HP NonStop ODBC Server Reference Manual

Abstract

This manual describes the HP NonStop™ ODBC Server, a product that allows applications written for the Microsoft ODBC interface, Microsoft SQL Server interface, and Sybase SQL Server interface to access an HP NonStop SQL/MP database. This manual describes the NonStop ODBC Server hardware and software requirements, architecture, the CORE SQL language, the Transact-SQL language, stored procedures, pass-through mode, customized NonStop SQL/MP catalogs, and mapping tables.

Product Version

NonStop ODBC Server 2.0

Supported Release Version Updates (RVUs)

This manual supports D32.00 and all subsequent D-series RVUs, and G01.00 and all subsequent G-series RVUs, until otherwise indicated in a new edition.

Part Number	Published
429151-002	September 2003

Document History

Part Number	Product Version	Published
138789	NonStop ODBC Server 2.0	April 1998
424092-001	NonStop ODBC Server 2.0	September 1999
426087-001	NonStop ODBC Server 2.0	May 2000
429151-001	NonStop ODBC Server 2.0	January 2001
429151-002	NonStop ODBC Server 2.0	September 2003

HP NonStop ODBC Server Reference Manual

Glossary | Index | Examples | Figures | Tables

3. CORE SQL Language (continued)

3. CORE SQL Language (continued)

4. Transact-SQL Language (continued)

4. Transact-SQL Language (continued)

4. Transact-SQL Language (continued)

6. Using Pass-Through Mode (continued)

7. Managing Customized Catalogs

8. HP NonStop ODBC Server Mapping Tables (continued)

Examples

Figures

Figures (continued)

Tables

Tables (continued)

Tables (continued)

Tables (continued)

What's New in This Manual

Manual Information

Abstract

This manual describes the HP NonStop™ ODBC Server, a product that allows applications written for the Microsoft ODBC interface, Microsoft SQL Server interface, and Sybase SQL Server interface to access an HP NonStop SQL/MP database. This manual describes the NonStop ODBC Server hardware and software requirements, architecture, the CORE SQL language, the Transact-SQL language, stored procedures, pass-through mode, customized NonStop SQL/MP catalogs, and mapping tables.

Product Version

NonStop ODBC Server 2.0

Supported Release Version Updates (RVUs)

This manual supports D32.00 and all subsequent D-series RVUs, and G01.00 and all subsequent G-series RVUs, until otherwise indicated in a new edition.

Part Number	Published
429151-002	September 2003

Document History

Part Number	Product Version	Published
138789	NonStop ODBC Server 2.0	April 1998
424092-001	NonStop ODBC Server 2.0	September 1999
426087-001	NonStop ODBC Server 2.0	May 2000
429151-001	NonStop ODBC Server 2.0	January 2001
429151-002	NonStop ODBC Server 2.0	September 2003

New and Changed Information

This edition contains the following changes:

- Under Using Pass-Through Syntax, a line is added on page 6-3 to clarify that the syntax shown is not supported by the SQL PREPARE and the SQL EXECUTE statements.

- Under USERCAT INSTALL Statement on page 7-20, the `EMPTY` option is added.
- Under Effects of USERCAT INSTALL on page 7-21, a bullet point is added describing what will happen if the `EMPTY` option is specified while installing the catalog.
- The *`output-filename`* is updated to indicate that the log table is created in the catalog where VALIDATE, REFRESH, or INSTALL is being run, in:
    - USERCAT INSTALL Statement on page 7-20
    - USERCAT REFRESH Statement on page 7-29
    - USERCAT VALIDATE Statement on page 7-33
- Under ADD NET_SERVICE on page 7-53, the `SO_KEEPALIVE {0 | 1 }` option is updated, indicating that the default value is 1.
- Under ADD PROFILE on page 7-66, the `DEFAULT_SCHEMA` option is updated to convey that the *`schema-name`* cannot be used for qualification of the object name for DDL statements.
- Under ADD SERVERCLASS on page 7-81, the range for *`available-servers`* and *`max-servers`* is updated, and the `GOV_ERROR` option is added.
- START SCS on page 7-100 is updated, describing what happens if the user is not a registered ODBC user.
- This publication is updated to reflect new product names:
    - Since product names are changing over time, this publication might contain both HP and Compaq product names.
    - Product names in graphic representations are consistent with the current product interface.
- All references to Windows 3.1, Windows 3.1 NEC and 3.11 Workgroup are removed because they are no longer supported.

About This Manual

This manual describes the HP NonStop ODBC Server, which allows applications written for the Microsoft ODBC interface, Microsoft SQL Server interface, and Sybase SQL Server interface to access a HP NonStop SQL/MP database. ODBC clients are supported on Windows workstations; SQL Server clients are supported on DOS, Windows, OS/2, and UNIX workstations. The NonStop ODBC Server runs on the HP

server, although one of its components, the HP NonStop ODBC/MP Driver, runs on the client workstation.

This manual contains reference information for the NonStop ODBC Server, describes the ODBC or SQL Server features and statements that the NonStop ODBC Server supports, and describes how the NonStop ODBC Server accommodates the differences between ODBC or SQL Server and NonStop SQL/MP.

Audience

This manual is intended for system administrators and database administrators who perform the following functions:

- Initially install the NonStop ODBC Server or migrate from an earlier version of the NonStop ODBC Server
- Prepare NonStop SQL/MP databases for access with the NonStop ODBC Server
- Prepare application programs to run with the NonStop ODBC Server
- Modify ODBC or SQL Server application programs to run with the NonStop ODBC Server

End users of applications that access NonStop SQL/MP databases can also use this manual for its language and feature descriptions.

Readers of this manual should be familiar with the following:

- The ODBC CORE SQL dialect of the SQL language or the SQL Server Transact-SQL dialect
- The fundamentals of NonStop SQL/MP
- The fundamentals of the HP NonStop Kernel operating system
- The operation of the PC or workstation applications being used

Organization of This Manual

This manual contains the following sections:

- Section 1, Introduction, provides an overview of the NonStop ODBC Server, describes the hardware and software requirements, and summarizes NonStop ODBC Server support of ODBC or SQL Server features.

- Section 2, Architecture Overview, shows how the NonStop ODBC Server reconciles the architecture of NonStop SQL/MP with the architecture of ODBC or SQL Server. It describes how ODBC or SQL Server, NonStop SQL/MP, and the NonStop ODBC Server manage the database and database objects, describe and access database objects, implement security, handle transactions and data consistency, allocate storage, support user access, and maintain data integrity.
- Section 3, CORE SQL Language, describes names, data types, functions, and other elements used in SQL statements, as well as the CORE SQL statements the NonStop ODBC Server supports. This section presents the syntax for each statement and describes how the statement differs from ODBC and NonStop SQL/MP statements.
- Section 4, Transact-SQL Language, describes names, data types, functions, and other elements used in SQL statements, as well as the Transact-SQL statements the NonStop ODBC Server supports. This section presents the syntax for each statement and describes how the statement differs from SQL Server and NonStop SQL/MP statements.
- Section 5, Stored Procedures, describes the NonStop ODBC Server support of stored procedures and gives guidelines for users who need to write stored procedures, using either the C programming language or COBOL85.
- Section 6, Using Pass-Through Mode, describes how to use pass-through mode to execute NonStop SQL/MP statements, catalog utility commands, and trace commands.
- Section 7, Managing Customized Catalogs, describes how to customize a catalog and how to maintain customized catalogs, including the use of NOSUTIL, the NonStop ODBC Server utilities.
- Section 8, HP NonStop ODBC Server Mapping Tables, describes the mapping tables that the NonStop ODBC Server uses to implement an ODBC or SQL Server database.
- Section 9, UNIX Notes, contains information for users who are accessing the NonStop ODBC Server from a UNIX workstation.
- Appendix A, Summary of Support for ODBC Features, lists ODBC language features and indicates which features the NonStop ODBC Server supports.
- Appendix B, Summary of Support for SQL Server Features, lists SQL Server language features and indicates which features the NonStop ODBC Server supports.
- Appendix C, Summary of Support for ODBC 2.10 Functions, lists ODBC functions and indicates which functions the NonStop ODBC Server supports.
- Appendix D, Summary of System Installation Defaults, lists the default values, set at system installation time by the installation process, for named attributes of the NonStop ODBC Server.

- Appendix E, Changing Passwords in a Three-Tier Environment, describes how to change a password in an ODBC configuration in which the ODBC/MP driver is invoked by an application server on behalf of a user application.
- Appendix F, Creating Partitioned Tables, describes how to create a partitioned table using a Partition Overlay Specification (POS) Template and the CREATE TABLE statement.
- Glossary defines technical terms used in this manual, including ODBC terms, SQL Server terms, NonStop SQL/MP terms, and NonStop ODBC Server terms.

Related Manuals

Two other manuals in the NonStop ODBC Server library are:

- HP *NonStop ODBC Server Installation and Management Manual*—describes the installation, configuration, management, and tuning of the NonStop ODBC Server and its components.
- HP *NonStop ODBC Server Messages Manual*—documents the error and warning messages generated by the NonStop ODBC Server components.

ODBC Documentation

For information about Microsoft ODBC, version 2.0, see the *Microsoft ODBC 2.0 Programmer's Reference and SDK Guide*, © 1994, Microsoft Corporation, ISBN 1-55615-658-8.

For additional information about ODBC CORE SQL, see the *X/Open and SQL Access Group SQL CAE specification* (1992).

For information about workstation tools that are compatible with the NonStop ODBC Server, see the "NonStop ODBC Server Data Sheet," which your service provider can obtain for you.

SQL Server Documentation

For information about SQL Server, see the Microsoft SQL Server library. If you are using the Sybase SQL Server software, use the Sybase manuals rather than the Microsoft manuals.

NonStop SQL/MP Documentation

All users need the following NonStop SQL/MP manuals:

- *NonStop SQL/MP Reference Manual*—provides basic NonStop SQL/MP reference information. The manual includes all standard NonStop SQL/MP statements and discusses essential concepts for using the statements, such as locking, parallelism, buffering, DDL/DML concurrency considerations, and DEFINEs.

- *NonStop SQL/MP Messages Manual*—lists the NonStop SQL/MP messages for all NonStop SQL/MP components. It also includes file-system messages that can be issued only on NonStop SQL/MP objects

If you are unfamiliar with NonStop SQL/MP, the *Introduction to NonStop SQL/MP* provides feature, functional, and conceptual overviews.

Other HP Documentation

The following manuals contain information about the HP NonStop servers and software products used with the NonStop ODBC Server:

- *File Utility Program (FUP) Reference Manual*—describes the utility program used for managing files on NonStop systems.
- *Guardian User's Guide*—provides task-oriented instructions for using the HP Tandem Advanced Command Language (TACL) and various Guardian environment utilities.
- *TACL Reference Manual*—presents the syntax and operations of the standard commands and functions available in the command interpreter.
- *NonStop Transaction Management Facility (TMF) Planning and Configuration Guide* and *NonStop Transaction Management Facility (TMF) Operations and Recovery Guide*—describe how to use the HP NonStop Transaction Management Facility (TMF) to protect a database against disk, system, or program failures.
- *TCP/IP Management Programming Manual*—describes the programmatic interface to the HP NonStop TCP/IP data communication software.
- *Multilan/TLAM Programming Manual*—describes the programmatic interfaces provided by the HP Tandem LAN Access Method (TLAM) application programs.
- *PTrace Reference Manual*—describes how to use the PTrace utility to display trace files created through the use of the Subsystem Control Facility (SCF).
- *NonStop TS/MP Pathsend and Server Programming Manual* and the *Pathway/TS TCP and Terminal Programming Guide*–describes how to develop stored procedures in the Pathway environment.
- The Subsystem Control Facility (SCF) manual for the communication protocol you are using.

Sun Workstations

For users who are accessing the NonStop ODBC Server from a Sun workstation, the following manuals contain background information about Sun UNIX:

- *SunOS Reference Manual*—describes UNIX calls and commands available from a Sun interface terminal.
- *System and Network Administration Manual*—contains administrative information about using a Sun system.

These manuals are included with a Sun workstation and are available from Sun Microsystems.

Notation Conventions

Hypertext Links

Blue underline is used to indicate a hypertext link within text. By clicking a passage of text with a blue underline, you are taken to the location described. For example:

This requirement is described under Backup DAM Volumes and Physical Disk Drives on page 3-2.

General Syntax Notation

This list summarizes the notation conventions for syntax presentation in this manual.

UPPERCASE LETTERS. Uppercase letters indicate keywords and reserved words. Type these items exactly as shown. Items not enclosed in brackets are required. For example:

```
MAXATTACH
```

lowercase italic letters. Lowercase italic letters indicate variable items that you supply. Items not enclosed in brackets are required. For example:

`file-name`

computer type. `Computer type` letters within text indicate C and HP NonStop Kernel Open System Services (OSS) keywords and reserved words. Type these items exactly as shown. Items not enclosed in brackets are required. For example:

```
myfile.c
```

italic computer type. *`Italic computer type`* letters within text indicate C and HP NonStop Kernel Open System Services (OSS) variable items that you supply. Items not enclosed in brackets are required. For example:

`pathname`

[ ] Brackets. Brackets enclose optional syntax items. For example:

```
TERM [\system-name.]$terminal-name
```

```
INT[ERRUPTS]
```

A group of items enclosed in brackets is a list from which you can choose one item or none. The items in the list can be arranged either vertically, with aligned brackets on each side of the list, or horizontally, enclosed in a pair of brackets and separated by vertical lines. For example:

```
FC [ num  ]
   [ -num ]
   [ text ]
```

```
K [ X | D ] address
```

{ } Braces. A group of items enclosed in braces is a list from which you are required to choose one item. The items in the list can be arranged either vertically, with aligned braces on each side of the list, or horizontally, enclosed in a pair of braces and separated by vertical lines. For example:

```
LISTOPENS PROCESS { $appl-mgr-name }
                  { $process-name  }
```

```
ALLOWSU { ON | OFF }
```

| Vertical Line. A vertical line separates alternatives in a horizontal list that is enclosed in brackets or braces. For example:

```
INSPECT { OFF | ON | SAVEABEND }
```

… Ellipsis. An ellipsis immediately following a pair of brackets or braces indicates that you can repeat the enclosed sequence of syntax items any number of times. For example:

```
M address [ , new-value ]…
```

```
[ - ] {0|1|2|3|4|5|6|7|8|9}…
```

An ellipsis immediately following a single syntax item indicates that you can repeat that syntax item any number of times. For example:

```
"s-char…"
```

Punctuation. Parentheses, commas, semicolons, and other symbols not previously described must be typed as shown. For example:

```
error := NEXTFILENAME ( file-name ) ;
```

```
LISTOPENS SU $process-name.#su-name
```

Quotation marks around a symbol such as a bracket or brace indicate the symbol is a required character that you must type as shown. For example:

```
"[" repetition-constant-list "]"
```

Item Spacing. Spaces shown between items are required unless one of the items is a punctuation symbol such as a parenthesis or a comma. For example:

```
CALL STEPMOM ( process-id ) ;
```

If there is no space between two items, spaces are not permitted. In this example, no spaces are permitted between the period and any other items:

```
$process-name.#su-name
```

Line Spacing. If the syntax of a command is too long to fit on a single line, each continuation line is indented three spaces and is separated from the preceding line by a blank line. This spacing distinguishes items in a continuation line from items in a vertical list of selections. For example:

```
ALTER [ / OUT file-spec / ] LINE

   [ , attribute-spec ]…
```

!i and !o. In procedure calls, the !i notation follows an input parameter (one that passes data to the called procedure); the !o notation follows an output parameter (one that returns data to the calling program). For example:

```
CALL CHECKRESIZESEGMENT (  segment-id                    !i
                         , error        ) ;              !o
```

!i,o. In procedure calls, the !i,o notation follows an input/output parameter (one that both passes data to the called procedure and returns data to the calling program). For example:

```
error := COMPRESSEDIT ( filenum ) ;                      !i,o
```

!i:i. In procedure calls, the !i:i notation follows an input string parameter that has a corresponding parameter specifying the length of the string in bytes. For example:

```
error := FILENAME_COMPARE_ (  filename1:length           !i:i
                            , filename2:length ) ;       !i:i
```

!o:i. In procedure calls, the !o:i notation follows an output buffer parameter that has a corresponding input parameter specifying the maximum length of the output buffer in bytes. For example:

```
error := FILE_GETINFO_ (  filenum                        !i
                        , [ filename:maxlen ] ) ;        !o:i
```

Notation for Messages

This list summarizes the notation conventions for the presentation of displayed messages in this manual.

Bold Text. Bold text in an example indicates user input typed at the terminal. For example:

```
ENTER RUN CODE

?123

CODE RECEIVED:      123.00
```

The user must press the Return key after typing the input.

Nonitalic text. Nonitalic letters, numbers, and punctuation indicate text that is displayed or returned exactly as shown. For example:

```
Backup Up.
```

lowercase italic letters. Lowercase italic letters indicate variable items whose values are displayed or returned. For example:

p-register

process-name

[ ] Brackets. Brackets enclose items that are sometimes, but not always, displayed. For example:

```
Event number = number [ Subject = first-subject-value ]
```

A group of items enclosed in brackets is a list of all possible items that can be displayed, of which one or none might actually be displayed. The items in the list can be arranged either vertically, with aligned brackets on each side of the list, or horizontally, enclosed in a pair of brackets and separated by vertical lines. For example:

```
proc-name trapped [ in SQL | in SQL file system ]
```

{ } Braces. A group of items enclosed in braces is a list of all possible items that can be displayed, of which one is actually displayed. The items in the list can be arranged either vertically, with aligned braces on each side of the list, or horizontally, enclosed in a pair of braces and separated by vertical lines. For example:

```
obj-type obj-name state changed to state, caused by
{ Object | Operator | Service }
```

```
process-name State changed from old-objstate to objstate
{ Operator Request. }
{ Unknown.          }
```

| Vertical Line. A vertical line separates alternatives in a horizontal list that is enclosed in brackets or braces. For example:

```
Transfer status: { OK | Failed }
```

% Percent Sign. A percent sign precedes a number that is not in decimal notation. The % notation precedes an octal number. The %B notation precedes a binary number. The %H notation precedes a hexadecimal number. For example:

```
%005400
```

```
%B101111
```

```
%H2F
```

```
P=%p-register E=%e-register
```

Notation for Management Programming Interfaces

This list summarizes the notation conventions used in the boxed descriptions of programmatic commands, event messages, and error lists in this manual.

UPPERCASE LETTERS. Uppercase letters indicate names from definition files. Type these names exactly as shown. For example:

```
ZCOM-TKN-SUBJ-SERV
```

lowercase letters. Words in lowercase letters are words that are part of the notation, including Data Definition Language (DDL) keywords. For example:

```
token-type
```

!r. The !r notation following a token or field name indicates that the token or field is required. For example:

```
ZCOM-TKN-OBJNAME      token-type ZSPI-TYP-STRING.          !r
```

!o. The !o notation following a token or field name indicates that the token or field is optional. For example:

```
ZSPI-TKN-MANAGER      token-type ZSPI-TYP-FNAME32.         !o
```

Change Bar Notation

Change bars are used to indicate substantive differences between this manual and its preceding version. Change bars are vertical rules placed in the right margin of changed portions of text, figures, tables, examples, and so on. Change bars highlight new or revised information. For example:

> The message types specified in the REPORT clause are different in the COBOL85 environment and the Common Run-Time Environment (CRE).
>
> The CRE has many new message types and some new message type codes for old message types. In the CRE, the message type SYSTEM includes all messages except LOGICAL-CLOSE and LOGICAL-OPEN.

1 Introduction

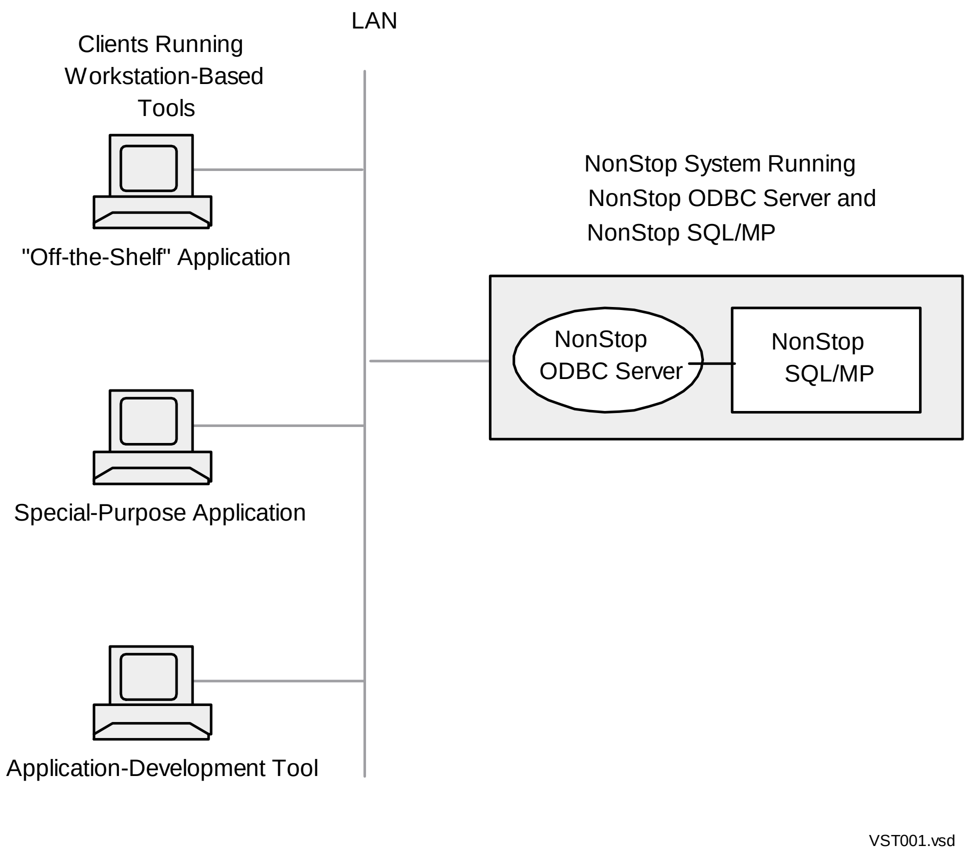
This section introduces the HP NonStop ODBC Server. It covers the following topics:

Purpose of the NonStop ODBC Server

The NonStop ODBC Server allows programs developed for Microsoft's ODBC (Open Database Connectivity) or for Microsoft's or Sybase's SQL Server to access HP NonStop SQL/MP data. ODBC is an application program interface (API)—specifically, a call-level interface (CLI)—that runs on personal computers under the Windows environment. SQL Server is part of a client/server system that runs on PCs and UNIX workstations. The NonStop ODBC Server runs on HP NonStop systems and allows a NonStop system to act as a server.

