of column projections and column functions.

**Corrective action:** Ensure that the SQL clause contains either column projections or functions, but not both.

**SQLEX-104 \*\* NOT ALLOWED HERE**

**Cause:** A syntactically valid item appears in an invalid location within the SQL statement.

**Corrective action:** Check the corresponding expression.

**SQLEX-103 \*\* COMPARISON CONDITION RAISED**

**Cause:** The comparison contains a logical error.

**Corrective action:** Check the corresponding expression.

**SQLEX-102 \*\* ZERO DIVIDE CONDITION**

**Cause:** An expression attempted to divide by 0.

**Corrective action:** Check the corresponding expression.

**SQLEX-101 \*\* UNDEFINED SYMBOL**

**Cause:** The SQL variable, @x, has not been defined.

**Corrective action:** Ensure that you have defined all the @ SQL variables used by each MOVE instruction or SQL statement.

**SQLEX-100 \*\* ALREADY DEFINED**

**Cause:** The SQL variable, @x, has already been declared or the transaction, \*\* Y \*\*, has already been defined by a START TRANSACTION command.

**Corrective action:** Use another variable or transaction name.

**SQLEX-99 \*\* INVALID DB FX ON REMOTE DB**

**Cause:** The system attempted to call the DB\_GETCATADDRESS function, which cannot be executed remotely.

**Corrective action:** Internal error: contact Nat System.

**SQLEX-98 \*\* UNDEFINED DB IN TRANSACTION**

**Cause:** The logical database that precedes the message has not been opened by a START TRANSACTION.

**Corrective action:** Invoke a START TRANSACTION for the database displayed in the message.

**SQLEX-97 \*\* ORDER BY POSITION OVERFLOW**

**Cause:** The column number specified in the ORDER BY clause of a SELECT statement exceeds the number of columns in the RESULT table.

**Corrective action:** Check the number specified in the ORDER BY clause of the SELECT statement.

**SQLEX-96 \*\* NOT IN PRJ OR IN VALUE LIST**

**Cause:** The column indicated in the message is not part of the columns in the selection result. When inserting rows, all the columns in the table's Primary Key must be specified in the list of columns being inserted. All the columns specified in an ORDER BY clause (which determines the sort order of the selection result) must exist in the result table.

**Corrective action:** Specify a column name that is part of the selection result.

**SQLEX-95 \*\* COPY ROW DATA TYPE MISMATCH**

**Cause:** A COPY command has been executed on two tables whose column types do not match.

**Corrective action:** Check that the column types in the destination table match those in the source table.

**SQLEX-94 \*\* ORDER BY NOT ALLOWED ON THIS COLUMN**

**Cause:** The ORDER BY clause is not allowed for certain column types (GRAPHIC and VARGRAPHIC).

**SQLEX-93 \*\* SQL WORKDB ERROR**

**Cause:** Error accessing the INTERNAL WORK DATABASE in the SQL engine.

**Corrective action:** If you obtain the message 'INIT WORKDB ERROR', check that the SQL work database (SQLWKDB) has been defined correctly in the NSDB.INI file. Otherwise, contact Nat System.

**SQLEX-92 \*\* UNDEFINED CURSOR**

**Cause:** An attempt was made to execute a NEXT or PREV search statement using a cursor that has not been defined by OPENCURSOR + SEARCH.

**Corrective action:** Check the code that performs the search.

**SQLEX-91 \*\* MONITOR INIT ERROR**

**Cause:** Error encountered while initializing the DBMON utility.

**SQLEX-90 \*\* UPDATE OPERATION ON READONLY DB**

**Cause:** An update command (INSERT, MODIFY, etc.) has been executed on a database defined as READONLY in the NSDB.INI file.