Many workstation-based applications have been written to use ODBC or SQL Server; applications can be commercial programs used just as they are shipped, or special-purpose programs developed for a single task, or application-development tools used to create other applications for database management. These applications can access NonStop SQL/MP data by using the NonStop ODBC Server, as shown in Figure 1-1.

Figure 1-1. Workstation-Based Tools Accessing NonStop SQL/MP Using the NonStop ODBC Server


LAN
Clients Running Workstation-Based Tools
"Off-the-Shelf" Application
Special-Purpose Application
Application-Development Tool
NonStop System Running NonStop ODBC Server and NonStop SQL/MP
NonStop ODBC Server
NonStop SQL/MP
VST001.vsd

Application Interface

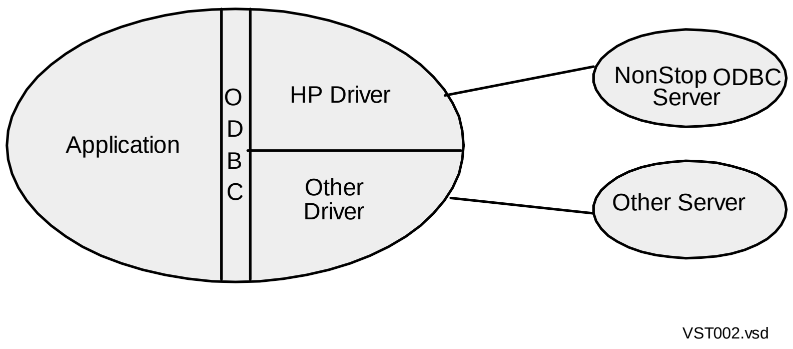
The NonStop ODBC Server allows NonStop SQL/MP access to applications written to run using the ODBC API as well as those written to run with SQL Server. ODBC is the application program interface for ODBC applications. The API for SQL Server applications is called DB-LIBRARY (DBLIB). The following subsections describe the differences and similarities in the two types of interfaces.

ODBC Interface

ODBC is Microsoft’s interface for accessing data in a heterogeneous environment of database management systems. It provides a single interface for accessing data stored in a variety of proprietary personal computer, minicomputer, and mainframe databases. ODBC provides a universal data access interface to ease the need for independent software vendors and corporate developers to use multiple APIs. By using ODBC and ODBC drivers, an application can access data from multiple, diverse databases.

Figure 1-2 shows the relationships among an application, the ODBC interface, ODBC drivers, and the servers with which they relate.

Figure 1-2. ODBC Interface



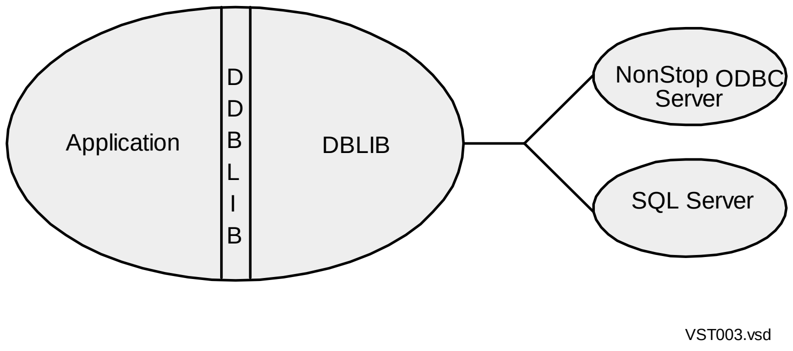
A number of different ODBC drivers (code that enables connection to a server), each created for a different server, are supplied by different manufacturers. When the application requests a connection to a specific server (by name), the ODBC Driver Manager (supplied by Microsoft as part of Windows or NT) loads the appropriate driver as a Dynamic Link Library (DLL); the driver then connects to the server.

Most drivers support the ODBC CORE SQL "dialect" of the SQL language; they all support their own host dialects of SQL. The HP NonStop ODBC/MP Driver supports the CORE SQL dialect.

DB-LIBRARY Interface

Figure 1-3 shows the relationships among an application using DB-LIBRARY interface, the server connection code, and the server.

Figure 1-3. DB-LIBRARY Interface



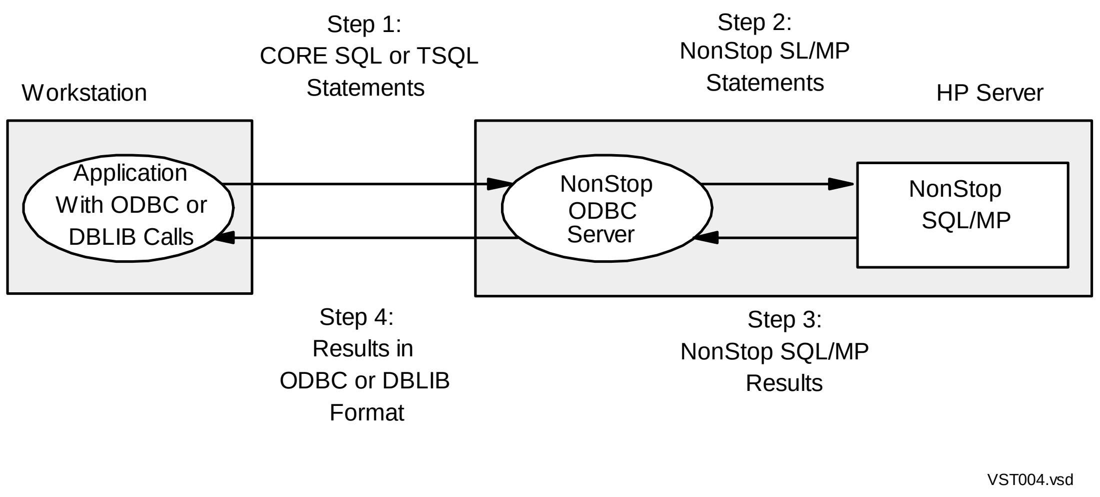
The DB-LIBRARY interface uses the Transact-SQL dialect of the SQL language exclusively. An application of this type can communicate with either the NonStop ODBC Server or the SQL Server.

How the NonStop ODBC Server Works

A workstation-based application program written to run with ODBC contains calls to the ODBC API that the NonStop ODBC/MP Driver packages for transmission to the NonStop ODBC Server. An application written to run with SQL Server contains calls to the DBLIB API to perform similar functions.

Figure 1-4 illustrates how the NonStop ODBC Server accepts SQL statements, submits them to NonStop SQL/MP, and returns the output to the client application.

Figure 1-4. Relationships Among Applications, the NonStop ODBC Server, and NonStop SQL/MP



As illustrated in this figure, an SQL statement (CORE SQL for ODBC, Transact-SQL for DBLIB) is submitted to NonStop SQL/MP as follows:

1. An application issues an ODBC or DB-LIBRARY call that sends an SQL statement to the NonStop ODBC Server.
2. The NonStop ODBC Server converts the client SQL statement to a NonStop SQL/MP statement and sends the statement on to NonStop SQL/MP.
3. NonStop SQL/MP processes the statement and returns NonStop SQL/MP data (or a diagnostic message) to the NonStop ODBC Server.
4. The NonStop ODBC Server converts the NonStop SQL/MP output to ODBC or SQL Server format and sends the data to the application program.

Finally, the application retrieves the data using ODBC or DB-LIBRARY calls.

Users of the NonStop ODBC Server

The people who use the NonStop ODBC Server are system managers, administrators, and end users.

Table 1-1 lists the types of NonStop ODBC Server users and describes their primary interaction with the NonStop ODBC Server.

Table 1-1. NonStop ODBC Server Users

Type	Primary Activity
System manager	Installs and manages the NonStop ODBC Server
Database administrator	Configures and manages the NonStop ODBC Server, manages customized catalogs, and manages NonStop SQL/MP users and NonStop SQL/MP data
Client administrator	Configures and manages the client applications
Client user	Writes the client ODBC or DBLIB applications
End user	Uses client applications that access NonStop SQL/MP data by means of the NonStop ODBC Server.

Hardware and Software Requirements

To use the NonStop ODBC Server, you need a NonStop system, a Windows, DOS, or UNIX workstation, and networking hardware and software. The following subsections describe the hardware and software requirements for a NonStop system and for a DOS/Windows or UNIX workstation.

NonStop Server Requirements

Table 1-2 lists the hardware and software needed on a HP NonStop server to run the NonStop ODBC Server.

Table 1-2. Hardware and Software Requirements for the NonStop Server

Requirement	Description
Computer	NonStop server running a D-series or G-series version of the HP NonStop Kernel operating system
Operating system	NonStop Kernel, D30 version or later
NonStop SQL/MP	NonStop SQL/MP, D30 version or later
	Installing the NonStop SQL/MP sample database is also recommended.
Network software	TCP/IP
Network hardware	LAN controller

DOS/Windows Workstation Requirements

Table 1-3 lists the hardware and software required to run the NonStop ODBC Server with a Windows workstation using ODBC; Table 1-4 lists the hardware and software required to run the NonStop ODBC Server with a DOS/Windows workstation using DB-LIBRARY.

Table 1-3. Hardware and Software Requirements for Workstations Using ODBC With Windows 95, Windows 98,Windows 2000, Windows XP or Windows NT

Requirement	Description
Computer	Intel-based workstation with a minimum 80486 microprocessor
Memory	At least 32 MB
Disk space	At least 40 MB free
Operating system	Microsoft Windows NT 3.5x (or later), Microsoft Windows 95, Windows 98, Windows 2000, or Windows XP Professional.
Network hardware	3COM (Etherlink III), Novell (NE2000), or UB (NIUpc or EOTP) LAN board
Network software	TCP/IP: Microsoft Winsock 1.1 or later

Note. For Windows NT, Windows 95, Windows 98, Windows 2000, or Windows XP Professional, HP provides only a 32-bit driver. Existing 16-bit applications can continue to operate with the 32-bit driver.

Table 1-4. Hardware and Software Requirements for Workstations Using DBLIB

Requirement	Description	
Computer	Intel-based workstation with a minimum 80386 microprocessor	
Memory	At least 8 MB	
Disk space	At least 40 MB free	
Operating system	DOS 3.3, Windows 3.1, or OS/2.0 (or later versions)	
Network hardware	3COM (3C503 or 3C507), Novell (NE2000), or UB (NIUpc or EOTP) LAN board	
Network software	TCP/IP:	Microsoft, Novell, or FTP TCP/IP
DB connectivity software	Sybase Open Client/C *	
	Microsoft TCP/IP:	Net-Lib MS TCP
	Novell TCP/IP:	Net-Lib Nov LWP TCP
	FTP TCP/IP:	Net-Lib FTP PC/TCP

* Database connectivity software is provided by Sybase and must be ordered to match the operating system and network software you are using.

Note. The NonStop ODBC server negotiates to a version 4.6 (or less) driver for clients using DBLIB. Therefore, a DBLIB that requires a driver version greater than 4.6 is not supported.

UNIX Workstation Requirements

Table 1-5 lists the hardware and software required to run the NonStop ODBC Server with a UNIX workstation.

Table 1-5. Hardware and Software Requirements for UNIX Workstations

Requirement	Description
Computer	UNIX workstation
Memory	At least 32 MB
Disk space	At least 70 MB free
Operating system	Sun 4.1.1 or later, HP-UX, or other appropriate UNIX OS
Network hardware	Ethernet board
Network software	TCP/IP
DB connectivity software	Sybase Open Client/C (for the relevant operating system)

SQL Language Support

The NonStop ODBC Server supports the ODBC CORE SQL dialect of the SQL language or the SQL Server Transact-SQL dialect (or the NonStop SQL/MP syntax using pass-through commands). Clients can access NonStop SQL/MP databases using whichever dialect their application implements.

In an environment shared by both CORE SQL applications and Transact-SQL applications, however, it is suggested that users adhere to the minimum attributes offered by either dialect (for example, using names not more than 32 characters in length, the Transact-SQL maximum) to promote the optimum sharing of data access among all clients in the environment.

Supported and Unsupported ODBC Features

There are three levels of SQL syntax recommended for use with ODBC. They are:

Minimum set	A set of SQL statements suitable for access to a flat file.
Core set	SQL syntax based on the X/OPEN and SQL Access Group SQL CAE (Common Applications Environment) specification. The core set is a superset of the minimum set.
Extended set	SQL syntax for datetime usage, outer joins, scalar functions, and stored-procedure calls. The extended set is a superset of the minimum and core sets.

The NonStop ODBC Server supports all of the core-level statements (CORE SQL) and most of the extended syntax. Table 1-6 summarizes the ODBC SQL features and indicates whether the NonStop ODBC Server supports the feature and where to find further information on the feature. For a complete list of NonStop ODBC Server support, see Appendix A, Summary of Support for ODBC Features.

Table 1-6. NonStop ODBC Server Support of ODBC Features (page 1 of 2)

Feature	Description of NonStop ODBC Server Support		More Information
Aggregate functions	All aggregate functions are supported.		Aggregates on page 3-27
Database names	Database names are not defined in ODBC CORE SQL, but the NonStop ODBC Server supports object references of the format *database.owner.object*. A database name is a user-defined name, up to 60 characters long; the NonStop ODBC Server maps it internally to a NonStop SQL/MP Catalog (disk subvolume).		Names on page 3-3
Data types	All CORE SQL data types, at both Core level and Extended level, are supported.		Data Types on page 3-8
Expressions and operators	All CORE SQL operators are supported in expressions.		Expressions and Operators on page 3-25
Functions	The following functions are supported:		Functions on page 3-14
	UCASE	(Text function)	
	EXP MOD PI	(Numeric functions)	
	CURDATE CURTIME DAYOFMONTH DAYOFWEEK HOUR MINUTE MONTH NOW SECOND YEAR	(Datetime functions)	
	DATABASE USER	(System functions)	
	CONVERT	(Explicit data conversion)	

Table 1-6. NonStop ODBC Server Support of ODBC Features (page 2 of 2)

Feature	Description of NonStop ODBC Server Support	More Information
Names	The following types of names follow ODBC naming rules: column name correlation name database name index name procedure name table name username view name All names have special considerations.	Database names on page 1-8 Usernames in this table Names on page 3-3
Usernames	Usernames are supported as qualifiers to table/view/index names. A username is a user-defined name up to 60 characters long.	Names on page 3-3
CORE SQL statements	Data Definition Language (DDL) statements, such as CREATE TABLE and DROP INDEX, are supported. Data Manipulation Language (DML) statements, such as SELECT and UPDATE, are supported. The CALL statement, for execution of stored procedures, is supported. GRANT and REVOKE statements are recognized, but their actions are not performed.	Section 3, CORE SQL Language
Stored procedures	Execution of stored procedures is supported, but the procedures must be created in the Pathway environment.	Section 5, Stored Procedures

Supported and Unsupported SQL Server Features

Not all SQL Server features are supported by the NonStop ODBC Server. Table 1-7 summarizes SQL Server features and indicates whether the NonStop ODBC Server supports the feature and where to find further information on the feature. For a complete list of NonStop ODBC Server support, see Appendix B, Summary of Support for SQL Server Features.

Table 1-7. NonStop ODBC Server Support of SQL Server Features (page 1 of 4)

Feature	Description of NonStop ODBC Server Support	More Information
Aggregate functions	All aggregate functions are supported.	Aggregates on page 4-31
Alias usernames	Alias usernames are supported.	ADD ALIAS on page 7-42
COMPUTE BY clause	Not supported	SELECT on page 4-82
Database names	Database names are logical 60-character identifiers. They have a default format of: *node_volume_subvolume* For example: test_vol3_persnl The logical database name maps to a Guardian node, volume, and subvolume and must follow Guardian naming rules.	Names on page 4-5
Data types	The following data types are supported: BIT CHAR DATETIME FLOAT INT MONEY SMALLINT SYSNAME TEXT TINYINT VARCHAR	Data Types on page 4-12
Dateparts	The following dateparts are supported: year month day hour minute second millisecond	Data Types on page 4-12

Table 1-7. NonStop ODBC Server Support of SQL Server Features (page 2 of 4)

Feature	Description of NonStop ODBC Server Support	More Information
Expressions and operators	All operators *except* the following are supported in expressions: & | ^ ~ *= =* "+"	Expressions and Operators on page 4-29
Functions	The following functions are supported: DATEADD DATEDIFF DATEPART GETDATE EXP PI POWER UPPER DB_NAME SUSER_NAME USER_ID USER_NAME CONVERT	Functions on page 4-17
Global variables	The following global variables are supported: @@connections @@error @@max_connections @@textsize @@trancount	Variables on page 4-26
Multiple opens	Multiple concurrent database connections are supported.	Section 2, Architecture Overview

Table 1-7. NonStop ODBC Server Support of SQL Server Features (page 3 of 4)

Feature	Description of NonStop ODBC Server Support	More Information
Names	The following types of names follow SQL Server naming rules: column name correlation name index name table name variable name view name Database and owner names, however, must follow special rules.	Database names on page 1-8 Owner names on page 1-12 in this table Names on page 4-5
NULL values	NULL values are supported. When used with the ORDER BY clause, however, NULL values come after all others (in SQL Server, NULL values come before all others).	NULL Values on page 4-34
Owner names	An owner name is a logical username associated with a Guardian logon ID.	Names on page 4-5
Rules	Rules are *not* supported. NonStop SQL/MP has a similar feature, however, called a constraint.	Not described in this manual.
System procedures	*Not* supported; however, alternatives are available for some system procedures.	System Procedures on page B-17
System tables	The following system tables are supported: syscolumns sysindexes sysobjects sysprotects systypes sysusers sysdatabases sysmessages	System Tables on page B-20

Table 1-7. NonStop ODBC Server Support of SQL Server Features (page 4 of 4)

Feature	Description of NonStop ODBC Server Support	More Information
Transact-SQL statements	Data Definition Language (DDL) statements, such as CREATE DATABASE and DROP TABLE, are supported.	Section 4, Transact-SQL Language
	Data Manipulation Language (DML) statements, such as SELECT and UPDATE, are supported.	
	Transaction management statements, such as BEGIN TRANSACTION and COMMIT TRANSACTION, are supported.	
	Control-of-flow statements, such as IF...ELSE, WHILE, and GOTO, are *not* supported.	
	The CREATE and DROP statements for defaults, procedures, rules, and triggers are *not* supported.	
	GRANT and REVOKE statements are accepted, but their actions are not performed.	
Stored procedures	Execution of stored procedures is supported, but the procedures must be created in the Pathway environment.	Section 5, Stored Procedures
Triggers	Triggers are *not* supported.	Not described in this manual

2 Architecture Overview

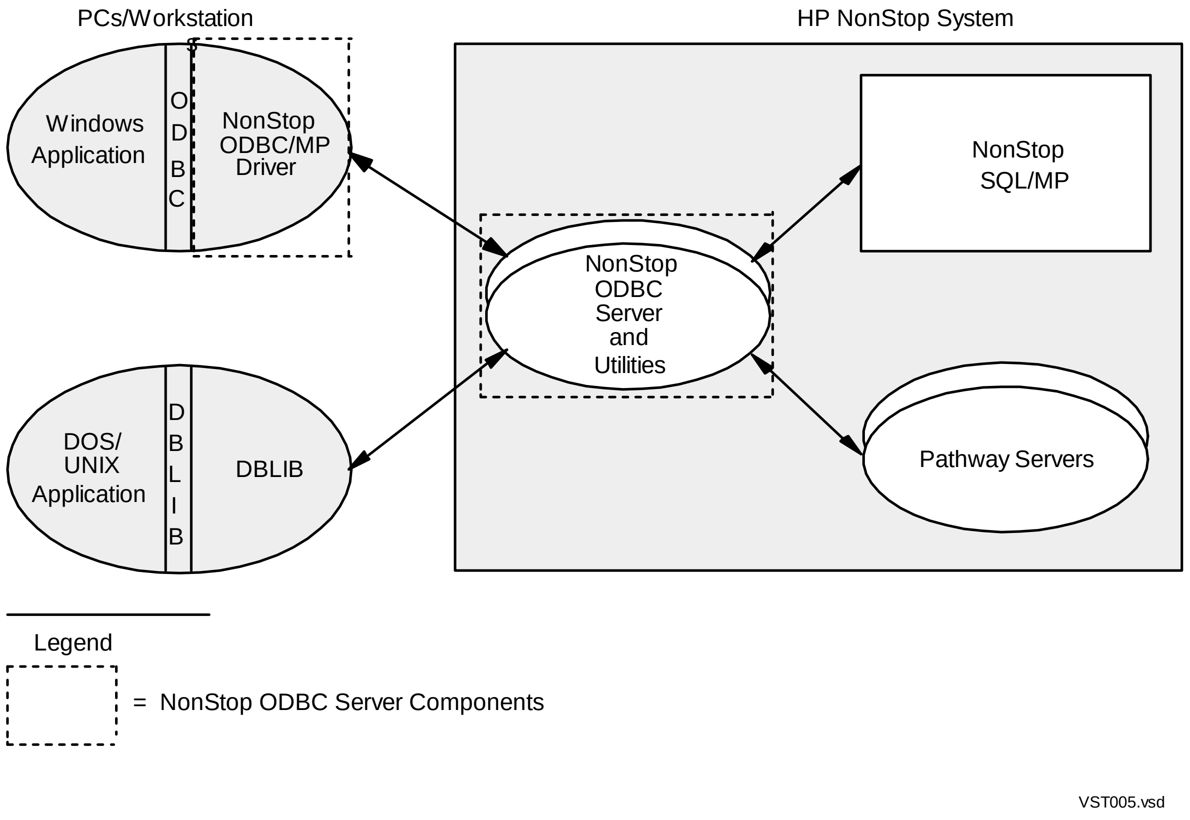
This section gives a high-level architectural overview of the HP NonStop ODBC Server, showing how the components of the NonStop ODBC Server interact to provide connectivity to HP NonStop SQL/MP databases for client applications that issue SQL statements from either Microsoft's ODBC interface or SQL Server's DBLIB interface.

This section covers the following topics:

- Background information, including client/server operations and SQL gateways
- Client-side view of the NonStop ODBC Server, including types of clients, databases and datasources, and client connections
- Server-side view of the NonStop ODBC Server, including the following:
    - Major components of the NonStop ODBC Server
    - Message flows
    - Objects and relationships
    - Name mapping
    - NonStop ODBC Server catalogs
    - NonStop ODBC server (the server process within the NonStop ODBC Server)
    - SQL communication subsystem (SCS)
    - NonStop ODBC Server utilities (NOSCOM and NOSUTIL)
    - NonStop ODBC Server execution

Figure 2-1 shows the most simplified view of the NonStop ODBC Server architecture. The NonStop ODBC Server consists of processes on the HP server that translate between the application SQL environment and the NonStop SQL/MP environment. Some of these processes are utilities and others are the run-time translators. There is also a facility to support the execution of Stored Procedures by issuing a ServerClass_Send to a Pathway server class program. The NonStop ODBC Server also includes a HP driver for ODBC clients that communicates with the server.

Figure 2-1. NonStop ODBC Server Generalized Architecture



Following subsections cover background information, the client-side view, and the server-side view. The client-side view describes what the application deals with (connecting, sessions, databases, SQL dialect, configuration of options) and the server-side view describes the HP server aspects (process structure, message flow, translation schemes, configuration, and management).

The NonStop ODBC Server has many types of users; it is useful to identify them, as each type sees the architecture differently.

Client user	The person who writes or uses the client DBLIB or ODBC application. This user is aware of connecting to the database and using the SQL database objects.
Client administrator	The person who configures and manages the client applications. This user is aware of network communication and configuring the HP NonStop ODBC/MP driver, but does not necessarily know much about the HP side.
HP system administrator	The person who configures and manages the NonStop ODBC Server components on the HP side. For this discussion, this category also includes administrative jobs for NonStop SQL/MP, Pathway, and the operating system.

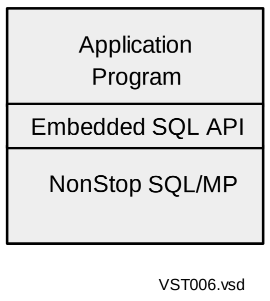
Background

The following gives background information on the topics of client/server operations, gateways, and NonStop ODBC Server terminology.

Client/Server Applications

An application program on a NonStop server can use the services of NonStop SQL/MP by means of an embedded SQL interface. In this type of interface, the NonStop SQL/MP statements are written in the source program intermixed with the regular statements of the programming language (COBOL or C, for example). A compiler translates these embedded statements into calls to the NonStop SQL/MP run-time routines in the system library. The run-time routines use the SQL file system to access NonStop SQL/MP tables and other objects.

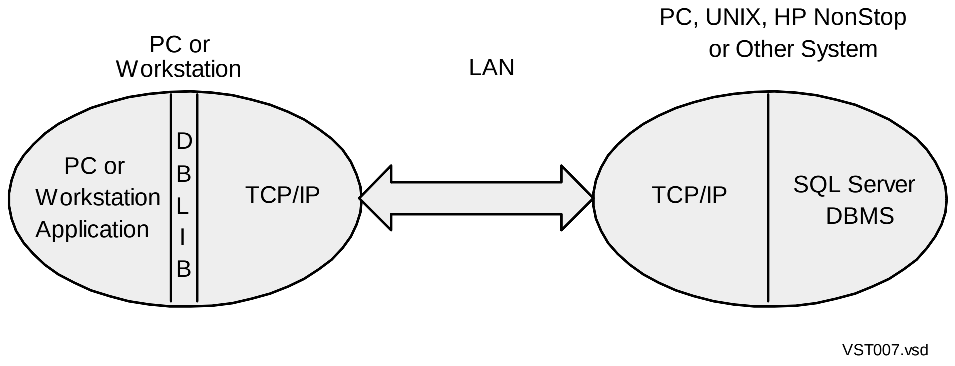
Figure 2-2. Application Program Interface for NonStop SQL/MP on a NonStop Server



Embedded SQL is the application program interface (API) for NonStop SQL/MP on NonStop servers. The API supports the valid syntax of SQL statements and the valid sequences in which they can be executed. The NonStop SQL programmer also has knowledge of other issues such as the Guardian ID of the application process, the default subvolume, security, TS/MP transactions, and the format of the NonStop SQL/MP catalog tables.

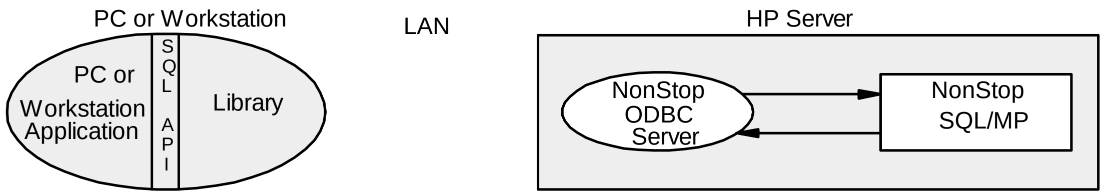
Most DBMS vendors have a different kind of implementation of their SQL product. The application is typically on a PC or workstation and it makes explicit calls to C functions in a library. The library connects to a database server program, usually running on another machine, and sends it the SQL to be processed. In the case of the Microsoft/Sybase product, the client library is called DBLIB, and the server is called SQL Server.

Figure 2-3. Application Program Interface (API) for SQL Server


PC or Workstation
LAN
PC, UNIX, HP NonStop or Other System
PC or Workstation Application
DBLIB
TCP/IP
TCP/IP
SQL Server DBMS
VST007.vsd

The NonStop ODBC Server is a HP product that provides connectivity between certain SQL client applications (actually between certain SQL APIs) and NonStop SQL/MP.

Figure 2-4. NonStop ODBC Server in a Client/Server Environment

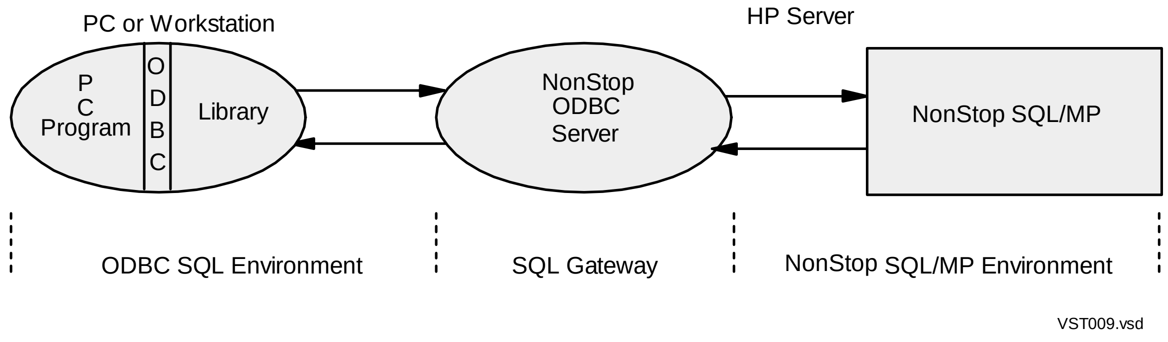


The NonStop ODBC Server supports both DBLIB and ODBC clients. There are other products on the HP server to support DAL (Data Access Language) and Oracle clients. The goal of all these products is to promote applications that can use NonStop SQL/MP on the NonStop server. The current suite of products covers the most popular client SQL APIs.

SQL Gateways

The NonStop ODBC Server can be thought of as an “SQL gateway” in that it connects the client’s SQL environment (DBLIB or ODBC) with the HP SQL environment (NonStop SQL/MP).

Figure 2-5. The NonStop ODBC Server as a SQL Gateway



The function of a gateway is to translate all the objects in the SQL environment (SQL syntax, data types, error codes, SQL catalog structures, and so on) between the two sides.

The term “mapping” is used to mean the mechanism of translating an object on one side (for example, ODBC) to the corresponding object on the other side (NonStop SQL/MP). The NonStop ODBC Server employs several different types of mapping schemes.

A HP SQL gateway has two main requirements:

- The gateway should be transparent. The clients should not be aware of being connected to a different type of database server. This allows client applications to run unchanged.
- The gateway must allow clients to take advantage of major HP product features including high availability, scalability, and high performance.

These two requirements are sometimes in conflict when a special interface is needed to use HP product features. For example, to invoke special DML query options in NonStop SQL/MP, you must issue a NonStop SQL/MP CONTROL statement (which a DBLIB client or ODBC client does not recognize).

Mapping tasks are not always well-defined and simple. Often, there is a choice of how a client item is mapped to a NonStop SQL/MP item. An example is table naming, in which ODBC uses ANSI SQL-like three-part names and NonStop SQL/MP uses Enscribe file names. The NonStop ODBC Server chooses a fixed mapping scheme when that is the best choice and provides a configurable scheme when there is no single best choice.

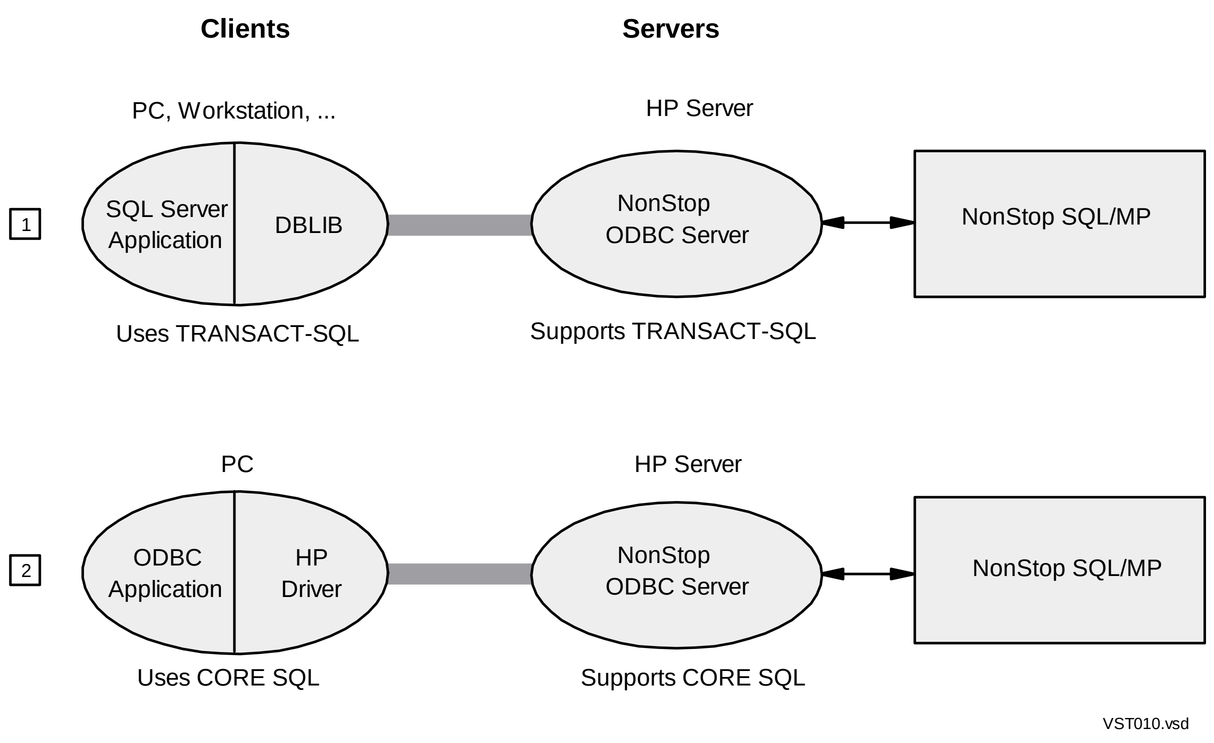
Client-Side View

The description of the client-side view of the NonStop ODBC Server architecture covers aspects visible to the client user and client administrator. This includes the SQL API usage as well as configuration.

Types of Clients

The NonStop ODBC Server supports two types of clients, as shown in Figure 2-6.

Figure 2-6. Two Types of Clients Supported by the NonStop ODBC Server



- Client 1 is a DBLIB/TSQL client, using the Transact-SQL (TSQL) dialect and making DBLIB function calls.
- Client 2 is an ODBC/CORE client; it uses ODBC CORE SQL and makes ODBC CLI function calls.

For ease of discussion, the following terms are also used in the descriptions of clients:

CORE client	Uses CORE SQL dialect (client 2)
TSQL client	Uses Transact-SQL dialect (client 1)
DBLIB client	Uses DBLIB API (client 1)
ODBC client	Uses ODBC API (client 2)

The NonStop ODBC Server architecture is able to service both client types in the same way, which is possible because the needs of both client types are usually the same. In the few cases where they differ, the NonStop ODBC Server provides a solution to accommodate them. For example, TSQL clients recognize names up to 32 characters long and CORE clients recognize longer names; to accommodate both, the NonStop ODBC Server supports 60-character names.

The advantages of supporting both clients with one architecture are simplicity of design and the opportunity to provide shared objects for the various clients. Note that clients of just one type are unaffected by NonStop ODBC Server extensions. For example, if a system has only TSQL clients, they do not recognize names over 32 characters long.

Databases and Datasources

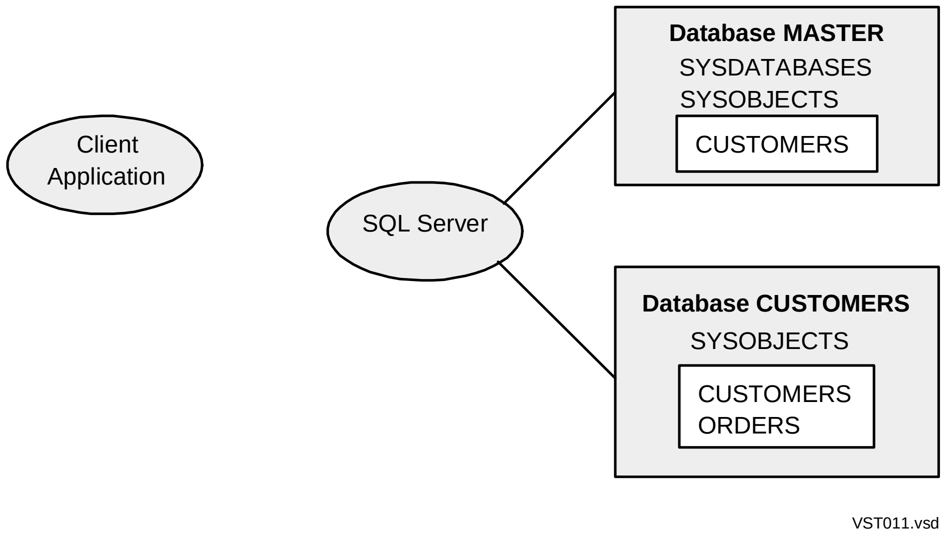
The following description of databases and datasources is based on the client-side view, but it also covers enough of the server-side view to indicate the NonStop ODBC Server implementation.

Both DBLIB and ODBC clients make a connection to where their data is maintained. In ODBC the connection is to a datasource, for DBLIB it is a server. Both are local logical identifiers that translate into a unique network server name and server location. The database view (what a database is, how it is named) is slightly different for each client type; descriptions of both follow.

Both types of clients treat a database as an object that has a name, has a catalog of SQL objects, maintains the SQL objects, can process SQL statements, and can handle transactions.

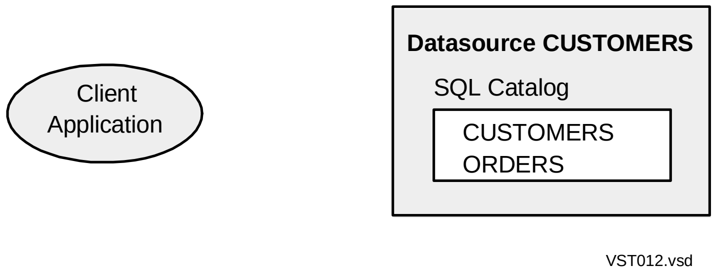
The DBLIB client actually connects to a particular SQL Server that manages a set of databases, as shown in Figure 2-7. The connection requests can specify the particular database to connect to (there are also commands to switch to another database). The set of databases always contains a special master database that catalogs the other databases as well as other information pertinent to the SQL Server. The master database corresponds to the NonStop SQL/MP system catalog.

Figure 2-7. Database View for a DBLIB Client



The ODBC client recognizes a set of datasources and can connect to one of them, as shown in Figure 2-8. A datasource is essentially a database. All the datasources are independent (there is no master database, as with DBLIB). Datasource is meant to be a vendor-neutral term, avoiding the need to distinguish among connecting to a database, connecting to a server that accesses a database, and connecting to anything else.

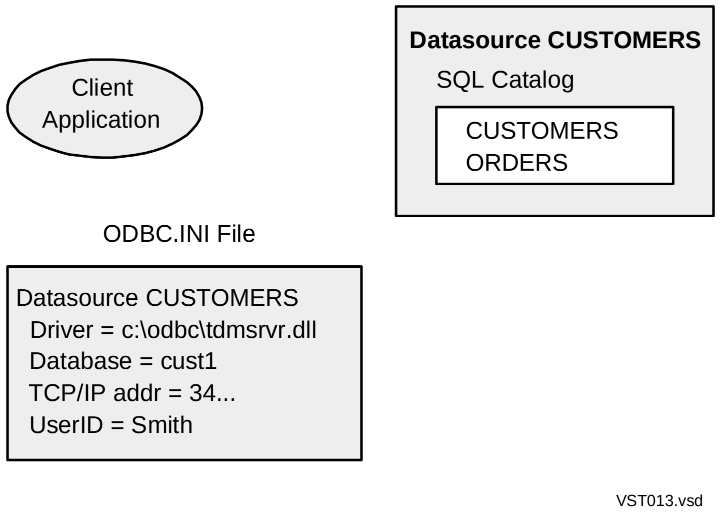
Figure 2-8. Database View for an ODBC Client



When an ODBC client connects to a datasource, the ODBC driver manager on the PC accesses a special ODBC.INI file, or the ODBC registry for Windows NT or Windows 95, to get information about the datasource. The file indicates which ODBC driver to load to access the datasource. The file can contain other information, such as the actual server to connect to (in the case of SQL Server), the database name to use, and so on.

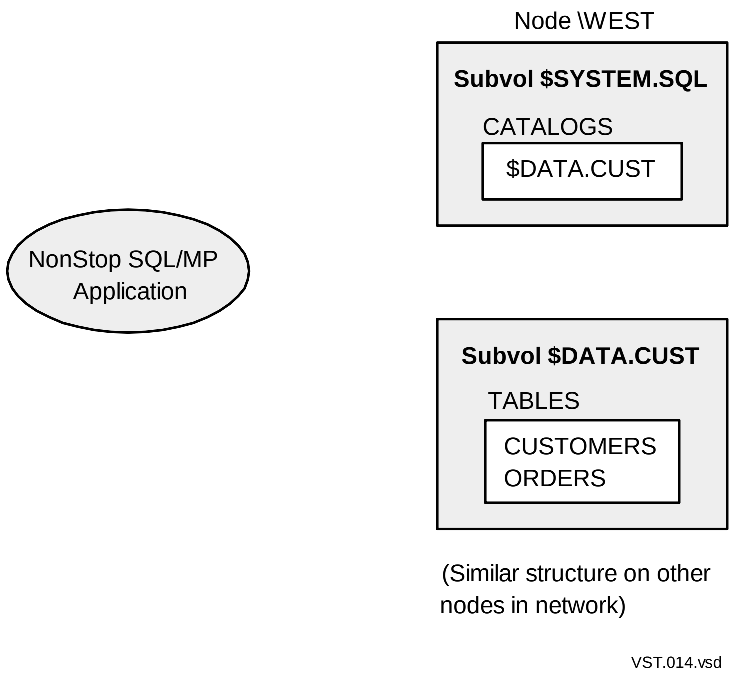
In Figure 2-9, if the client connected to the datasource named CUSTOMERS, the Nonstop ODBC/MP driver \ODBC\TDMSRVR.DLL would be loaded. The driver would then read the ODBC.INI file and determine that database CUST1 should be used with a user ID of SMITH. Microsoft documents this use of the ODBC.INI file, but other schemes are possible (the driver could communicate with a name server to bind the datasource name, and so on).

Figure 2-9. Interpreting an ODBC Datasource



The NonStop ODBC Server provides a database architecture compatible with both ODBC and DBLIB views, as well as being compatible with the NonStop SQL/MP architecture. With NonStop SQL/MP, each node has one NonStop SQL/MP system catalog and an arbitrary set of regular NonStop SQL/MP catalogs. A SQL object on a node can be registered in any catalog on that node. See Figure 2-10.

Figure 2-10. SQL Catalog View for a NonStop SQL/MP User

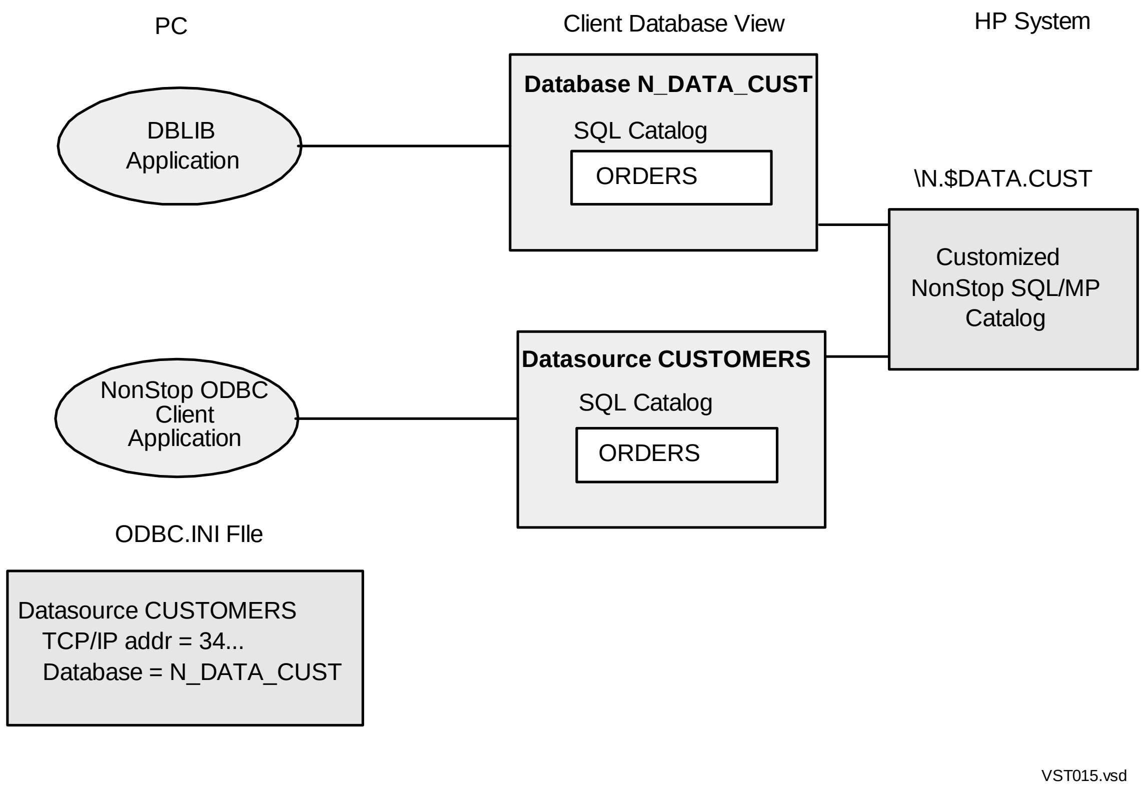


NonStop SQL/MP is distinctive in that objects such as SQL tables are named directly with names in Guardian format, such as \WEST.$DATA.CUST.ORDERS, that do not involve the NonStop SQL/MP catalog name or any concept of the “database” in which the table resides.