**Corrective action:** Remove the READONLY parameter defined for the database in the NSDB.INI file.

**SQLEX-89 \*\* NET BUFFER OVERFLOW**

**Cause:** Internal error.

**Corrective action:** Contact Nat System.

**SQLEX-88 \*\* REMOTE BUFFER SIZE TOO SHORT**

**Cause:** The communication buffer is too short.

**Corrective action:** Contact Nat System.

**SQLEX-87 \*\* SQL STATEMENT NOT READY(COMPILED)**

**Cause:** An attempt was made to execute an SQL statement that has not been compiled or to execute a statement that had compiler errors.

**Corrective action:** Correct or compile the statement.

**SQLEX-86 \*\* INVALID NULL INDICATOR TYPE**

**Cause:** The variable type used for null indicators must be *Integer*.

**Corrective action:** Define the null indicator as an integer.

**SQLEX-85 \*\* MISSING HOST ADDRESSES TO BIND**

**Cause:** The number of columns exceeds the number of host variables supplied in the SQL statement.

**Corrective action:** Add the missing host variables to the statement.

**SQLEX-84 \*\* JOIN ARGUMENT TYPE MISMATCH**

**Cause:** An attempt was made to join two mismatching or incompatible columns

**Corrective action:** Check the join arguments in the statement.

**SQLEX-82 \*\* MIXING INNER AND OUTER JOIN**

**Cause:** Inner and outer joins cannot be combined in the same statement.

**Corrective action:** Modify the join operations.

**SQLEX-81 \*\* DOES NOT SUPPORT NULL**

**Cause:** An attempt was made to query a NOT NULL column using a NULL VALUE argument.

**Corrective action:** Modify the search argument.

**SQLEX-80 \*\* INTERNAL STRUCT OVERFLOW**

**Cause:** Internal resource overflow in module.

**Corrective action:** Contact Nat System.

**SQLEX-79 \*\* UNDEFINED FROM TABLE**

**Cause:** The table specified in the FROM clause does not exist in the CATALOG.

**Corrective action:** Check the table name in the FROM clause.

**SQLEX-78 \*\* INDEX SIZE OVERFLOW**

**Cause:** The total physical size of the index columns exceeds 250 bytes.

**Corrective action:** Reduce the size of the index.

**SQLEX-77 \*\* NO MORE CURSORS TO CLOSE ID**

**Cause:** An attempt was made to close an undefined cursor.

**Corrective action:** Check the context of the close cursor command.

**SQLEX-76 \*\* TOO MANY OPENED CURSORS**

**Cause:** Cursor table overflow.

**Corrective action:** Reduce the number of cursors opened.

**SQLSY-075 \*\* INVALID JOIN EXPRESSION**

**Cause:** Syntax error in join expression.

**Corrective action:** Check the syntax of the join.

**SQLEX-074 \*\* CONVERSION CONDITION RAISED**

**Cause:** An attempt was made to convert incompatible data types .

**Corrective action:** Check the conversions requested by the statement.

**SQLSY-073 \*\* NOT IMPLEMENTED**

**Cause:** Function not implemented.

**Corrective action:** Please wait!

**SQLSY-072 \*\* UNDEFINED SQL REFERENCE**

**Cause:** The SQL statement references an undefined object.

**Corrective action:** Check the SQL statement and ensure that all the object names are valid.

**SQLEX-071 \*\* SQL INTERNAL TABLE OVERFLOW**

**Cause:** Internal table overflow.

**Corrective action:** Contact Nat System.

**SQLEX-070 \*\* INT USAGE MSG2**

**Cause:** This error code number is reserved for internal use and should never be returned to the calling application.

**Corrective action:** Contact Nat System.

**SQLEX-069 \*\* ALREADY DEFINED ON NETWORK****SQLEX-068 \*\* COLUMN DATA TYPE MISMATCH****SQLEX-067 \*\* INT USAGE MSG1**

**Cause:** This error code number is reserved for internal use and should never be returned to the calling application.

**Corrective action:** Contact Nat System.

**SQLEX-066 \*\* PKEY INDEX ALREADY DEFINED**

**Cause:** The PRIMARY KEY INDEX used to identify a table can only be defined once, when the table is created.

**Corrective action:** Remove the PRIMARY KEY option from the CREATE INDEX statement.

**SQLEX-065 \*\* POSITION INDEX DROPPED**

**Cause:** The position index used by a cursor has been dropped by a concurrent transaction (DROP statement) and is now unusable.