Figure 2-11 shows the complete database configuration:

- The HP server has a customized NonStop SQL/MP catalog on subvolume \N.$DATA.CUST.
- The DBLIB client accesses a database named N_DATA_CUST.
- The ODBC client accesses a datasource named CUSTOMERS and the ODBC.INI file binds this datasource to database N_DATA_CUST.
- Both clients use a NonStop ODBC server on a HP server and access the same NonStop SQL/MP data.

Figure 2-11. Database View for a NonStop ODBC Server Client

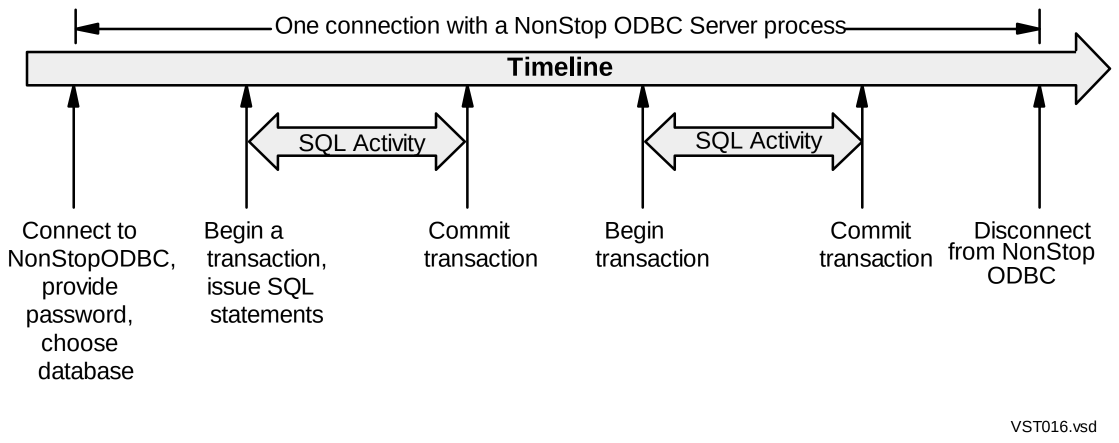


For the DBLIB client there is also a configuration file with the network address of the SQL Server. See the Microsoft or Sybase SQL Server documentation for details.

Connections

A client application uses the NonStop ODBC Server by connecting to a particular NonStop ODBC server, issuing SQL statements, and disconnecting, as shown in Figure 2-12.

Figure 2-12. A Client Connection With the NonStop ODBC Server

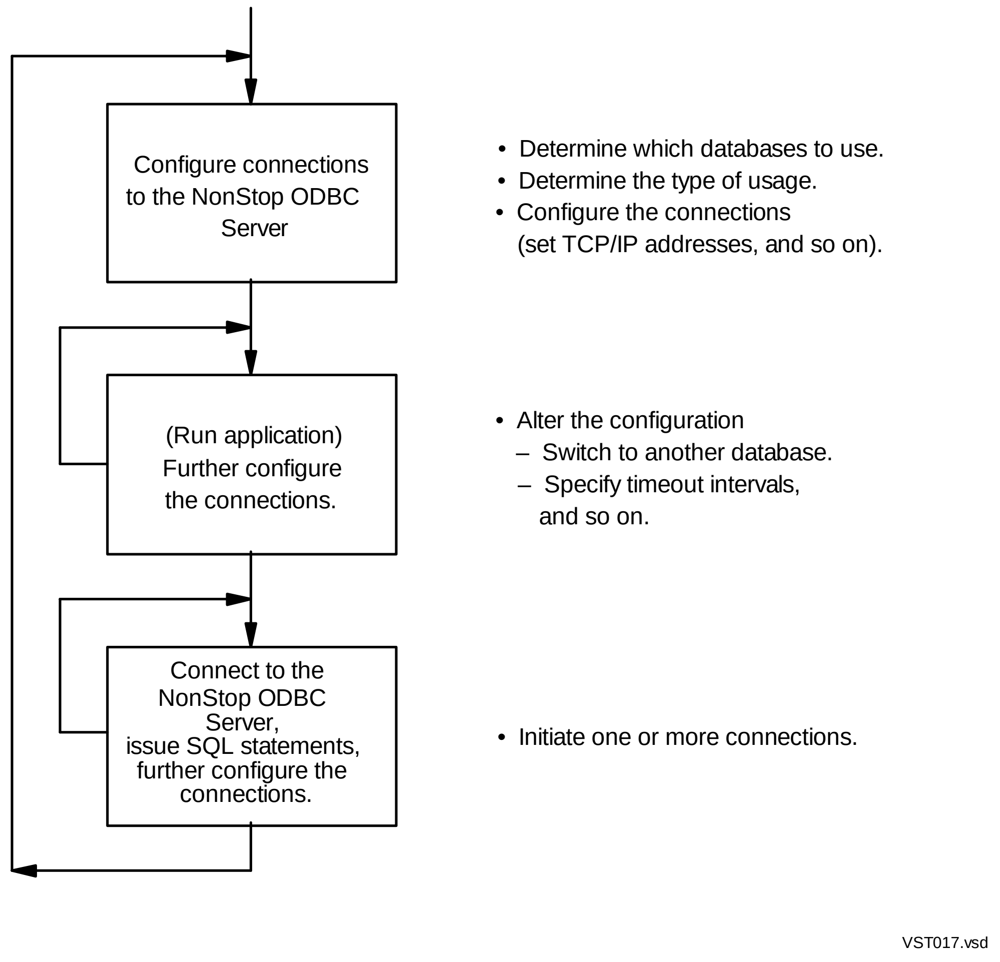

One connection with a NonStop ODBC Server process
Timeline
SQL Activity
SQL Activity
Connect to NonStopODBC, provide password, choose database
Begin a transaction, issue SQL statements
Commit transaction
Begin transaction
Commit transaction
Disconnect from NonStop ODBC
VST016.vsd

A connection is the model for client/server interactions with SQL database systems for both DBLIB and ODBC. The client application can do more, such as having several concurrent connections or automatically handling connect/disconnect for the end user, but the basic mechanism is the same.

The connecting process, in which the application identifies itself and provides a password and database name, is described in the next subsection.

In addition to an application program invoking connections, there are other client-side aspects of using the NonStop ODBC Server. With DBLIB and ODBC, the parameters of a connection (read/write or read-only, which database to use, and so on) can be configured before the application connects or before the application program is run. In addition, other parameters (maximum data returned, database to use, and so on) can be altered by the application during the connection. See Figure 2-13.

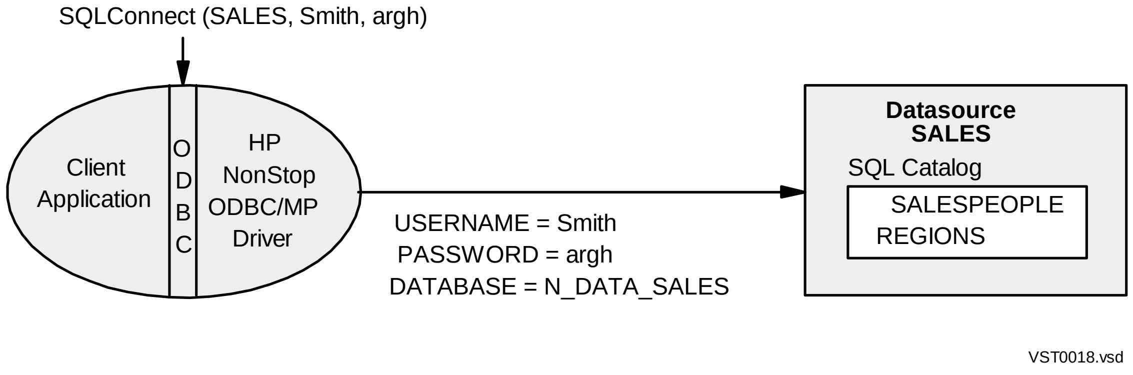
Figure 2-13. Client Activities in Using the NonStop ODBC Server


Configure connections to the NonStop ODBC Server
• Determine which databases to use.
• Determine the type of usage.
• Configure the connections (set TCP/IP addresses, and so on).
(Run application) Further configure the connections.
• Alter the configuration
– Switch to another database.
– Specify timeout intervals, and so on.
Connect to the NonStop ODBC Server, issue SQL statements, further configure the connections.
• Initiate one or more connections.
VST017.vsd

Connecting

During the connection process from the client side, a client connects to a specific database under a username. The client supplies a password so the database management system can authenticate the username. Usernames in the NonStop ODBC Server are mapped to Guardian usernames for use with NonStop SQL/MP.

Figure 2-14. NonStop ODBC Server Client Connection Process



In this figure, the client issues a SQLConnect ODBC CLI call that connects the client to the SALES datasource with username SMITH and password “argh.” For an ODBC client, the message flowing to the NonStop ODBC Server contains the username, password, and the database name (obtained from the ODBC.INI file for datasource=SALES). A DBLIB client would connect with the username, password, and database name.

In ODBC on Windows, several other variants are available to the clients: the user ID or password can be configured in the ODBC.INI file, or the driver can open a window and prompt the user for a username, password, or both. In both ODBC and DBLIB, the database name can alternatively be configured on the server side (based on the username in the user profile).

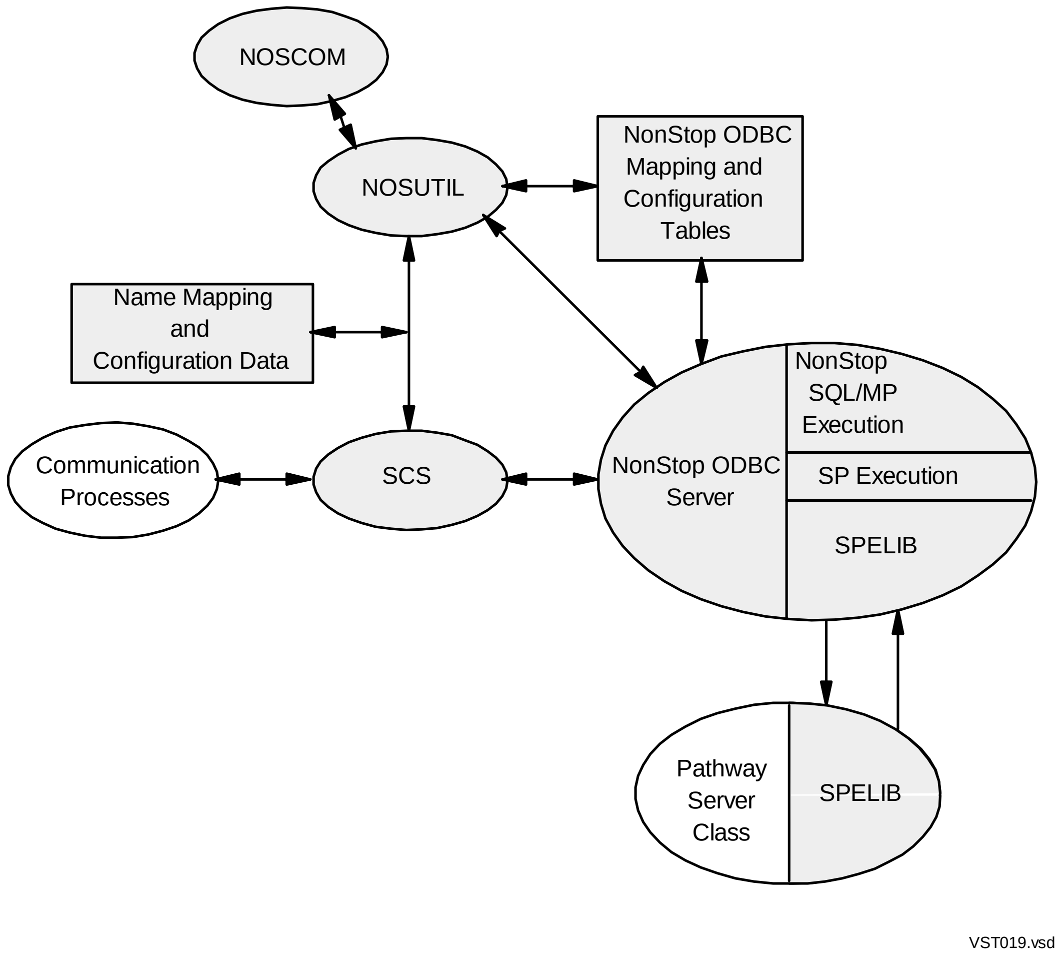
Server-Side View

The server-side view of the NonStop ODBC Server architecture describes aspects visible to the server administrator. This view includes the support of the NonStop ODBC Server objects on NonStop servers, process architecture, message flows, and configuration and management interfaces.

Major Components

The major components of the NonStop ODBC Server are shown in Figure 2-15. Unshaded forms are relevant components other than NonStop ODBC server entities; rectangles represent data.

Figure 2-15. Major Components of the NonStop ODBC Server on a NonStop Server



Other processes used by NonStop SQL/MP, such as the SQL Compiler, DP2, and TS/MP, are also involved.

A brief description of each component is presented here; expanded descriptions appear in this section from page 2-27 onwards.

- NonStop ODBC server

  This process performs the SQL gateway actions for a client. There is a separate NonStop ODBC server process for each client connection. The process receives SQL requests from a client, translates them to NonStop SQL/MP (or to a ServerClass_Send for a stored procedure execution), executes the translated statements, collects the results (or errors), translates data to the client's types, and replies to the client.

- SQL Communication Subsystem (SCS)

This subsystem handles all communication with the client workstation and manages the NonStop ODBC servers. To avoid costly process start-ups, SCS maintains server classes of previously started NonStop ODBC servers in an available list. Each client connection results in the creation of an association between the client network ID and an assigned available NonStop ODBC server. SCS passes the connection message to this server for authentication and subsequently passes all messages in both directions until the connection is terminated. SCS uses SCF for reporting run time performance data and EMS to report internal state changes as well as diagnostics.

- NOSUTIL

  This utility process performs several services. It is a server process initiated by any of the processes, NOSCOM, SCS or NonStop ODBC server, to customize NonStop SQL/MP catalogs for NonStop ODBC Server use and to manage NonStop ODBC Server mapping and configuration tables. NOSUTIL can also be called by a NonStop ODBC server to support create database and drop database commands. NOSUTIL is also called by SCS to acquire initial server class definitions and periodic updates to server class configurations. Additionally, NOSUTIL provides name-mapping service to SCS at connection time to determine which server class is to be used.

- NOSCOM

  HP client interface used to issue NonStop ODBC utility commands. It starts a NOSUTIL process and sends utility commands and receives result data or error messages. NOSCOM also provides the common CI commands of HELP, FC, HISTORY and LOG.

- SPELIB

  This call library is provided by the NonStop ODBC Server for Pathway server class programs that implement stored procedures. The functions in SPELIB are used to decode the stored procedure calls and to encode results to be returned.

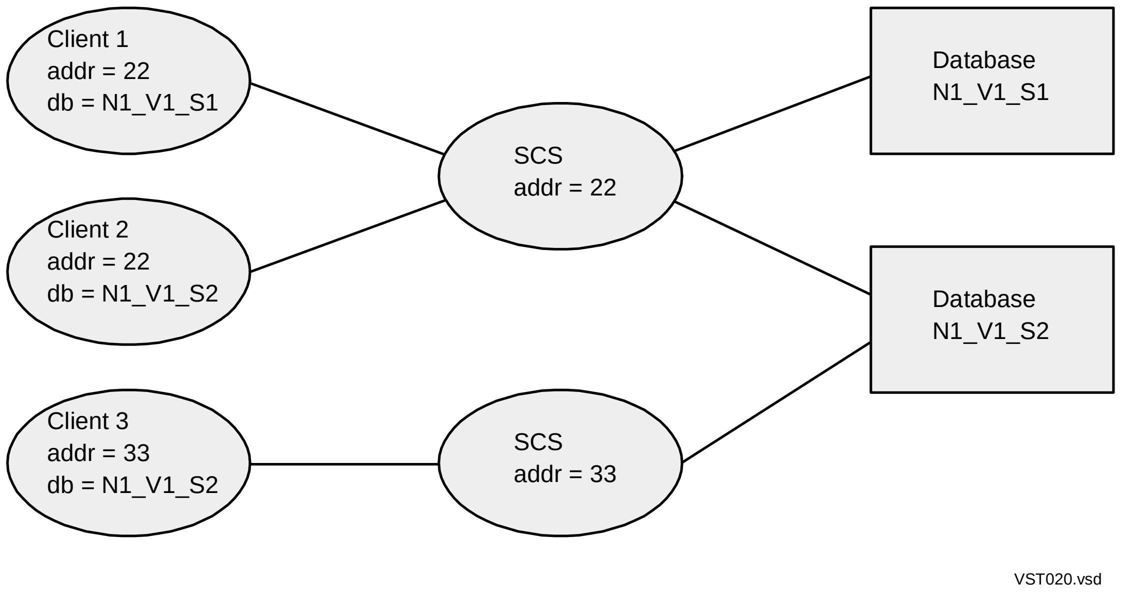
- Mapping and configuration data

  This data is stored in several NonStop ODBC catalog SQL tables. The NonStop ODBC Server maintains SQL tables to map client names to NonStop SQL/MP names and to help provide SQL catalog tables for ODBC and DBLIB clients. There are also SQL tables to hold server-side attributes of users. SCS is also driven by a SQL table that defines the server classes.

Each SCS is identified by a network address (such as an IP address) to clients. The client's ODBC.INI file or DBLIB configuration file is configured with the appropriate address (for DBLIB, SCS is like a SQL Server). You can think of one SCS, together with the components in Figure 2-15, as one NonStop ODBC server system.

A NonStop server might contain multiple SCS processes. Each is free to access any of the customized catalogs registered in its designated system catalog mapping table, ZNSDB. Figure 2-16 shows three clients connecting to NonStop ODBC Server databases through two SCS processes. Clients 1 and 2 use the top SCS and are connected to two different databases. Clients 2 and 3 use different SCS processes and are both connected to database N1_V2_S2. Note that in this example the clients all access the same set of databases by the same names, without regard to the SCS used (of course, if they are ODBC clients, they could access different datasource names that lead to the same database).

Figure 2-16. Client Connections Through SCS

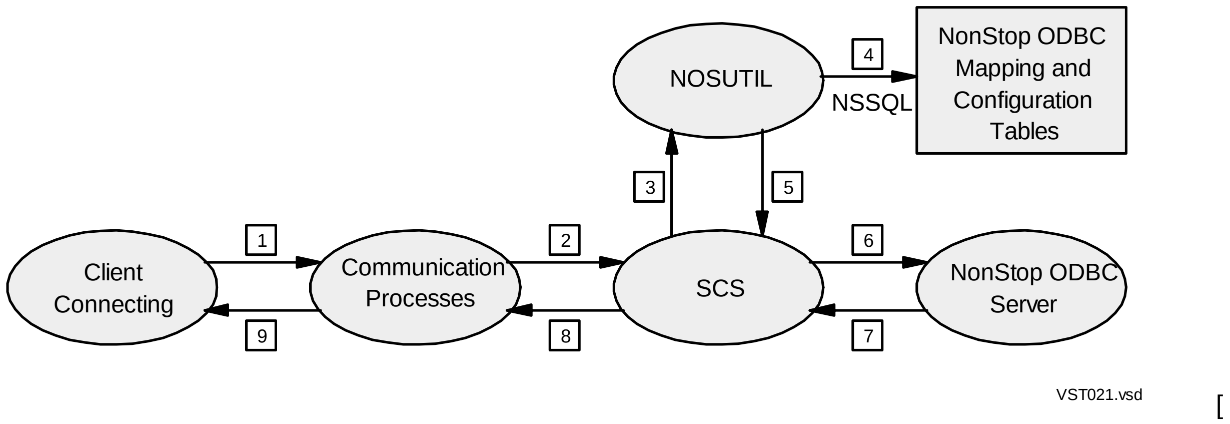


Message Flows

Understanding the HP side of the NonStop ODBC Server involves recognizing what objects are present and how they are manipulated. To that end, it will help to view the message flows between the components. For this and following discussions, it is important to distinguish between what the NonStop ODBC Server does and the way in which it does it.

Suppose that the NonStop ODBC Server has been installed and the clients configured. When a client connects, a connect or logon message (the HP term logon and the ODBC/DBLIB term login are interchangeable) arrives at the appropriate SCS. SCS chooses a NonStop ODBC server and passes the logon message to the server. Figure 2-17 shows the message flow that results when a client connects.

Figure 2-17. Message Flow for a Client Connection



1, 2 Client connection causes a message to be sent to the designated SCS. The message contains a username, a password, and perhaps a database name.

3 SCS recognizes the connect message and starts a thread for it. SCS sends a message (containing the connect message) to NOSUTIL to determine what server class should be used.

4 Using the username from the connect message, NOSUTIL accesses the NonStop ODBC Server mapping tables (using NonStop SQL/MP) to determine the Guardian name and the server class for the client.

5 NOSUTIL returns the information to SCS.

6 SCS picks a server from the designated server class (or starts one, if necessary) and passes the connect message to it.

7 The chosen NonStop ODBC server processes the connect message. Using the password from the connect message and the Guardian ID from NOSUTIL, the client is authenticated. The NonStop ODBC server initializes itself and replies to the connect message.

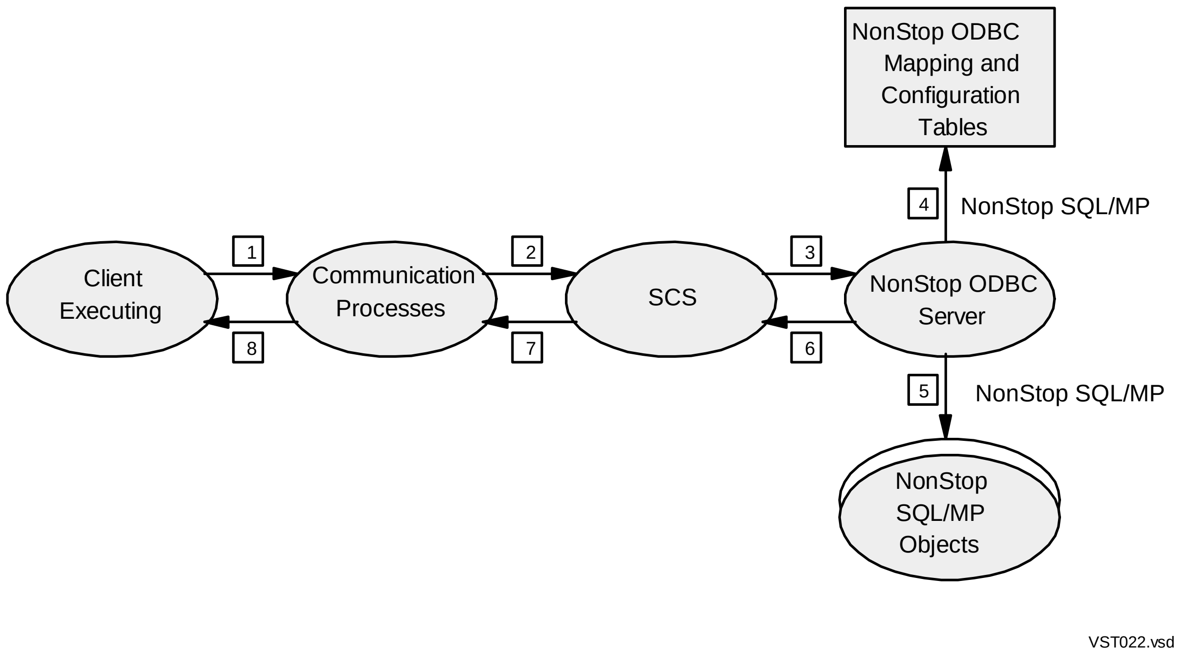
8, 9 The reply is sent to the client to complete the connection. The client can now proceed to execute SQL statements.

To improve performance, SCS caches the look-up done by NOSUTIL, so messages 3, 4, and 5 can often be omitted. When SCS is dynamically configured while in operation, the cache is reset to an empty state.

Having connected successfully, the client can start sending SQL statements to execute.

Using the SQL API, the client submits one (or more) SQL statements; DBLIB or the NonStop ODBC/MP driver sends a logical message to the NonStop ODBC server. The NonStop ODBC server executes the statement or statements and replies with the results. The client is then free to submit more SQL statements.

Figure 2-18. Message Flow for Client SQL Execution



1, 2 Client executes SQL statement at the API; a message is sent to SCS containing the statement.

3 SCS forwards the message, without examining its contents, to the appropriate NonStop ODBC server.

4 The NonStop ODBC server translates the SQL statement from Transact-SQL or CORE SQL to NonStop SQL/MP. This involves mapping (translating) certain names, such as table names, from TSQL or CORE form to NonStop SQL/MP form. To do this, the NonStop ODBC server uses NonStop SQL/MP to access the mapping tables.

5 The NonStop ODBC server executes the client's statement. This often involves executing NonStop SQL/MP statements against the database.

6 The NonStop ODBC server translates the result data (or error message) from NonStop SQL/MP form to TSQL or CORE SQL form and sends a reply.

7, 8 The reply is sent to the client and the results or error message are returned through the API.

To improve performance, the NonStop ODBC Server caches the reading of the mapping tables so that message 4 can often be omitted. When SCS is dynamically configured while in operation, the cache is reset to an empty state.

If the client SQL statement is a stored procedure execution, Step 4 involves mapping the stored procedure name to a Pathway system and the server class that contains the body of the procedure. In addition, Step 5 is a ServerClass_Send to the server class. The other steps remain the same.

NonStop ODBC Server Objects and Relationships

In addition to enabling clients to connect to and use a database, the NonStop ODBC Server also supports configuring many aspects of the connection for performance and management reasons.

Clients connect under a username; all the client's attributes are based on this name. The term "logical username" refers to names in the client's name space. The NonStop ODBC Server supports aliases so the logical name with which the client connects can be mapped to another logical name. In addition, the logical name is mapped to the Guardian name under which the NonStop ODBC server runs.

The following are the types of names used by the NonStop ODBC Server:

Alias username	The logical name provided by the client in the connect/logon message. This name distinguishes the client to the NonStop ODBC Server.
Logical username	The logical name the client uses for object qualification. All alias usernames are mapped to a logical username (which can be the same name for more than one alias username).
Guardian username	The name under which the NonStop ODBC server performs authentication and all access authorization. There is a one to one association between a logical username and a Guardian username.

Most attributes of a client connected to the NonStop ODBC Server are associated with the client's alias username; a few are associated with the corresponding logical username or Guardian name.

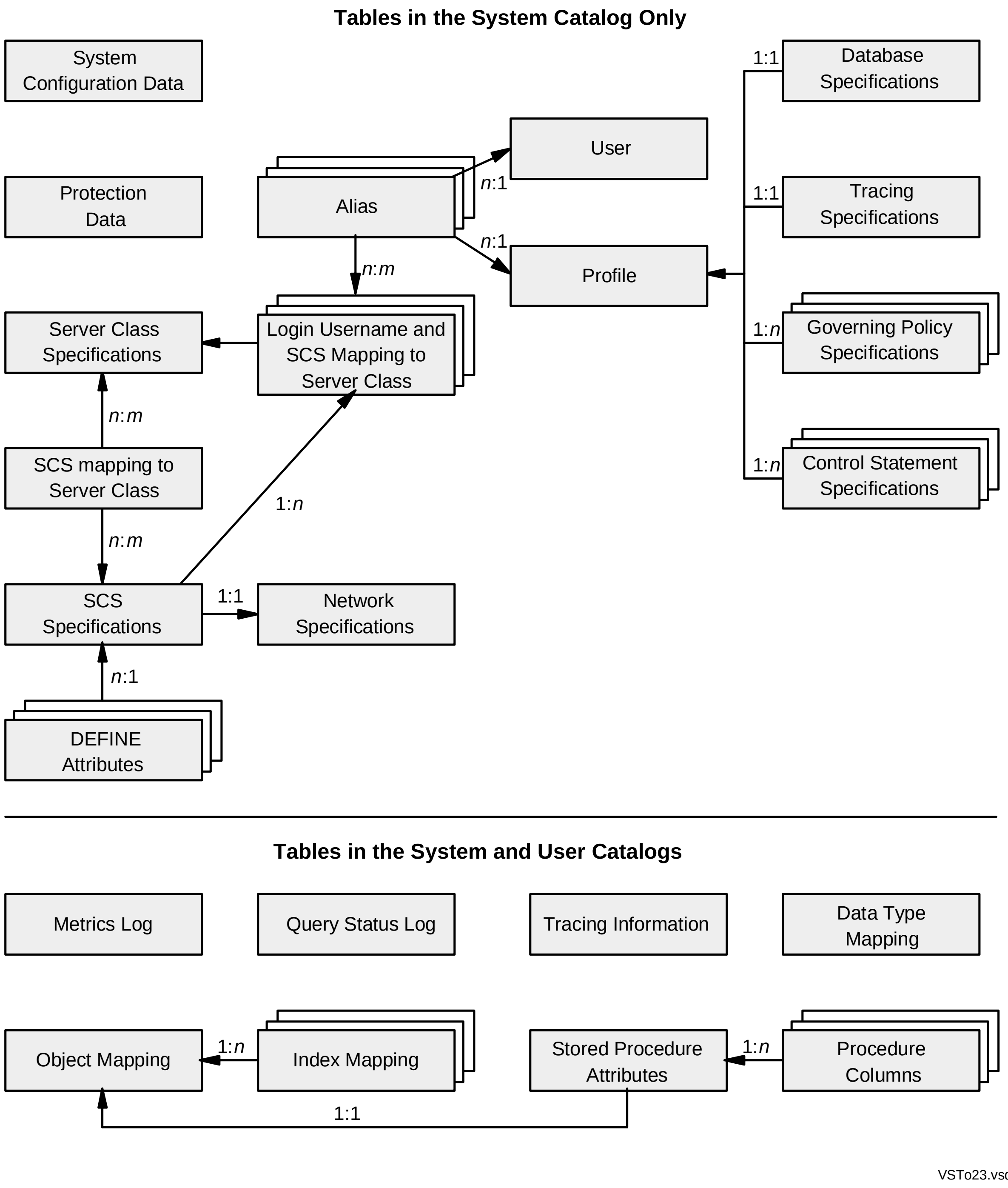
NonStop ODBC Server entities of interest are as follows:

User	A set of attributes that describes the association between the logical username and the Guardian username.
Alias	An association of alternate usernames with profile names and logical usernames.
Profile	A set of attributes that describe the NonStop ODBC Server environment, the NonStop SQL/MP environment, and the resource accounting environment. The profile references both the resource accounting and tracing configurations applicable to this profile.
SCS configuration	A set of attributes that describe the SQL Communication Subsystem process creation values, network protocol and service characteristics, performance and diagnostic settings, and program definitions.

NonStop ODBC server configuration	A set of attributes that describe the NonStop ODBC server process creation values, default values, program definition, server counts, and performance and diagnostic settings.
Resource governing information	A set of governing policies that define what tests are to be made on resource usage and what action is to be taken if resource usage exceeds preset limits.
Trace information	A set of attributes defining what is traced (input and output messages, SQL statements executed, errors generated, hits and misses to cached SQL statements) and where the trace output is placed. Most items are for debugging, but SQL caching statistics are of use for performance tuning.

The relationships among the names and the objects are shown in Figure 2-19.

Figure 2-19. Relationships Among NonStop ODBC Server Objects



A number of alias usernames can be related to a single logical username. The NonStop ODBC Server imposes a 1:1 relationship between logical and Guardian usernames to simplify NonStop ODBC Server utilities (which sometimes need to derive a logical username from a Guardian username).

The following is an example involving usernames. There are several alias usernames by which a client can connect. The name “SQL_JONES” illustrates using essentially the same name on both the client and server side (which is useful if the named person does some work on a PC and some work on the NonStop server). The other “Joneses” represent two aliases that also map to the logical username “SQL_JONES.” The difference is that each alias is mapped to a different profile.

The subsequent two aliases permit the database administrator to login as either “DBO” or “ADMINSTRATOR,” both of which map to the same logical username, Guardian username, and profile.

Alias Username	Logical Usename	Profile	Guardian Username
SQL_JONES	SQL_JONES	CLASS2	SQL.JONES
JONES_CACHE_ON	SQL_JONES	CLASS_CACHE_ON	SQL.JONES
JONES_CACHE_OFF	SQL_JONES	CLASS_CACHE_OFF	SQL.JONES
DBO	DBO	ADMIN	SUPER.MGR
ADMINSTRATOR	DBO	ADMIN	SUPER.MGR
GUEST	SQL_GUEST	DEFAULT	SQL.GUEST

The particular choice of relationships between these objects—what is related to what and in what ratio the relationships exist—is based on several factors. In general, all attributes of a client are associated with the alias username, when possible, to allow maximum flexibility. A few attributes are associated with the logical or Guardian username for simplicity or performance.

Name Mapping

A principal job of the NonStop ODBC Server is to map (translate) several types of names, including those of databases, users, tables, and columns, between the client and server environments. The requirements for mapping—to be flexible, easy to maintain, and to perform well—usually conflict. The NonStop ODBC Server uses a table look-up scheme. The table is a NonStop SQL/MP table. Each row represents one or more relationships between a client and a SQL/MP name.

NonStop ODBC Server Catalogs

The NonStop ODBC Server catalogs are sets of NonStop SQL/MP tables and views maintained by the NonStop ODBC Server. They hold configuration information, are involved in name mapping, and are used to support client-visible SQL catalogs.

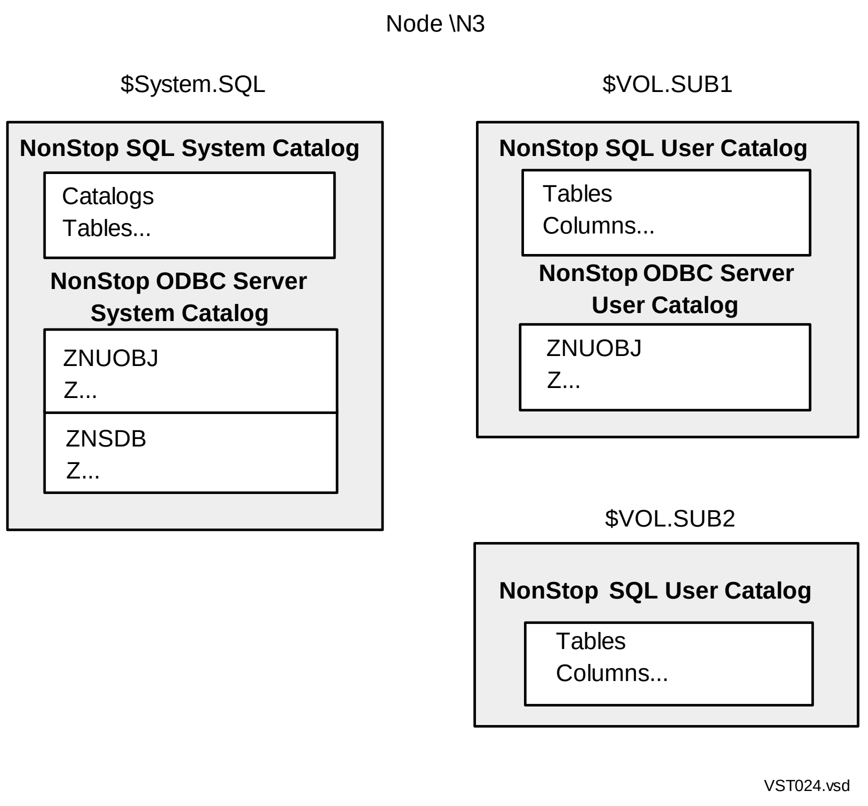
It is useful to think of the NonStop ODBC Server catalogs as extensions of NonStop SQL/MP catalogs. A NonStop ODBC Server catalog can exist only on a subvolume that contains a NonStop SQL/MP catalog (recall that in this case NonStop SQL/MP objects appear to clients as a database). When asked to create a database, NOSUTIL creates a SQL/MP catalog if one doesn’t exist in the same subvolume.

There are two types of NonStop ODBC Server catalogs: the NonStop ODBC Server system catalog and a NonStop ODBC Server user catalog. There must be at least one NonStop ODBC system catalog on a system for the NonStop ODBC system to function. There may be zero or more NonStop ODBC customized user catalogs on a system. Each customized user catalog is registered in one or more NonStop ODBC system catalogs. The NonStop ODBC Server system catalog, which must exist if there are any NonStop ODBC Server objects on the node, includes a NonStop ODBC Server user catalog.

Figure 2-20 shows an example of NonStop ODBC Server catalogs. Subvolume $SYSTEM.SQL on node \N3 contains the NonStop SQL/MP system catalog and the NonStop ODBC Server system catalog. $VOL.SUB1 contains a NonStop SQL/MP catalog and a NonStop ODBC Server user catalog. $VOL.SUB2 contains a NonStop SQL/MP catalog, but no NonStop ODBC Server user catalog.

The names of all the NonStop ODBC Server catalog tables start with “ZN” to minimize conflicts with user naming. In this figure, clients would access databases named MASTER and N3_VOL_SUB1. The system catalog is always named MASTER, which follows SQL Server conventions.

Figure 2-20. NonStop ODBC Server Catalogs



NonStop ODBC Server Catalog Schema

The following are the tables in a NonStop ODBC Server user catalog:

Table	Usage
ZNUDT	Map data types
ZNUIX	Map index names
ZNUMTRX	Log resource accounting data
ZNUOBJ	Map table and view names
ZNUPCOL	Map stored procedure column names
ZNUPROC	Map stored procedure names
ZNUQST	Log query status data
ZNUTRA*	Log trace data

* Template only; user can define actual table name.

These tables are used by the NonStop ODBC server to map table, view, index, and stored procedure names. The ZNUPCOL describes the parameters of the stored procedures.

The NonStop ODBC Server system catalog serves as a NonStop ODBC Server user catalog and has several other uses: it maps items that are constant over the user catalogs: database names, data types, error messages, logical usernames, and alias-to-logical relationships. It holds configuration information for usernames. Finally, it holds some special information needed to support the DBLIB catalogs SYSPROTECTS and SYSVALUES.

The following are the tables in a NonStop ODBC Server system catalog:

Table	Usage
ZNSALT	Configure alias usernames
ZNSCON	Configure CONTROL statement attributes
ZNSDB	Map database names
ZNSDEF	Configure DEFINE attributes
ZNSGOV	Configure resource governing policies
ZNSMSG	Map error messages
ZNSNET	Configure network service attributes
ZNSPROF	Configure profiles
ZNSPROT	Help support SYSPROTECTS for TSQL
ZNSSCFG	Configure system attributes
ZNSSCS	Configure SCS
ZNSSER	Configure server class
ZNSSMAP	Map SCS to server class
ZNSTRA	Configure trace information

Table	Usage
ZNSUMAP	Map login username and SCS to server class
ZNSUS	Map logical usernames to Guardian usernames
ZNSVALUE	Help support SYSVALUES for TSQL
ZNU*x*	All tables of a NonStop ODBC Server user catalog

Client Catalogs

A previous description showed how the NonStop ODBC Server catalogs are used by the NonStop ODBC server to map client names to NonStop SQL/MP names. The other main use of the NonStop ODBC Server catalogs is to support client SQL catalogs for TSQL and CORE SQL users.

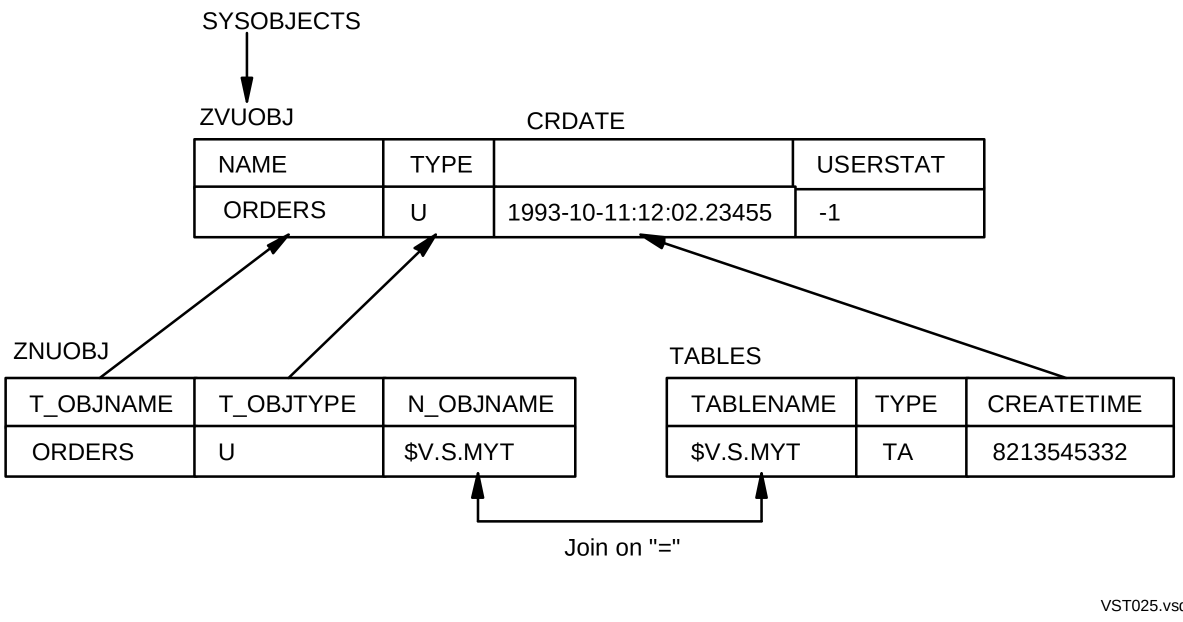
TSQL Catalogs

For TSQL users, SQL Server supports several SQL catalog tables for user and system catalogs. DBLIB clients typically access a subset of these tables in their applications.

Figure 2-21 shows how the TSQL catalog SYSOBJECTS is supported using the NonStop SQL/MP catalog table TABLES and the NonStop ODBC Server user catalog table ZNUOBJ (only enough columns of SYSOBJECTS are shown to illustrate the techniques).

ZVUOBJ is a NonStop SQL/MP view on the join of tables TABLES and ZNUOBJ. SYSOBJECTS is a TSQL table whose name is mapped to ZVUOBJ (so ZNUOBJ contains another entry for the mapping between SYSOBJECTS and ZVUOBJ, not shown here).

Figure 2-21. Support of TSQL Catalog SYSOBJECTS



Think of ZNUOBJ as an extension of TABLES. ZNUOBJ has a column N_OBJNAME (NonStop SQL/MP object name) that contains the same entries as TABLENAME in TABLES. Joining the two tables on these columns gives the “extended TABLES” table.

A row of SYSOBJECTS is built by taking columns from a join row. Columns for SYSOBJECTS are selected from TABLES when possible; otherwise, they are selected from ZNUOBJ. The table NAME (A#_#TABLE) is taken from ZNUOBJ. The table TYPE (U for user table) is also taken from ZNUOBJ. The CRDATE (creation date) is taken from TABLES, but it must be converted to the correct format. The USERSTAT column has no meaning on a NonStop server, so it is always set to the constant -1.

The following example illustrates the NonStop SQL/MP CREATE VIEW statement used to create the view ZVUOBJ. When NOSUTIL customizes a NonStop SQL/MP catalog, it issues this NonStop SQL/MP statement. NOSUTIL also inserts a record into ZNUOBJ (using a NonStop SQL/MP INSERT statement) that maps the TSQL table name SYSOBJECTS to ZVUOBJ.

```
CREATE VIEW ZVUOBJ (NAME, TYPE, CRDATE, USERSTAT, ...) AS
  SELECT
    T_OBJNAME, T_OBJTYPE CONVERTTIMESTAMP(CREATETIME), -1,...
  FROM TABLES, ZNUOBJ
  WHERE TABLES.TABLENAME = ZNUOBJ.N_OBJNAME;
```

This example reflects the basic techniques used to support all client catalogs.