**Corrective action:** Reposition the cursor using the SEARCH statement.

**SQLEX-064 \*\* TABLE PKEY INDEX MISSING**

**Cause:** An attempt was made to create a table without specifying a PRIMARY KEY INDEX, which must be defined for each table.

**Corrective action:** Define the PRIMARY KEY and recreate the table.

**SQLEX-063 \*\* PKEY INDEX CANNOT BE BUILT**

**Cause:** A BUILD INDEX statement has been executed for a PRIMARY KEY INDEX, which is meaningless.

**Corrective action:** Re-execute the BUILD INDEX statement for an index other than the PRIMARY KEY.

**SQLLEX-062 \*\* DBSPACE INVALID PATH**

**Cause:** The pathname of the physical file defined for the TABLESPACE is invalid.

**Corrective action:** Change the pathname specified by the CREATE TABLESPACE statement and re-execute it. Ensure that you have not specified an invalid pathname in the NSDB.INI file when opening an existing database.

**SQLLEX-061 \*\* DBSPACE SHARING VIOLATION**

**Cause:** An NS-DB engine attempted to start a transaction on a database for which a transaction has already been started by another NS-DB engine in a concurrent OS/2 session.

**Corrective action:** Activate Client/Server mode (NSDBCLI) to allow concurrent access to a database.

**SQLLEX-060 \*\* CAN'T BE DROPPED**

**Cause:** The requested object cannot be deleted (e.g. DROP TABLE SYSTABLES: attempt to delete an NS-DB system table).

**Corrective action:** Check the DROP statement.

**SQLLEX-059 \*\* CLEANUP INIT ERROR**

**Cause:** The REORG command encountered a start-up error.

**Corrective action:** Contact Nat System.

**SQLLEX-058 \*\* BUILD INIT ERROR**

**Cause:** The BUILD command encountered a start-up error.

**Corrective action:** Contact Nat System.

**SQLLEX-057 \*\* PHYSICAL PAGE IN USE**

**Cause:** A transaction attempted to update a physical page that is already in use by a concurrent transaction. This error is only returned if JOURNAL=YES is specified in the NSDB initialization file.

**Corrective action:** Wait for the concurrent transaction to execute a COMMIT or ROLLBACK before re-executing the update statement.

**SQLLEX-056 \*\* DATABASE RECORD TOO LONG**

**Cause:** This return code is used internally by NSDB and should never be returned to the caller.

**Corrective action:** Contact Nat System.

**SQLEX-055 \*\* INFORMATION NOT FOUND IN CATALOG**

**Cause:** An attempt was made to DROP an object that does not exist in the CATALOG.

**Corrective action:** Check the name of the object that you want to drop.

**SQLEX-054 \*\* DEST INTG INDEX DROP NOT ALLOWED**

**Cause:** A FOREIGN KEY INDEX cannot be deleted by a DROP INDEX command.

**SQLEX-053 \*\* PKEY INDEX DROP NOT ALLOWED**

**Cause:** A PRIMARY KEY INDEX cannot be deleted by a DROP INDEX command.

**Corrective action:** Use the DROP TABLE command if you want to drop the TABLE and the PKEY INDEX.

**SQLEX-052 \*\* DATA OVERFLOW**

**Cause:** Data overflow in the USER data area while inserting or updating a table row.

**Corrective action:** Check the data structure in your update area. If required, use the DCLGEN utility to display the precise structure of the SEGMENT or RECORD defined for the table in the NS-DB catalog.

**SQLEX-051 \*\* NO MORE CLIENT TRANSACTIONS AVAILABLE**

**Cause:** The maximum number of active transactions allowed for the NS-DB CLIENT module (NS0XDBCL) has been reached.

**Corrective action:** Reduce the number of simultaneous DB\_OPEN or DB\_STARTRN commands in your application.

**SQLEX-050 \*\* CLIENT INIT FAILURE**

**Cause:** Error initializing CLIENT for CLIENT/SERVER mode.

**Corrective action:** Investigate the NETBIOS error or the communication module error associated with this message.

**SQLEX-049 \*\* UNMANAGED CATALOG TABLE**

**Cause:** SYSTEM CATALOG integrity error.

**Corrective action:** Contact Nat System.

**SQLEX-048 \*\* COMMUNICATION MSG OVERFLOW**

**Cause:** Communication buffer overflow. This error may occur if you define very large table structures with rows that exceed 8192 bytes.



**Corrective action:** Reduce the size of your table structures: a well-defined table (i.e. one that complies with the normalization rules for relational databases) should contain a restricted number of columns!

**SQLEX-047 \*\* DB FX INVALID ON CATALOG**

**Cause:** Update statements cannot be applied to SYSTEM CATALOG tables (SYSCOLUMNS, SYSINDEXES, etc.).

**Corrective action:** Use the commands provided by the Data Description Language (D.D.L.) to manipulate these tables.

**SQLEX-046 \*\* SERVER WAIT FAILURE**

**Cause:** Communication error: server wait.

**Corrective action:** Investigate the NETBIOS error or the communication module error associated with this message.

**SQLEX-045 \*\* SERVER NOT RESPONDING**

**Cause:** Communication error: client waiting for server.

**Corrective action:** Investigate the NETBIOS error or the communication module error associated with this message. This message is often returned when the database SERVER has aborted.

**SQLEX-044 \*\* SERVER NOT READY SQLEX-044**

**Cause:** Communication error: cannot connect to server.

**Corrective action:** Investigate the NETBIOS error or the communication module error associated with this message.

**SQLEX-043 \*\* SERVER CALL ERROR**

**Cause:** Communication error: server call.

**Corrective action:** Investigate the NETBIOS error or the communication module error associated with this message. This message is often returned when the database SERVER has aborted after starting a remote transaction.

**SQLEX-042 \*\* CONNECTION FAILURE**

**Cause:** Error initializing the Client/Server layer.

**Corrective action:** Investigate the NETBIOS error or the communication module error associated with this message.

**SQLEX-041 \*\* SEMAPHORE ERROR**

**Cause:** Internal semaphore error.

**Corrective action:** Contact Nat System.

**SQLEX-040 \*\* DB RECORD LOCKED**

**Cause:** An attempt was made to access a locked record (reserved for a future version).

**SQLEX-039 \*\* TRANS JOURNALLING ERROR**

**Cause:** Internal transaction journalling error.

**Corrective action:** Contact Nat System.

**SQLEX-038 \*\* INIT FUNCTION NOT PERFORMED**

**Cause:** An attempt was made to execute an NS-DB function before calling the DB\_INIT initialization function.

**Corrective action:** Call DB\_INIT before calling any other NS-DB functions.

**SQLEX-037 \*\* UNKNOWN DATABASE ID**

**Cause:** An NS-DB transaction has been started on a database that has not been defined in the .INI file.

**Corrective action:** Ensure that your database has been defined in the NSDB.INI file.

**SQLEX-036 \*\* ENVIR STRING MISSING**

**Cause:** The NS-DB= environment parameter, used to define the location of the NSDB initialization file, has not been set.