ODBC Catalogs

Unlike database vendors, ODBC does not define a SQL catalog schema. Instead, the ODBC CLI defines functions intended to extract catalog information.

The ODBC functions cover tables, columns, index columns, stored procedures, and stored procedure columns (the parameters of the procedures). Each CLI function has parameters that serve to qualify on object names. For example, the function for tables (called SQLTables) has parameters to qualify on each of the three parts of the name and the table type.

The NonStop ODBC Server supports these calls for ODBC by having the NonStop ODBC/MP driver issue corresponding SQL statements against the NonStop SQL/MP and NonStop ODBC Server catalog tables.

For example, an ODBC application issues a function call of the form:

```
SQLTables (tablename = "XY*")
```

The NonStop ODBC/MP driver forms a query of the following generalized form and sends it to the NonStop ODBC Server:

```
select ...
from ZVUOBJ
where name like "XY%"
```

The NonStop ODBC server processes the SELECT statement in the normal manner.

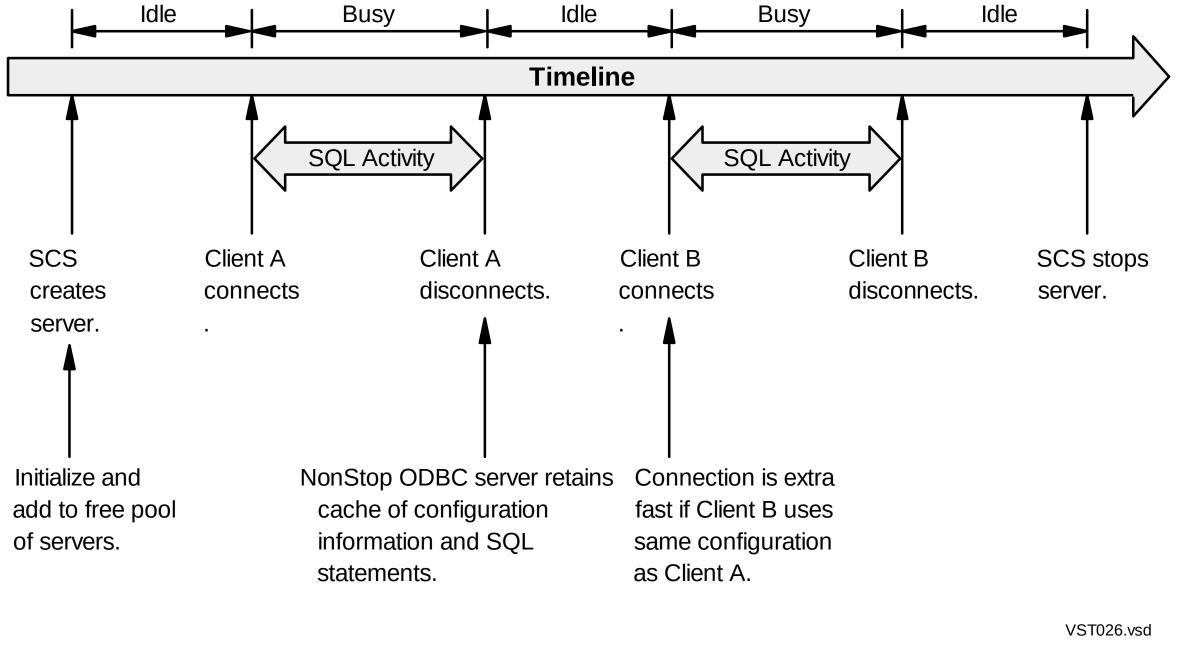
An additional table (ZVUOCOL) was added to the NonStop ODBC Server catalog tables needed for TSQL support to enable the NonStop ODBC Server to support ODBC catalog functions.

NonStop ODBC Server

The NonStop ODBC server is the physical server process within the NonStop ODBC Server product. The principal task of the NonStop ODBC server is to deal with SQL statements from clients. This is similar to NonStop SQL/MP's task of compiling and executing NonStop SQL/MP statements. Each NonStop ODBC server is dedicated to one client connection at a time. When not servicing a client, SCS maintains idle NonStop ODBC servers in a free pool.

The life cycle of an NonStop ODBC server process is shown in Figure 2-22.

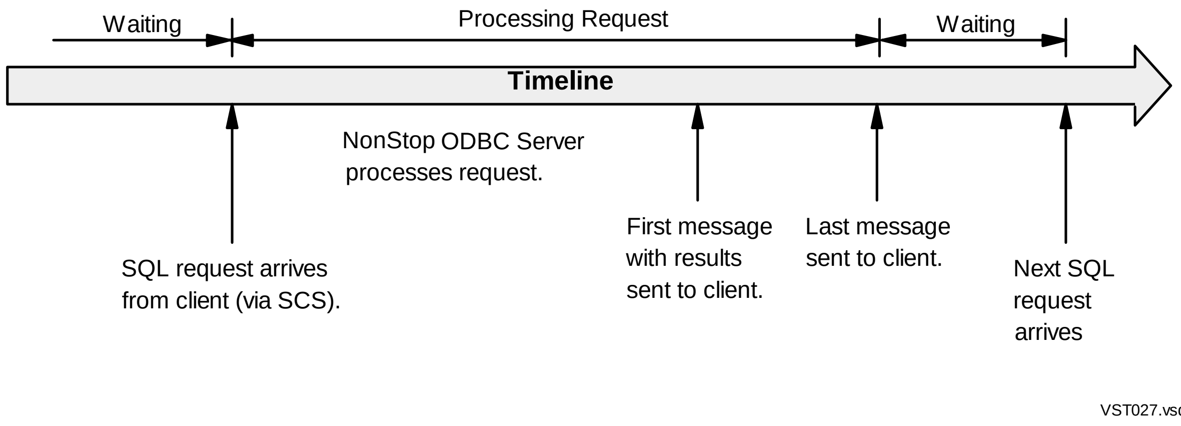
Figure 2-22. Life Cycle of an NonStop ODBC Server Process


Idle
Busy
Idle
Busy
Idle
Timeline
SQL Activity
SQL Activity
SCS creates server.
Client A connects.
Client A disconnects.
Client B connects.
Client B disconnects.
SCS stops server.
Initialize and add to free pool of servers.
NonStop ODBC server retains cache of configuration information and SQL statements.
Connection is extra fast if Client B uses same configuration as Client A.
VST026.vsd

When SCS creates the NonStop ODBC server, the server initializes itself then waits until client A connects. When A disconnects, the server holds its cached information. Client B then connects for a session. Finally, SCS stops the server, perhaps because there are too many idle servers in the server class.

The NonStop ODBC server handles each SQL request fully before processing the next one. Processing a request involves reading the entire input message (perhaps several packets), performing all the actions required, and returning the results to the client (again, perhaps several packets). This is illustrated in Figure 2-23.

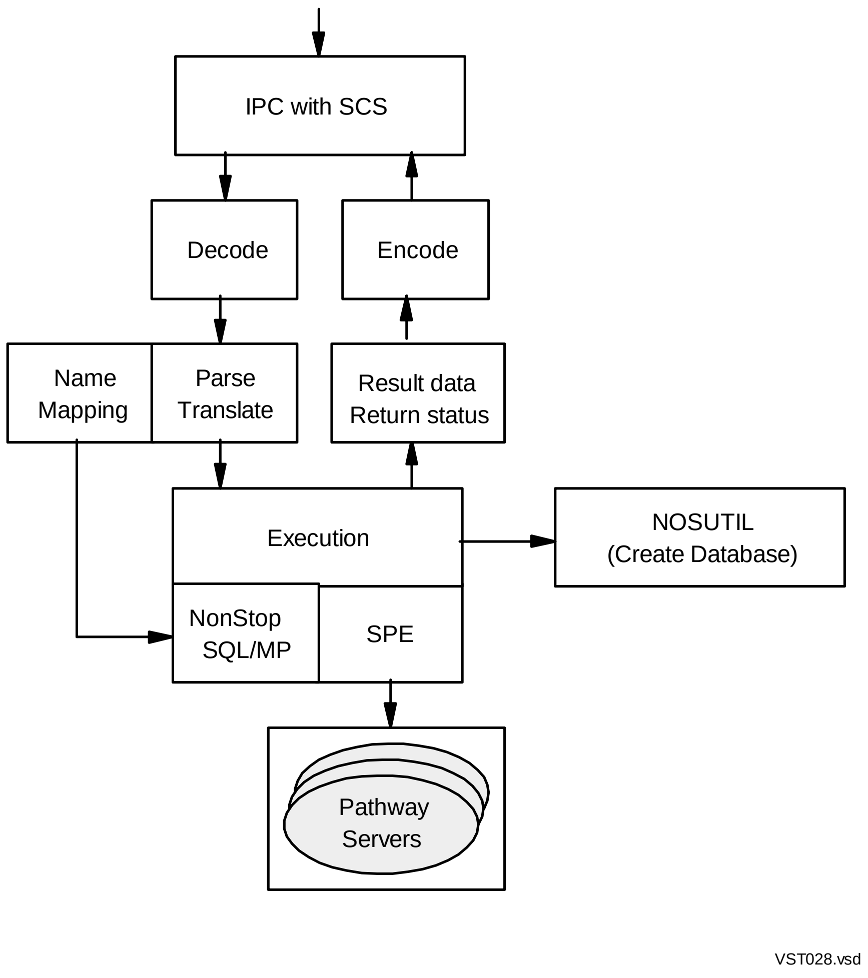
Figure 2-23. Processing an SQL Request


Waiting
Processing Request
Waiting
Timeline
NonStop ODBC Server processes request.
SQL request arrives from client (via SCS).
First message with results sent to client.
Last message sent to client.
Next SQL request arrives
VST027.vsd

As the NonStop ODBC server obtains results (such as rows fetched for a cursor), it starts sending the results to the client. Based on this behavior, the client library can do a variety of things: it could wait for all results before returning to the application, it could return results as available, it could even support asynchronous calls. The NonStop ODBC/MP driver returns a result row as soon as it receives it from the NonStop ODBC server.

When a request arrives, the NonStop ODBC server uses one or more READ operations on the interprocess communication (IPC) with SCS to receive the entire request, which is then decoded. A SQL request can be one SQL statement or several (called a batch in TSQL; batches are supported for DBLIB clients only).

Figure 2-24. Inside the NonStop ODBC Server



Each statement is parsed, translated, and placed in a list of statement objects. There are actually two parsers, one for Transact-SQL and one for CORE SQL; the choice is based on the mode in which the NonStop ODBC server is running. The parser builds a parse tree in memory; the translator traverses the tree and makes changes.

An important translator task is to map certain client names into NonStop SQL/MP names; the translator calls a name mapping module to do this. The name mapping module uses NonStop SQL/MP SELECT statements to read the NonStop ODBC Server catalog tables for the needed map entries. For performance, the name mapping module caches the map entries in memory.

Another important task of the translator is to decide which SQL statements can be cached and which need to be parameterized. If they can be cached, the translator converts TSQL statements and execute-direct CORE SQL statements into the prepare-and-execute form. If they need to be parameterized, the translator inserts dynamic parameters, represented by question marks (?), in the correct places in each statement to be prepared; the translator builds the input parameter values for the execute stage using the literal values the question marks displaced.

Finally, the translator traverses the parse tree and produces the text of the NonStop SQL/MP statement to be executed. For pass-through statements, the parsing proceeds only to the point of discovering the identifying SELECT “TDM:” phrase. At that point, the translator omits the remainder of the parsing and begins checking for parameters in the pass-through statement itself.

If any errors are detected during parsing or translating, the errors are returned to the client. Otherwise, the translator calls the execution module, which proceeds to execute the translated statements one by one.

For NonStop SQL/MP statements, the execution module checks the cache as appropriate, executes the statements, encodes the results, and sends them to the client. For stored procedure execution, the execution module calls a stored procedure execution (SPE) module that assembles and executes the ServerClass_Send. The execution module then retrieves and processes result sets from the SPE module just as it does for a NonStop SQL/MP cursor SELECT; each result set is encoded and sent to the client.

Between issuing NonStop SQL/MP fetches and receiving messages from a Pathway server class program (for SPE), the NonStop ODBC server checks whether a Cancel message has arrived from the client. If so, the server closes the NonStop SQL/MP cursor or cancels any ServerClass_Send in progress.

SQL Communication Subsystem

SCS performs three major tasks:

- SCS handles client communication.

  SCS relieves the NonStop ODBC servers from the necessity of handling most of the network communication and protocol issues. SCS supports the various communication protocols (NETBIOS or TCP/IP) and has a thread for each client connection. There is a simple open/close/read/write IPC defined between SCS and the NonStop ODBC servers. This is the same service provided by the Transaction Delivery Process (TDP) for Remote Server Call (RSC) and Pathway.

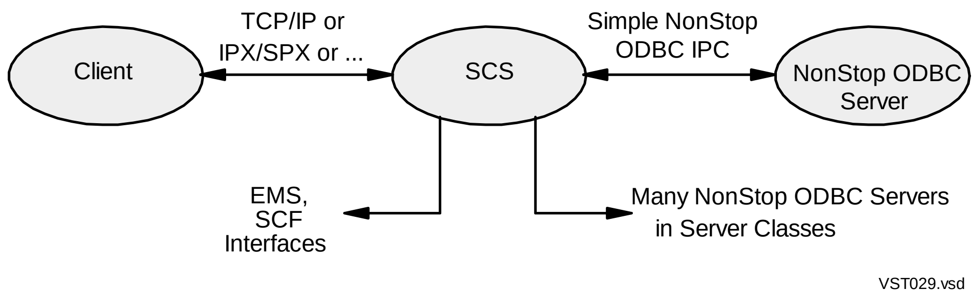
- SCS manages server classes.

SCS manages server classes, much as Pathway does. SCS starts and stops servers according to configuration information. This task is done solely for performance, to avoid excessive NEWPROCESS commands and SQL compilation for client connections.

- SCS provides a management interface.

  SCS provides interfaces to the Subsystem Control Facility (SCF) and the Event Management Service (EMS) to help administrators manage a NonStop ODBC Server system.

Figure 2-25. SCS Tasks

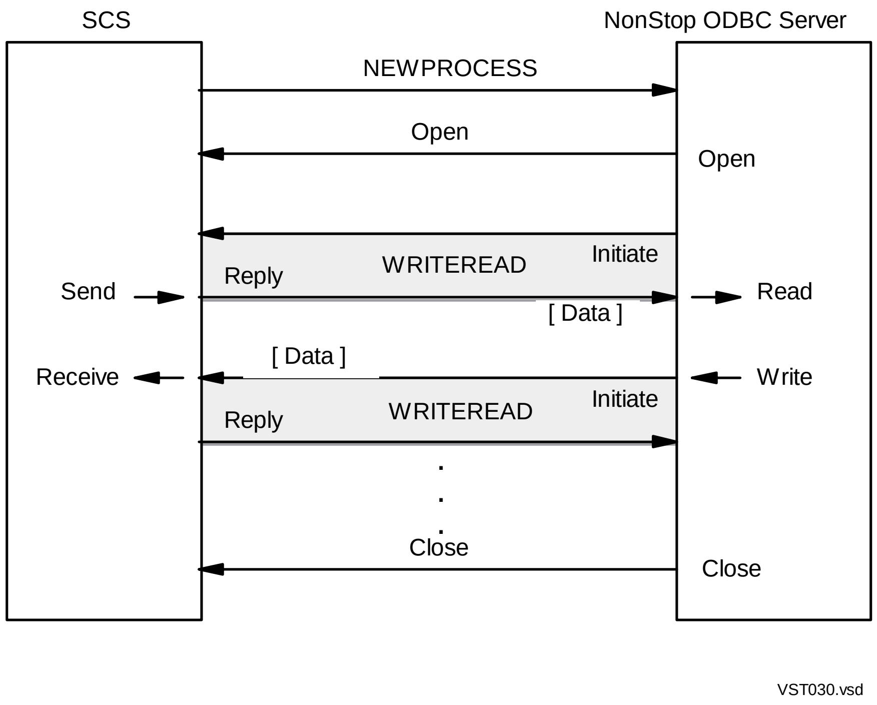


Client Communication

After starting and initializing itself, a NonStop ODBC server opens SCS and uses WRITEREAD to support both a logical Read and Write. A WRITEREAD is posted when the NonStop ODBC server is ready to read a request message from the client. After processing a request, WRITEREAD is used to send reply data to the client.

Figure 2-26 illustrates this communication process.

Figure 2-26. Interprocess Communication Between SCS and NonStop ODBC Server



A client request is one logical message, but it can be sent in multiple 512-byte transport messages. As SCS receives messages from the client, it forwards them without change to the NonStop ODBC server.

When a request is being received, the NonStop ODBC server reads all the incoming messages before it starts processing the request. As it is acquiring result data, it writes messages to SCS as soon as the messages are ready.

While the NonStop ODBC server is processing a request, it usually has a WRITEREAD posted to SCS. If the client issues a cancellation request, the WRITEREAD can terminate; the NonStop ODBC server then cancels its activities.

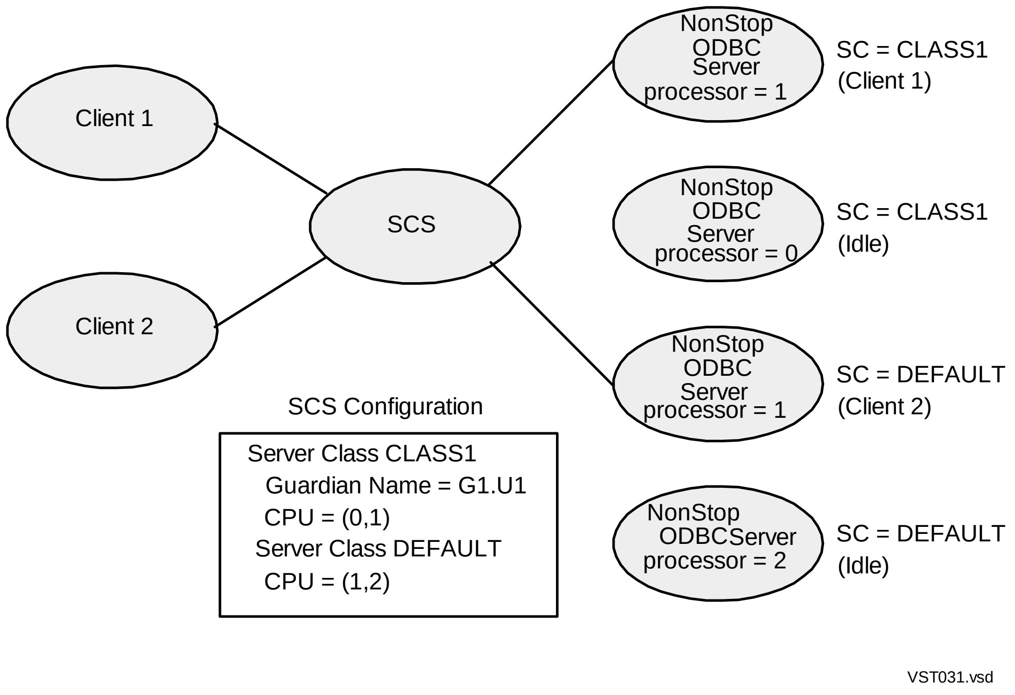
If a stored procedure is being executed, the NonStop ODBC server uses a NOWAIT ServerClass_Send so the NonStop ODBC server process can process the cancel message. If NonStop SQL/MP is being executed, which is done in the WAIT mode, the NonStop ODBC server does not process the cancel message until it has completed the SQL request. However, if a cursor SELECT is being executed, the NonStop ODBC server checks for cancellation between fetches of the cursor.

Server Classes

Server classes are named and have a number of attributes pertaining to the NonStop ODBC server processes in the class (for example, allowable processors and initial priority). There are also attributes limiting the number of servers and specifying the number of free servers. There is also a special DEFAULT server class to handle clients that are not configured for a specific server class.

Figure 2-27 shows two server classes, each with two NonStop ODBC servers. Two clients are connected, one to each server class.

Figure 2-27. SCS Management of Server Classes

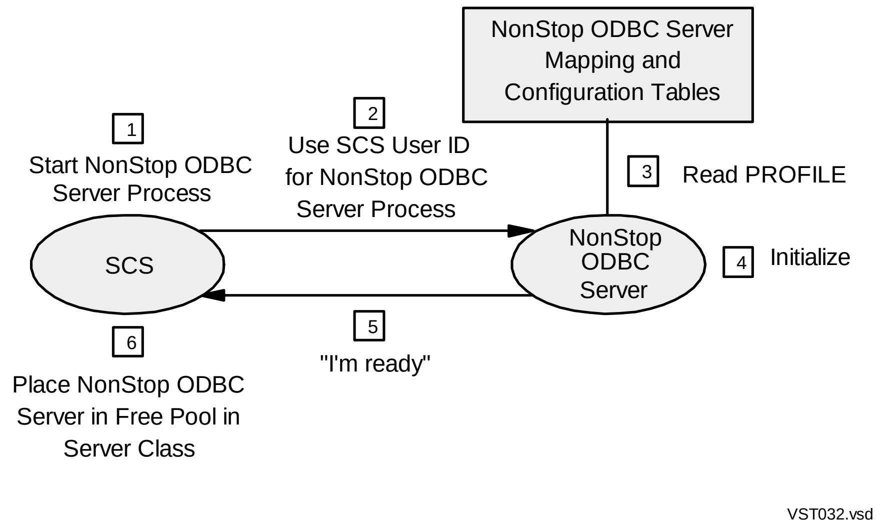


The ZNSSER table contains the server class configuration information used by SCS to start each NonStop ODBC server process. At start-up time, SCS gets the configuration information from NOSUTIL and sends each new NonStop ODBC server process a message containing this information. The NonStop ODBC server completes its initialization and replies to SCS that it is up. SCS then places the new NonStop ODBC server process in the pool of available servers.

When SCS first starts a NonStop ODBC server process, the new server process runs under the same user ID as the SCS process.When a client connects to a NonStop ODBC server, the server process calls the Guardian USER_AUTHENTICATE_ procedure to validate the client’s username and password. If the validation is successful, the NonStop ODBC server switches to the user ID of the connected client. The client can then send requests to the NonStop ODBC server.

Figure 2-28 shows the initialization of an NonStop ODBC server process.

Figure 2-28. Initializing a NonStop ODBC Server


NonStop ODBC Server Mapping and Configuration Tables
1
Start NonStop ODBC Server Process
2
Use SCS User ID for NonStop ODBC Server Process
3
Read PROFILE
SCS
NonStop ODBC Server
4
Initialize
5
"I'm ready"
6
Place NonStop ODBC Server in Free Pool in Server Class
VST032.vsd

Management Interfaces

One of the NonStop ODBC Server requirements is to provide interfaces for managing a potentially large and complex NonStop ODBC Server system (involving several clients or heavy workloads from each client).