**Corrective action:** Add the line 'SET NS-DB=Drive:\Path\NSDB.INI' to the CONFIG.SYS file and reboot the system.

**SQLEX-035 \*\* INIT FILE READ FAILURE**

**Cause:** The system was unable to open the NS-DB initialization file.

**Corrective action:** Ensure that the drive, directory and file specified by the NS-DB environment parameter are valid.

**SQLEX-034 \*\* LOGICAL NAME TOO LONG**

**Cause:** The maximum length allowed for a logical database name in the .INI file is 8 bytes.

**Corrective action:** Modify the logical database name specified by the .INI file and the DB\_OPEN and DB\_STARTRN commands in your applications.

**SQLEX-033 \*\* INIT FILE EMPTY**

**Cause:** The current initialization file is empty.

**Corrective action:** Enter information in the .INI file.

**SQLEX-032 \*\* INIT FILE TOO BIG**

**Cause:** The size of the initialization file cannot exceed 64 Kb.

**SQLEX-031 \*\* INIT FILE OPEN FAILURE**

**Cause:** Error encountered while opening the NSDB.INI file.

**Corrective action:** Ensure that the NS-DB environment parameter specifies an existing file with a valid drive and pathname.

**SQLEX-030 \*\* INTERNAL DATA OVERFLOW**

**Cause:** Internal work buffer overflow.

**Corrective action:** Contact Nat System.

**SE SQLEX-029 \*\* SEQUENTIAL POS ERROR**

**Cause:** Sequential position lost during a NEXT or PREV command.

**Corrective action:** Check that you are using the same cursor as the one used by the DB\_SEARCH command.

**SQLEX-028 \*\* INVALID COLUMN**

**Cause:** Internal error.

**Corrective action:** Contact Nat System.

**SQLEX-027 \*\* PHYSICAL RECORD MISMATCH**

**Cause:** The length of an item in the physical table row does not match the length defined for this item in the SYSTEM CATALOG.

**Corrective action:** Contact Nat System.

**SQLEX-026 \*\* INTERNAL INDEXING ERROR**

**Cause:** Invalid index structure for a table row.

**Corrective action:** Contact Nat System.

**SQLEX-025 \*\* UNEXPECTED KSDS FUNCTION**

**Cause:** Internal error.

**Corrective action:** Contact Nat System.

**SQLEX-024 \*\* REFERENTIAL INTEGRITY MESS IS DYNAMICALLY BUILT**

**Cause:** Referential integrity violation.

**Corrective action:** This error must be handled by the application according to the missing reference (e.g. INVOICE defined for a non-existent CLIENT). For further details on referential integrity, refer to the description of the CREATE INDEX Index\_Referential\_Integrity command.

**SQLSX-023 \*\* INV SPACE**

**Cause:** Physical error encountered while accessing a TABLESPACE due to an invalid drive or pathname.

**Corrective action:** Check the names and locations of the physical objects associated with the database's TABLESPACE.

**SQLSX-022 \*\* INCOMPLETE CATALOG DROP**

**Cause:** Some of the memory allocated for the SYSTEM CATALOG was not freed when the database was closed.

**Corrective action:** Contact Nat System.

**SQLSX-021 \*\* CATALOG INTEGRITY PB**

**Cause:** SYSTEM CATALOG integrity lost during memory load.

**Corrective action:** Contact Nat System.

**SQLSY-020 \*\* EXCEEDED**

**Cause:** Limit exceeded for the object specified in the message.

**SQLSX-019 \*\* DOES NOT EXIST IN CATALOG**

**Cause:** The corresponding object does not exist in the SYSTEM CATALOG.

**Corrective action:** Check the name of the object and, if required, create it using Data Description Language (D.D.L.) commands: CREATE TABLE, CREATE INDEX, etc.

**SQLSY-018 \*\* INVALID CLU NUMBER**

**Cause:** The value of a CLUSTERING NUMBER must lie between 1 and 254.

**Corrective action:** Keep within this range of values.

**SQLSX-017 \*\* EXISTS IN CATALOG**

**Cause:** An attempt was made to create an object that is already defined in the SYSTEM CATALOG.

**Corrective action:** Change the object name and resubmit the request.

---

**SQLLEX-016 \*\* DATA BASE TRANSACTION NOT OPENED**

**Cause:** Before manipulating a database (DB\_SEARCH, DB\_NEXT, etc.), a transaction must be started on it using the DB\_OPEN (NCL) or DB\_STARTRN (3GL) command.

**Corrective action:** Execute the DB\_STARTRN or DB\_OPEN command and resubmit your request.

**SQLLEX-015 \*\* SQL IDENT EXCEEDS 18 BYTES**

**Cause:** The length of an SQL identifier (table name, column name, etc.) exceeds 18 characters.

**Corrective action:** Reduce the length of the identifier.

**SQLLEX-014 \*\* UNKNOWN DATABASE FUNCTION**

**Cause:** The function code used is not an NSDB function code.

**Corrective action:** (3GL) Ensure that the DB\_FX function specified by the DB\_CALL is valid.

**SQLLE-013 \*\* UNEXPECTED END OF SOURCE**

**Cause:** Unexpected end of SQL source encountered while compiling an SQL command.

**Corrective action:** Check the SQL source submitted to the SQL compiler.

**SQLLE-012 \*\* LEXICAL ERROR**

**Cause:** An SQL compiler error occurred during the lexical phase.

**Corrective action:** Contact Nat System.

**SQLLE-011 \*\* CONSTANT INVALID OR TOO LONG**

**Cause:** Invalid constant structure detected during the compiler's lexical phase (e.g. string too long).

**SQLSY-010 \*\* SOURCE EXCEEDS 64 Kb**

**Cause:** An SQL statement cannot exceed 64 Kb.

**Corrective action:** Check the statement submitted to the SQL compiler.

**SQLSY-009 \*\* INVALID SEPARATOR**

**Cause:** Invalid separator.

**Corrective action:** Check the statement submitted to the SQL compiler.

**SQLSY-008 \*\* END OF SOURCE STREAM**

**Cause:** Unexpected end of SQL source encountered by the compiler.

**Corrective action:** Check the statement submitted to the SQL compiler.

**SQLSY-007 \*\* EXPECTED**

**Cause:** The compiler expected to find the verb displayed before 'EXPECTED' in the SQL statement located at the specified line and column.

**Corrective action:** Correct the statement.

**SQLSY-006 \*\* SYNTAX ERROR**

**Cause:** The verb displayed before 'SYNTAX ERROR' produces a syntax error for the statement that generated this message.

**Corrective action:** Correct the statement.

**SQLEX-005 \*\* DB DUPLICATE**

**Cause:** An existing table row in the database contains a PRIMARY KEY that is identical to the one you are trying to insert.

**Corrective action:** Change the PRIMARY KEY for the row and reinsert it.

**SQLEX-004 \*\* KSDSERR**

**Cause:** The SPACE MANAGEMENT module encountered an error while attempting to access the database.

**Corrective action:** Check that there is enough space left on the disks used to store the database's TABLESPACES and ensure that the specified paths exist on your machine. You can also use the DBCHK utility to check that the physical database structure is not damaged.

**SQLEX-003 \*\* INV CATALOG OBJECT TYPE**

**Cause:** Unknown object type stored in the SYSTEM CATALOG.

**Corrective action:** Contact Nat System.

**SQLEX-002 \*\* TOO MANY TRANSACTIONS**

**Cause:** The total number of transactions currently managed by NS-DB exceeds the maximum allowed (50).

**Corrective action:** Reduce the number of active transactions.

**SQLEX-001 \*\* TOO MANY OPENED DATA BASES**

**Cause:** The total number of databases opened by NS-DB exceeds the maximum allowed (15).

**Corrective action:** Reduce the number of databases opened.

**SQLEX000 \*\* SUCCESSFUL DB CALL**

**Cause:** Normal return from an NS-DB call.

**SQLEX+001 \*\* EMPTY DATABASE**

**Cause:** The database is empty.

**SQLEX+002 \*\* DATABASE RECORD NOT FOUND**

**Cause:** Following a database search operation (SEARCH, NEXT or PREV), no more records that match the search criteria were found in the database.