The NonStop ODBC Server does not provide additional functions for managing NonStop SQL/MP objects and Pathway systems beyond those provided by these products. The NonStop ODBC Server does provide a number of configuration interfaces (described in later subsections) as well as some basic EMS and SCF support within SCF so that an administrator can monitor a NonStop ODBC Server system.

- EMS interface–SCS generates EMS messages for session events (starting, stopping) and NonStop ODBC server events (starting, started). An administrator can scan the EMS log to diagnose problems.
- SCF interface–SCS supports commands to show the status of all sessions and all servers. The administrator can identify particular session and server attributes and use other tools to probe further. For example, the name of a NonStop ODBC server can be used to check process status or to stop the process.

Server Connections

As stated previously, a connection involves a DBLIB or ODBC client initiating a connection with a NonStop ODBC server. A connection is a significant event in that a number of things happen involving authentication, NonStop ODBC server selection, and attribute setting (much of what happens depends on configuration settings).

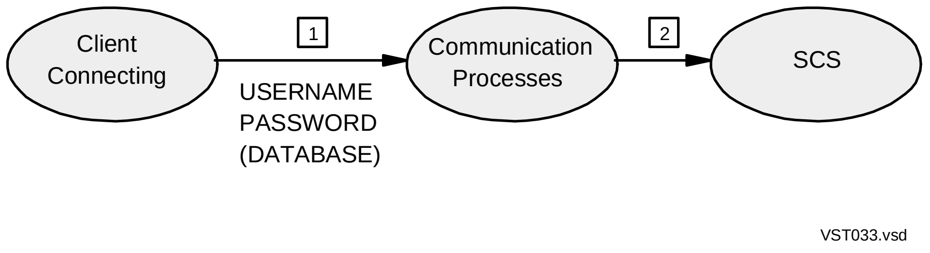
When connecting, a client application supplies a username, a password, and an optional database name. With DBLIB, these items are passed to the server in a logon message. With ODBC, the driver chooses how to connect to a server. For the NonStop ODBC Server, the HP driver mimics the DBLIB scheme (sends the same logon message).

ODBC supports a number of options that can be configured for a connection. A driver could send these options with the logon message, but the HP driver does not do so. Instead, it lets the connect operation finish, then sends a number of pass-through statements that set the various connect attribute values.

A client connection is shown in detail in Figures 2-29 through 2-32. This sequence shows where and how each aspect of the connection process is carried out.

When the client issues a connect request, DBLIB or the NonStop ODBC/MP driver sends a connect message containing the username, password, and, optionally, a database name to the configured network address where SCS is listening, as shown in Figure 2-29.

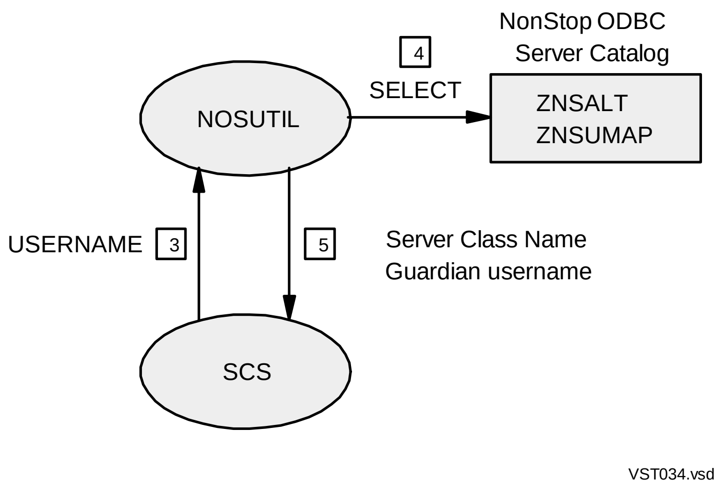
Figure 2-29. Client Initiation of Connection



The task of SCS is to determine from which server class to acquire a NonStop ODBC server. SCS maintains a private NOSUTIL process for this purpose. SCS extracts the username from the connect message and passes it to the NOSUTIL process. NOSUTIL determines the server class name associated with the username and returns the server class name in a message to SCS.

If the username is not found in the ZNSALT table in the NonStop ODBC Server system catalog, NOSUTIL attempts to decode the username to a Guardian name (the username must be of the form *group_user*, which corresponds to a valid Guardian name of the form *group.user*). If the decoding fails, NOSUTIL returns a “login rejected” error to SCS and to the client. Otherwise, NOSUTIL returns the DEFAULT server class name to SCS. NOSUTIL does not check if the Guardian name exists—that is done later.

Figure 2-30. SCS Determination of Server Class



SCS maintains a cache of username and server class name relationships that it checks before contacting NOSUTIL. If the username is found in the cache, Steps 3, 4, and 5 in Figure 2-30 are omitted.

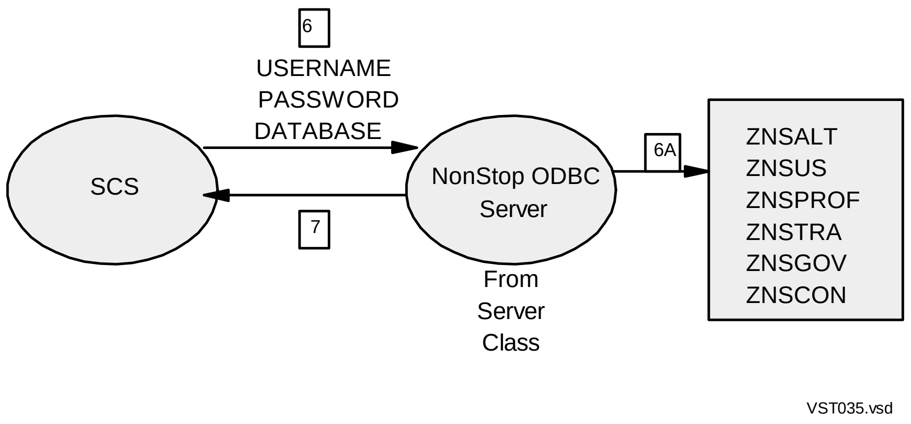
When the server class name is determined, SCS selects a free NonStop ODBC server from the class, starting one if necessary. SCS passes the original connect message from the client to the NonStop ODBC server.

The NonStop ODBC server reprocesses the connect message from the beginning. The server process checks the NonStop ODBC Server mapping table ZNSALT to determine if the username is an alias; if it is, the NonStop ODBC server uses the corresponding logical username. The server process accesses ZNSUMAP, ZNSALT, and ZNSUS to determine the attributes of the username—the Guardian name, server class name, and profile name to use. Finally, the NonStop ODBC server accesses the ZNSPROF table to determine the profile settings and, optionally, the ZNSCON, ZNSGOV, and ZNSTRA tables to determine the control, governing, and trace settings.

If the username is not found in ZNSALT, the NonStop ODBC server attempts to decode it to a Guardian name. If successful, the DEFAULT profile is used. In any case, if a Guardian name is obtained, the NonStop ODBC server authenticates the client by USERAUTHENTICATE command with the Guardian name and the password from the connect message.

If the logon fails, the NonStop ODBC server returns an error to the client; the process then returns to the set of free servers in the class. If the connection is successful, the NonStop ODBC server initializes itself according to the settings of the profile and returns a “successful connection” message to SCS.

Figure 2-31. NonStop ODBC Server Processes Connection

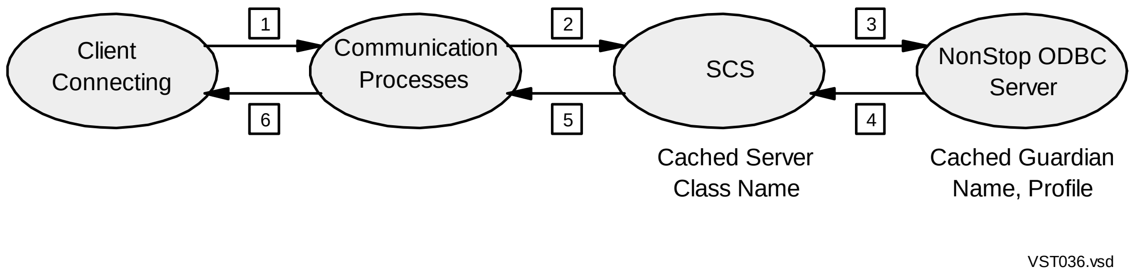


The NonStop ODBC server caches the connect-username—the Guardian name/profile name relationship it used for the most recent connection to it. If the new connection is under the same username, Step 6A in Figure 2-31 is omitted. Authentication of the user with the password in the connect message is done every time, however.

Finally, SCS notes that the connection was successful (it now maintains the thread for the connection) and returns a “successful connection” message to the client. The client can then proceed with issuing SQL statements.

If all the caching is successful, the minimum message flow to establish a connection is a completed circuit to a NonStop ODBC server, as shown in Figure 2-32.

Figure 2-32. Minimum Message Flow to Connect a Client



NOSUTIL

NOSUTIL is a utility process whose purpose is to manage the NonStop ODBC Server catalogs; it also provides services to look up catalog information. NOSUTIL can be invoked directly, through its command interface, NOSCOM, or be called by SCS and the NonStop ODBC server.

Catalog Integrity

The NonStop SQL/MP tables that make up the NonStop ODBC Server catalogs include certain integrity constraints, as follows:

- For example, the NonStop ODBC Server system catalog has a table, ZNSDB, that lists all customized NonStop SQL/MP catalogs supported for a given NonStop ODBC system catalog (that is, all NonStop ODBC Server user catalogs), and only those catalogs, on a NonStop node.
- Another example: if alias username “A” is associated with logical username “U” (in table ZNSALT), then “U” must have an entry in the username table (ZNSUS).
- Finally, the NonStop ODBC Server catalogs have referential associations to the NonStop SQL/MP objects they describe.

Administrators and users must understand how the NonStop ODBC Server handles catalog integrity. The NonStop ODBC Server does not provide fully active support of NonStop ODBC Server catalog integrity for two reasons:

- Changes are managed manually

  The NonStop ODBC Server is not notified by NonStop SQL/MP when some DDL operation is performed on an object in the NonStop SQL/MP catalog associated with a NonStop ODBC Server catalog. Therefore, the NonStop ODBC Server administrator must reestablish referential integrity with specific NonStop ODBC utility commands. Two utility functions are provided to do this.

  One function is VALIDATE, which causes NOSUTIL to scan a designated NonStop ODBC Server catalog and produce a report on integrity violations (such as a NonStop SQL/MP table that is in the NonStop SQL/MP catalog, but that is not in the NonStop ODBC Server catalog).

  The second function is REFRESH, which causes NOSUTIL to alter the catalog to fix any violations. REFRESH alters only the NonStop ODBC Server catalog; it does not drop a NonStop SQL/MP table that is not in the NonStop ODBC Server catalog.

- Better usability

  When an administrator or user is making changes to a NonStop ODBC Server catalog, such as defining a new alias name or profile, NOSUTIL does not check all the integrity constraints. This allows catalog changes to be made in any order.

  When a NonStop ODBC server is running, however, integrity in the NonStop ODBC Server catalog is checked for the items accessed (for example, if an alias name is mapped to a username but the username is not listed in ZNSUS, an error is raised).

Catalog Operations

The basic NonStop ODBC Server catalog operations are:

- INSTALL/DEINSTALL/VALIDATE/REFRESH/UPGRADE the NonStop ODBC Server system catalog
- INSTALL/DEINSTALL/VALIDATE/REFRESH/UPGRADE a NonStop ODBC Server user catalog
- ADD/REMOVE/MODIFY alias and usernames
- ADD/REMOVE/MODIFY stored procedure names and attributes
- ADD/REMOVE/MODIFY profile and trace information
- ADD/REMOVE/MODIFY SQL Communication Subsystem configuration
- ADD/REMOVE/MODIFY system configuration defaults
- ADD/REMOVE/MODIFY server class definitions and mapping
- ADD/REMOVE/MODIFY network service configurations
- ADD/REMOVE/MODIFY resource governing policies
- ADD/REMOVE visibility of NonStop SQL/MP indexes and tables
- ADD/REMOVE stored NonStop SQL/MP CONTROL statements
- ADD/REMOVE stored NonStop SQL/MP DEFINE statements
- CREATE/DROP log tables for resource accounting, query status, and tracing

INSTALL is the command to customize a NonStop SQL/MP catalog. Note that migration from the HP SQL Server Gateway is supported by NOSUTIL. For example, if INSTALL is called on a NonStop SQL/MP catalog that was previously customized for the HP SQL Server Gateway, NOSUTIL first automatically deinstalls the HP SQL Server Gateway catalog.

Executing NOSUTIL

NOSUTIL can be called in one of four ways:

- By the NonStop ODBC Server Configuration Manager

  The NonStop ODBC Server Configuration Manager, a graphical user interface (GUI) tool, can be run to issue pass-through configuration commands to NOSUTIL.

- By NOSCOM

  NOSUTIL can be run directly to perform the utility functions mentioned previously under “Catalog Operations.” NOSCOM is the TACL-like command interpreter used for entering all utility statements.

- By a NonStop ODBC server

  A NonStop ODBC server creates and uses a private NOSUTIL process to support the SQL statement CREATE DATABASE or DROP DATABASE. For CREATE

DATABASE, NOSUTIL creates the NonStop SQL/MP catalog (if it does not already exist) and customizes it.

- By SCS

  SCS maintains one private NOSUTIL process that it uses to process connect operations and to get configuration information. It passes the connect message to NOSUTIL, which accesses the NonStop ODBC Server system catalog to map the login name and to determine the server class to use to obtain a NonStop ODBC server.

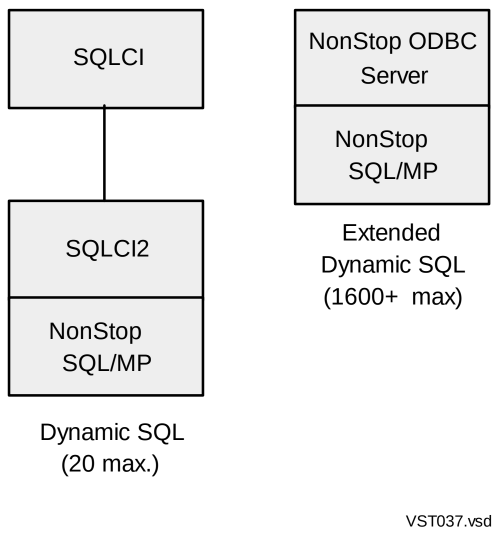
NonStop SQL/MP Execution

NonStop SQL/MP execution was originally modeled after the NonStop SQL/MP conversational interface (SQLCI). The DBLIB model is always execute immediate. There is no prepare-and-execute paradigm, so only one dynamic NonStop SQL/MP statement is needed. The remainder of the supplied dynamic statements (about 20) are available for pass-through commands. The limit of 20 is acceptable for pass-through commands.

ODBC supports prepare-and-execute operation, so the NonStop ODBC Server needs a different scheme, as a client can have several concurrently prepared statements. The NonStop ODBC Server still translates client SQL statements in the original way: it parses the statements, translates names and literal constants, makes syntax alterations, and forms a NonStop SQL/MP statement.

To support potentially large numbers of prepared statements, the NonStop ODBC Server uses the Extended Dynamic SQL of NonStop SQL/MP, whereby statements can be dynamically allocated and freed (the NonStop ODBC Server can sustain an arbitrary number of allocated and prepared statements as a function of allocated memory only. Examples of over 1600 statements have been achieved). For performance reasons, NonStop SQL/MP is executed directly from the NonStop ODBC server.

Figure 2-33. NonStop SQL/MP Execution in SQLCI and NonStop ODBC Server



SQLCI uses a separate SQLCI2 process for SQL statement execution, so a NonStop SQL/MP statement can be interrupted at any time (by stopping the SQLCI2 process). Therefore, a user can issue a Cancel request at any time. The NonStop ODBC Server partially supports a Cancel request by checking for the Cancel between issuing NonStop SQL/MP FETCH statements. If each FETCH statement involves little work, the client effectively has Cancel support. If each FETCH statement involves much work (such as a large sort), the user must stop the NonStop ODBC Server to interrupt execution.

Caching NonStop SQL/MP Compilations

To improve performance, the NonStop ODBC Server caches NonStop SQL/MP compilations. Using this strategy, the NonStop ODBC Server saves NonStop SQL/MP compilations in such a way that subsequent SQL execution can be done directly without the NonStop SQL/MP compile step.

The NonStop ODBC Server determines during statement translation which statements can be cached (also configurable within the profile for a user). If a statement can be cached—and caching is enabled for that user—the NonStop ODBC Server always checks the cache first before issuing a NonStop SQL/MP compilation instruction. The key for the cache is the NonStop SQL/MP statement itself, with extra blanks removed (“Select * from T” and “Select   * From   T are equivalent”).

Only DML (SELECT, INSERT, UPDATE, and DELETE) statements are eligible for caching. To improve caching, performance literals in WHERE expression clauses are translated into parameters if the operator is “=”. The other operators cannot be automatically converted because the optimizer views them as having different sensitivity according to the test value, causing wide ranging and not always valid query plans to be generated.

The basic caching algorithm follows:

- Prepared statements

  Client-prepared DML statements are cached as is.

- Execute-direct, with parameters

  Client execute-direct statements with parameters are cached as is.

- Execute-direct, without parameters

  Client execute-direct statements without parameters are first parameterized, then cached. Parameterizing involves changing some literal constants in the statement to dynamic parameters, represented by question marks (?). The NonStop ODBC Server parameterizes only when it seems useful and avoids doing so when no benefits are apparent.

In the following example of parameterizing, a client-issued statement

```
Insert into T Values (123, "abc")
```

would be changed to

```
Insert into T Values (?,?)
```

In the next example, the statement

```
Select * from T where col1 > 20 and col2 = 30
```

would be changed to

```
Select * from T where col1 > 20 and col2 = ?
```

In the second example, the literal constant 30 was changed to a parameter, but the literal constant 20 was not changed. If the 20 were changed to ?, NonStop SQL/MP might choose a different access plan.

Stored Procedure Execution

The NonStop ODBC Server supports Stored Procedure Execution (SPE) using ServerClass_Send to a Pathway server program. When the client issues an SPE SQL call, the NonStop ODBC server processes the call by referring to a NonStop ODBC Server catalog table and mapping the stored procedure name to a Pathway system and server class name. Stored procedure execution is described in detail in Section 5, Stored Procedures.

3 CORE SQL Language

Applications used with the HP NonStop ODBC Server can contain CORE SQL statements. The NonStop ODBC Server accepts these CORE SQL statements, translates them to HP NonStop SQL/MP statements, and sends the statements to NonStop SQL/MP. The NonStop ODBC Server can also execute some NonStop SQL/MP statements directly.

This section provides the following information:

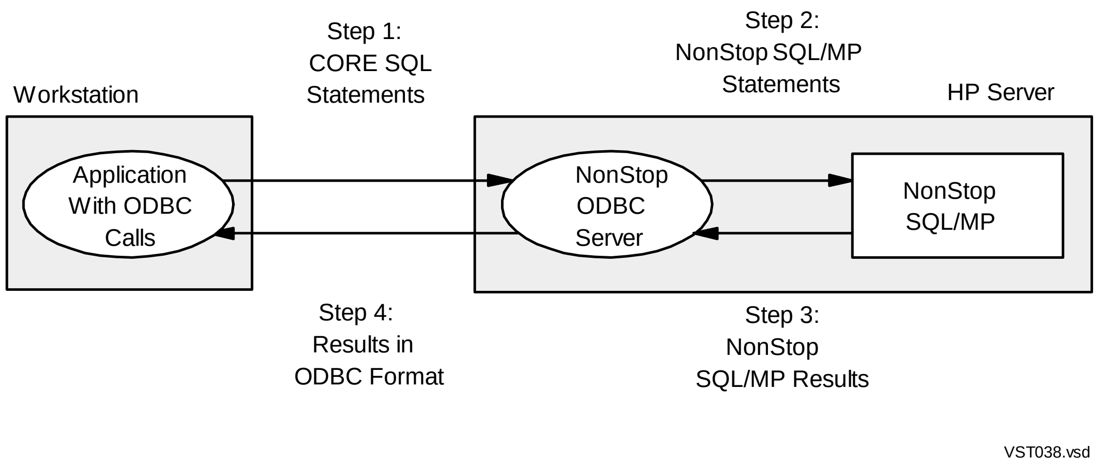
- Describes the language elements, such as names, data types, functions, and variables, used to construct CORE SQL statements.
- Summarizes the supported CORE SQL statements.
- Alphabetically lists and describes the CORE SQL statements that you can use in programs that interact with the NonStop ODBC Server.
- Gives examples of each supported statement.
- Explains the differences for each supported statement between the NonStop SQL/MP statement and the NonStop ODBC Server implementation of the statement.

NonStop ODBC Server Translation

Application programs used with the NonStop ODBC Server contain ODBC calls and CORE SQL statements.

Figure 3-1 illustrates the relationships among ODBC, CORE SQL, the NonStop ODBC Server, and NonStop SQL/MP.

Figure 3-1. Relationship Among CORE SQL, the NonStop ODBC Server, and NonStop SQL/MP


Workstation
Application With ODBC Calls
Step 1: CORE SQL Statements
Step 2: NonStop SQL/MP Statements
HP Server
NonStop ODBC Server
NonStop SQL/MP
Step 3: NonStop SQL/MP Results
Step 4: Results in ODBC Format
VST038.vsd

A CORE SQL statement is submitted to NonStop SQL/MP as follows:

1. An application program issues an ODBC call that sends a CORE SQL statement to the NonStop ODBC Server.
2. The NonStop ODBC Server translates the CORE SQL statement to a NonStop SQL/MP statement and sends the statement to NonStop SQL/MP.
3. NonStop SQL/MP processes the statement and returns NonStop SQL/MP data (or a diagnostic message) to the NonStop ODBC Server.
4. The NonStop ODBC Server translates the NonStop SQL/MP output to CORE SQL format and sends the result to the application program.

The application then retrieves the data using ODBC calls.