**SQLEX+003 \*\* IGNORED**

**Cause:** During an NS-DB operation, the system ignored the redundant item in the user query (displayed before 'IGNORED').

**SQLEX+004 \*\* CATALOG POSITION LOST**

**Cause:** Following a concurrent catalog operation (usually DROP INDEX), the current position maintained by an NS-DB cursor was lost.

**SQLEX+005 \*\* LOCKED RECORD READ**

**Cause:** An attempt was made to read a locked record (not implemented for the time being).

**SQLEX+006 \*\* INTERNAL USAGE MSG**

**Cause:** This message should never be returned to the calling entity.

**SQLEX+007 \*\* CARTESIAN PRODUCT IN PROGRESS**

**Cause:** The SQL statement submitted to NS-DB invokes a Cartesian product of two tables. Generally speaking, the result of this type of operation is meaningless (e.g. SELECT \* FROM CLIENT , INVOICE).

**SQLEX+008 \*\* EXCESSIVE HOST VARIABLES IGNORED**

**Cause:** The number of host variables passed by the SQL statement exceeds the number of columns accessed by the statement. NSDB ignored the redundant host variables (e.g. SELECT Z INTO :X , :Y FROM CLIENT).

**SQLEX+009 \*\* HOST VARIABLE LIST TOO SHORT**

**Cause:** The number of host variables passed in the SQL statement is less than the number of columns accessed by the statement (e.g. SELECT Z , Y INTO :X FROM CLIENT).

**SQLEX+010 \*\* NO ROW FOUND OR RESULT OF QUERY IS EMPTY TABLE**

**Cause:** Standard SQL message (error code +100), used to indicate that none of the columns in the database satisfy the specified search criteria.

**SQLEX+011 \*\* INT USAGE MSG3**

**Cause:** This message should never be returned to the calling entity.

**SQLEX+012 \*\* TRYING TO CLOSE UNDEFINED CURSOR**

**Cause:** An attempt was made to close a cursor that has not yet been opened.

**SQLEX+013 \*\* RECORD NOT MATCHING REQUEST**

**Cause:** Internal message used by the NS-DBR compiler.

**Corrective action:** If you receive this message, contact Nat System.

**SQLEX+014 \*\* AGGREGATE XXX IMPLIES SEQUENTIAL TABLE SCAN**

**Cause:** A SELECT statement defined with a LIKE, MAX, MIN, COUNT, AVG or SUM predicate will result in a sequential scan of the table. This can lead to poor performance when the SELECT statement is executed.

**SQLEX+015 \*\* JOIN ON NON-INDEX COLUMN**

**Cause:** An attempt was made to join two unindexed columns, which may result in very poor performance when the query is executed. The optimizer warns against these operations as a precaution.

**Corrective action:** Add the required indexes using the CREATE INDEX command, execute a BUILD command to build the index and try again.

**SQLEX+016 \*\* INTERNAL USAGE MSG**

**Cause:** This message is reserved for internal use and should never be returned to the caller.

**Corrective action:** Contact Nat System.

**SQLEX+017 \*\* DATA TRUNCATION OCCURS MSG**

**Cause:** While transferring data or converting to character format, the system truncated the user data or the information read from a column.

**Corrective action:** Ensure that you are not attempting to insert columns using data buffers that are too large or retrieve columns into variables that are too small.

**SQLEX+18 \*\* PRIMARY KEY UPDATE FORBIDDEN**

**Cause:** The SQL statement was executed successfully. However, the primary key was not updated since this is prohibited.



**SQLEX+21 \*\* DATABASE IS VIRTUAL**

**Cause:** The database that you have opened has been defined as a virtual database in the NSDB.INI file.