Unsupported ODBC Features

In this manual, unsupported ODBC features are underlined. For example, in the following syntax, the RESTRICT clause is unsupported. See the “Notation Conventions” for other conventions used in syntax representations.

```
DROP TABLE base-table-name

   [ CASCADE | RESTRICT ]
```

Language Elements

Language elements differ for CORE SQL and NonStop SQL/MP. This subsection describes the following CORE SQL language elements when using the NonStop ODBC Server:

- Names
- Data types
- Escape clauses
- Functions
- Search conditions
- Expressions and operators
- Aggregates
- Wild-card characters
- NULL values

Names

When using the NonStop ODBC Server, the names you specify for objects such as databases, tables, indexes, and columns must be in CORE SQL syntax; however, there are restrictions on some names, because the NonStop ODBC Server maps some names to Guardian names.

This subsection summarizes name usage, then describes the following types of names:

- Database names
- Owner names
- Table, view, and index names
- Procedure names
- Column and correlation names

Summary of Name Usage

In CORE SQL, most names are CORE SQL identifiers. NonStop SQL/MP does not have corresponding identifiers. Therefore, the NonStop ODBC Server maps these identifiers to NonStop SQL/MP simple names or Guardian names.

A CORE SQL identifier is a user-defined name, not longer than 60 characters, however, ODBC Server allows a maximum of only 30 characters. A CORE SQL identifier begins with a letter; the remaining characters can be letters, digits, or underscore (_) characters. The types of user-defined names are:

```
base-table-identifier
column-identifier
correlation-name
cursor-name
index-identifier
procedure-name-identifier
username
view-table-identifier
database-name
```

The *database-name* is not defined by CORE SQL, but is an extension of the NonStop ODBC Server to support three-part names, available in ISO SQL. The following constructs show valid examples of three-part names:

Base-table-name	`[ [database.]owner. ]`
	`[ database..        ]`
		base-table-identifier

View-table-name	`[ [database.]owner. ]`
	`[ database..        ]`
		view-table-identifier

Table-name	`[ [database.]owner. ]`
	`[ database..        ]`
		table-identifier

```
Index-name       [ [database.]owner. ]  index-identifier
                 [ database..        ]

Procedure-name   [ [database.]owner. ]  procedure-identifier
                 [ database..        ]

Column-name      [ {table-name      }. ]  column-identifier
                 [ {correlation-name}  ]
```

In the *`table-name`* shown above, *`table-identifier`* can be either a *`base-table-identifier`* or a *`view-table-identifier`*.

Table 3-1 summarizes name format and how the NonStop ODBC Server handles names.

Table 3-1. How the NonStop ODBC Server Maps CORE SQL Object Names

Name Type	**Name Format**	**How NonStop ODBC Server Handles Name**
Database name	*nodevolumesubvolume*	Maps it to \node.$*volume*.*subvolume*, which corresponds to a NonStop SQL/MP catalog.
Owner name	logical-username	Maps it to *group-name*.*user-name*, which corresponds to a Guardian logon name.
Table name	CORE SQL identifier*	Maps it to an 8-character Guardian file name. Might resort to generating a random name.
View name	CORE SQL identifier*	Maps it to an 8-character Guardian file name. Might resort to generating a random name.
Index name	CORE SQL identifier*	Maps it to an 8-character Guardian file name. Might resort to generating a random name.
Procedure name	CORE SQL identifier	Maps it to a Pathway server class.
Column name	CORE SQL identifier	Maps it to a NonStop SQL/MP simple name.
Correlation name	CORE SQL identifier	No mapping is required.

*The NonStop ODBC Server allows a maximum of 30 characters for table names, view names, and index names.

All names created using the NonStop ODBC Server are converted to uppercase letters before any SQL statement processing occurs. Therefore, names are not case sensitive; “My_Table” is considered the same as “my_table” or “MY_TABLE.”

Table 3-2 summarizes the rules for creating CORE SQL identifiers, NonStop SQL/MP simple names, and Guardian names.

Table 3-2. Rules for Object Names

	CORE SQL Identifier	NonStop SQL/MP Simple Name	Guardian Name			
			Node Name	Volume Name	Subvolume Name	Object Name
Length (in characters)	1–60*	1–30	1–8	1–7	1–8	1–8
First character	Letter	Letter	Backslash (\\)	$	Letter	Letter
Remaining characters	Letters Digits Underscore (_)	Letters Digits Underscore (_)	Letters Digits	Letters Digits	Letters Digits	Letters Digits
Example	table_a	second_val	\finance \system	$persnl $disk01	pay1994	table1

* The NonStop ODBC Server allows a maximum of 30 characters for table names, view names, and index names.

For more information about identifiers, see the following documents:

For Information About	See
CORE SQL identifiers	X/Open *CAE Specification*
NonStop SQL/MP identifiers	*NonStop SQL/MP Reference Manual*

Database Names

A logical database name can be any valid NonStop SQL/MP identifier:

You must separate the parts of the name with underscores (_).

Considerations for database names are as follows:

- Each portion of the name begins with a letter and consists of letters and digits. The maximum number of characters for each portion is as follows:

Portion	**Characters**
Node	7
Volume	6
Subvolume	8

- The NonStop ODBC Server maps the database name to a Guardian name as follows:

```
\node.$volume.subvolume
```

  For example, if you specify the database name CORP_VOL2_SALES, the NonStop ODBC Server maps it to the Guardian name\CORP.$VOL2.SALES, which identifies the catalog SALES on the volume $VOL2 on the node \CORP.

  For more information about Guardian names, see the *Guardian User's Guide*.

Owner Names

Each SQL object belongs to an owner. An owner name can be any valid NonStop SQL/MP identifier and is identified by one of the following:

- A logical username associated with a Guardian logon name by an ADD USER statement.
- A logical username associated with an alias username (and with a Guardian logon name) by an ADD ALIAS statement.

Considerations for owner names are as follows:

- An owner name is a CORE SQL identifier.
- An owner name is stored in the mapping tables with the associated object name. When you create an object under a specific owner name, you must reference that object using the owner name (or the associated logical username for an alias username) as a qualifier.

Note. Do not specify another owner when you create objects. If you do so, the other owner is listed in the NonStop ODBC Server mapping tables, while you are the owner of the object.

- You specify an owner name as part of an index, table, view, or stored procedure name as follows:

  ```
  [ [database.]owner.] ]  object
  [ database..         ]
  ```

  If you include the database name but omit the owner name, you must include an extra period to show the omission.

- All object-name references are fully qualified before the SQL statement is executed. If you do not specify an owner name, the logical username established at connection time is used to qualify the object. If the database name is missing, the default database indicated in the ZNSPROF profile record for the current logical username is used; if no default database has been set in ZNSPROF, MASTER is used as the database name.

Some additional considerations for alias usernames are as follows:

- They can be used in place of their associated logical usernames for logging on.
- They cannot be used for qualifying object names.
- They are system-wide and not restricted to a given database.
- If the owner name is missing from an object reference, the logical username associated with the alias username is used.

The NonStop ODBC Server authenticates a logon username as follows:

1. NOSUTIL first checks the length of the logon username. If the name is longer than 17 characters, it cannot be a Guardian name, so NOSUTIL sets the corresponding Guardian username in G_USERNAME to NULL. If the name is 17 or fewer characters, NOSUTIL does not set G_USERNAME.
2. The NonStop ODBC server then searches the username alias table, ZNSALT, for a mapping entry for the logon username:
   - If a mapping entry is found, the NonStop ODBC server determines the PROFILE_NAME and NOS_USERNAME from the ZNSALT table and the corresponding Guardian username (G_USERNAME) from the ZNSUS table. The NonStop ODBC server uses this information and the corresponding password to authenticate the user.
   - If a mapping entry is not found, the NonStop ODBC server does not consider the user to be an ODBC user. The NonStop ODBC server tries to authenticate the user; however, even if the authentication is successful, it does not allow the user to perform DDL operations.
3. If the authentication fails and the logon username is 17 or fewer characters, the NonStop ODBC server converts the name to a Guardian format name (`group.user`) by substituting the first underscore with a period. The NonStop ODBC server then tries to authenticate the user using the converted Guardian name.

Note. If Safeguard is in effect for the system and the username is an alias, the NonStop ODBC server always uses the alias name for the logon. If the username is not an alias, the NonStop ODBC server uses the Guardian name for the logon. If Safeguard is not in effect, the NonStop ODBC server always uses the Guardian name for the logon.

For more information, see the following documents:

For Information About	See
CORE SQL usernames	X/Open *CAE Specification*
Name mapping	Section 7, Managing Customized Catalogs
Guardian logon name	*Guardian User's Guide*

Table, View, and Index Names

For table, view, and index names, specify CORE SQL identifiers. The NonStop ODBC Server maps these identifiers to eight-character Guardian file names. Even though the names are mapped, you refer to the object by its CORE SQL name when accessing it using the NonStop ODBC Server.

Procedure Names

For procedure names, you specify CORE SQL identifiers. The NonStop ODBC Server maps these identifiers to the names of Pathway server class programs. Even though the names are mapped, you refer to the object by its CORE SQL name when accessing it using the NonStop ODBC Server.

Column and Correlation Names

For column names and correlation names, you specify CORE SQL identifiers. Mapping is not necessary, so you refer to the object by its CORE SQL name when accessing it using the NonStop ODBC Server.

Data Types

When creating and manipulating objects and data items, you must specify CORE SQL data types. Because CORE SQL and NonStop SQL/MP data types differ, the data types or values you specify are converted to corresponding NonStop SQL/MP data types.

Even if you are retrieving or manipulating data created with NonStop SQL/MP, you must specify CORE SQL, not NonStop SQL/MP, data types and values.

This subsection summarizes NonStop ODBC Server support of CORE SQL data types. Data type support falls in two categories:

- Creating objects and data
- Retrieving and manipulating NonStop SQL/MP objects and data

Creating Objects and Data

When you create objects and data, you specify CORE SQL data types and values. The NonStop ODBC Server handles the data type or value in one of two ways:

- Fully supports it—the data type or value is processed as you input it.
- Converts it—the data type or value is converted to a corresponding NonStop SQL/MP data type or value.

Table 3-3 summarizes the CORE SQL data types, the corresponding NonStop SQL/MP data types, and the conversion information.

Table 3-3. Conversion of CORE SQL Data Types to NonStop SQL/MP Data Types

CORE SQL Data Type	NonStop ODBC Server Support	Corresponding NonStop SQL/MP Data Type
CORE SQL Data Type		
CHAR	x	CHAR
DECIMAL	x	DECIMAL
DOUBLE PRECISION	x	DOUBLE PRECISION
FLOAT	x	FLOAT
INTEGER	x	INTEGER
NUMERIC	x	NUMERIC
REAL	x	REAL
SMALLINT	x	SMALLINT
VARCHAR	x	VARCHAR
Extended SQL Data Types		
BIGINT	x	LARGEINT
BINARY	x	CHAR
BIT	x	SMALLINT
DATE	x	DATETIME year to day
LONG VARBINARY	x	VARCHAR
LONG VARCHAR	x	VARCHAR
TIME	x	DATETIME hour to second
TIMESTAMP	x	DATETIME hour to fraction(6)
TINYINT	x	SMALLINT
VARBINARY	x	VARCHAR

x Indicates a supported data type

The NonStop ODBC Server supports the ODBC Extended SQL data types as well as those provided at the Core level.

ODBC describes the minimum limit that must be met by any conformant ODBC implementation, and implements a call interface by which an application can detect the actual implementation-defined limit. The NonStop ODBC Server satisfies each of the ODBC core-level minimums, and in most cases exceeds those minimums.

For more information, see the following documents:

For Information About	See
CORE SQL data types	*Microsoft ODBC 2.1 Programmer's Reference and SDK Guide*
NonStop SQL/MP data types	*NonStop SQL/MP Reference Manual*

Retrieving and Manipulating NonStop SQL/MP Data and Objects

When you access data and objects, the NonStop ODBC Server converts NonStop SQL/MP data types and values to CORE SQL data types and values. Table 3-4 summarizes the NonStop SQL/MP data types, corresponding CORE SQL data types, and conversion information.

Table 3-4. Conversion of NonStop SQL/MP Data Types to CORE SQL Data Types (page 1 of 2)

NonStop SQL/MP Data Type	Corresponding CORE SQL Data Type
CHAR	CHAR
DATE	DATE
DATETIME *qualifier*	DATE TIME TIMESTAMP
DECIMAL	DECIMAL
DOUBLE PRECISION	DOUBLE PRECISION
FLOAT	FLOAT
INTEGER	INTEGER
INTEGER unsigned	BIGINT
INTERVAL	CHAR
LARGEINT	BIGINT
NUMERIC	NUMERIC
REAL	REAL
SMALLINT	SMALLINT
SMALLINT unsigned	INTEGER
TIME	TIME

Table 3-4. Conversion of NonStop SQL/MP Data Types to CORE SQL Data Types (page 2 of 2)

NonStop SQL/MP Data Type	Corresponding CORE SQL Data Type
TIMESTAMP	TIMESTAMP
VARCHAR	VARCHAR
Others	CHAR or VARCHAR, as appropriate

DATETIME data can be converted to DATE, TIME, or TIMESTAMP, depending on the range of the *qualifier*. For example, DATETIME YEAR TO MONTH is converted to DATE; DATETIME HOUR TO HOUR is converted to TIME; DATETIME DAY TO SECOND is converted to TIMESTAMP. Any missing fields are zero-filled; for example, DATETIME “1922-03” YEAR TO MONTH is returned as DATE data type with the value “1922-03-00.”

INTERVAL data type is displayed as a character type, in ANSI format.

Any characters not in the ISO 8859.1 character set (Kanji and the like) are converted to CHAR or VARCHAR. It is the application’s responsibility to translate them.

Escape Clauses

ODBC defines the following entities as extensions to SQL:

- Date, time, and timestamp data
- Scalar functions such as numeric, string, and type conversion functions
- Outer joins
- Stored procedures

To be able to include one of these extensions in a CORE SQL statement, you must enclose it in an **escape clause**. An escape clause has the following syntax:

```
--*( VENDOR(vendor-name), PRODUCT(product-name) extension )*-
-
```

vendor-name

is the name of the company that makes the product that offers the extension.

product-name

is the name of the product that provides the extension.

Because in CORE SQL the vendor’s name is always “Microsoft,” and the product name is always “ODBC,” you can use a shorthand form of escape-clause syntax that omits those specifications, as follows:

```
{ extension }
```

Note. The braces ({ and }) in the preceding syntax do not follow the standard syntax conventions indicating choice; they are actual characters you must enter.

Escape Clauses for Date and Time Data

For date, time, and timestamp data, described previously, ODBC uses the following escape clauses:

```
--*( VENDOR(Microsoft), PRODUCT(ODBC) D value )*--

--*( VENDOR(Microsoft), PRODUCT(ODBC) T value )*--

--*( VENDOR(Microsoft), PRODUCT(ODBC) TS value )*--

    or

{ D 'value' }

{ T 'value' }

{ TS 'value' }
```

`D`

specifies that `value` is a date in the "*yyyy-mm-dd*" format.

`T`

specifies that `value` is a time in the "*hh:mm:ss*" format.

`TS`

specifies that `value` is a timestamp in the "*yyyy-mm-dd hh:mm:ss*[.*fff*[*fff*]]" format.

For example, each of the following statements updates the birthday of John Smith in the EMPLOYEE table:

```
update employee
   set birthday --*(vendor(Microsoft),product(ODBC) d '1939-01-27')*--
   where name="Smith, John"

update employee
   set birthday {d '1939-01-27'}
   where name="Smith, John"
```

Escape Clauses for Scalar Functions

When you include any scalar function (described under Functions on page 3-14) in a CORE SQL statement, the escape clause used has the following format:

```
--*( VENDOR(Microsoft), PRODUCT(ODBC) FN scalar-function )*--

     or

{ FN scalar-function }
```

`FN`

specifies that what follows is the invocation of a scalar function.

For example, the following SELECT statement specifies that string values in the NAME column are to be shifted to uppercase letters:

```
select {fn ucase(name)} from employee
```

Escape Clauses for Outer Joins

ODBC uses the following form of escape clause for specifying an outer join:

```
--*( VENDOR(Microsoft), PRODUCT(ODBC)

   OJ table-reference LEFT [ OUTER ] JOIN table-reference

   ON search-condition )*--

     or

{ OJ table-reference LEFT [ OUTER ] JOIN table-reference

  ON search-condition }
```

`oj`

specifies that what follows is an outer join specification.

For example, the following SELECT statement specifies a join condition between two table references:

```
select employee.name,dept.deptname
   from {oj employee left join dept
         on employee.deptid=dept.deptid}
   where employee.projid=544
```

Escape Clauses for Executing Stored Procedures

ODBC uses the following form of escape clause for executing stored procedures:

```
--*( VENDOR(Microsoft),PRODUCT(ODBC)
   [ ?= ] CALL procedure-name [ ( parameter [, ... ] ) ] )*--
     or
{ [ ?= ] CALL procedure-name [ ( parameter [, ... ] ) ] }
```

`CALL`

specifies that this is a call to a previously established procedure (a Pathway server class) for execution of that procedure.

For example, the following CALL statement invokes a stored procedure named SELECT_DEPT3000:

```
{ call select_dept3000 }
```

Functions

CORE SQL provides several types of functions for manipulating and retrieving data. This subsection summarizes NonStop ODBC Server support of CORE SQL functions, then describes, by the following function types, how to use the supported functions:

- Date functions
- Mathematical functions
- String functions
- System functions
- Data type conversion function

Summary of CORE SQL Functions

Table 3-5 summarizes the supported CORE SQL functions. Descriptions of each function follow the table.

Table 3-5. Supported CORE SQL Functions (page 1 of 2)

Function Type	Function	Comments
Date	CURDATE	See CURDATE on page 3-16
	CURTIME	See CURTIME on page 3-16
	DAYOFMONTH	See DAYOFMONTH on page 3-16
	DAYOFWEEK	See DAYOFWEEK on page 3-16

Table 3-5. Supported CORE SQL Functions (page 2 of 2)

Function Type	Function	Comments
	HOUR	See HOUR on page 3-16
	MINUTE	See MINUTE on page 3-17
	MONTH	See MONTH on page 3-17
	NOW	See NOW on page 3-17
	QUARTER	See QUARTER on page 3-17
	SECOND	See SECOND on page 3-17
	TIMESTAMPADD	See TIMESTAMPADD on page 3-17
	TIMESTAMPDIFF	See TIMESTAMPDIFF on page 3-18
	YEAR	See YEAR on page 3-19
Mathematical	EXP	Fully supported
	MOD	Fully supported
	PI	Fully supported
String	UCASE	Fully supported
System	DATABASE	Fully supported
	USER	Fully supported
Conversion	CONVERT	SeeConversion Function on page 3-23

For a complete list of the CORE SQL functions, see Appendix A, Summary of Support for ODBC Features.

Date Functions

The NonStop ODBC Server supports the following CORE SQL date functions, which manipulate datetime data:

- CURDATE
- CURTIME
- DAYOFMONTH
- DAYOFWEEK
- HOUR
- MINUTE
- MONTH
- NOW
- QUARTER
- SECOND

- TIMESTAMPADD
- TIMESTAMPDIFF
- YEAR

In all of the syntax descriptions that follow, *`date-expression`* or *`time-expression`* can be a date or time literal, a column name, or the result of another function.

CURDATE

The CURDATE function has the following syntax:

```
CURDATE ( )
```

The CURDATE function returns the current date.

CURTIME

The CURTIME function has the following syntax:

```
CURTIME ( )
```

The CURTIME function returns the current time.

DAYOFMONTH

The DAYOFMONTH function has the following syntax:

```
DAYOFMONTH ( date-expression )
```

The DAYOFMONTH function evaluates *`date-expression`* and returns the corresponding day of the month as an integer value.

DAYOFWEEK

The DAYOFWEEK function has the following syntax:

```
DAYOFWEEK ( date-expression )
```

The DAYOFWEEK function evaluates *`date-expression`* and returns the corresponding day of the week as an integer value, where 1 represents Sunday.

HOUR

The HOUR function has the following syntax:

```
HOUR ( time-expression )
```

The HOUR function returns the hour in *`time-expression`* as an integer.

MINUTE

The MINUTE function has the following syntax:

```
MINUTE ( time-expression )
```

The MINUTE function returns the minute in *time-expression* as an integer.

MONTH

The MONTH function has the following syntax:

```
MONTH ( date-expression )
```

The MONTH function returns the month in *date-expression* as an integer.

NOW

The NOW function has the following syntax:

```
NOW ( )
```

The NOW function returns the current datetime as a timestamp value.

QUARTER

The QUARTER function has the following syntax:

```
QUARTER (date-expression )
```

The QUARTER function returns the quarter of the year in the range 1 to 4, in which 1 represents January 1 through March 31.

SECOND

The SECOND function has the following syntax:

```
SECOND ( time-expression )
```

The SECOND function returns the second in *time-expression* as an integer.

TIMESTAMPADD

The TIMESTAMPADD function has the following syntax:

```
TIMESTAMPADD ( interval, integer-exp, timestamp-exp )
```

The TIMESTAMPADD function adds a number of intervals to a timestamp value.

`interval`

is the interval to be added to the timestamp. It is one of the following INTERVAL keywords:

- SQL_TSI_FRAC_SECOND
- SQL_TSI_SECOND
- SQL_TSI_MINUTE
- SQL_TSI_HOUR
- SQL_TSI_DAY
- SQL_TSI_MONTH
- SQL_TSI_YEAR

Fractional seconds are expressed in millionths of a second (microseconds).

`integer-exp`

is the number of intervals to be added. It is the name of a column, the result of another scalar function, or a numeric literal to be added to *`timestamp-exp`*.

`timestamp-exp`

is the timestamp to be incremented. It is the name of a column; the result of another scalar function; or a time, date, or timestamp literal.

If *`timestamp-exp`* is a time value and *`interval`* is days, weeks, months, quarters, or years; the data portion of *`timestamp-exp`* is set to the current date before the resulting timestamp is calculated.

If *`timestamp-exp`* is a date value and *`interval`* is fractional seconds, seconds, minutes, or hours, the time portion of *`timestamp-exp`* is set to 0 before the resulting timestamp is calculated.

The following SQL statement returns the names of employees and their one-year anniversary dates:

```
select name, {fn timestampadd (SQL_TSI_YEAR, 1, hire_date )}
   from employees
```

TIMESTAMPDIFF

The TIMESTAMPDIFF function has the following syntax:

```
TIMESTAMPDIFF ( interval, timestamp-exp1, timestamp-exp2 )
```

TIMESTAMPDIFF returns the number of intervals by which one time is greater than another time.

interval

is the interval by which *timestamp-exp2* is greater than *timestamp-exp1*. Valid values of *interval* are the following keywords:

- SQL_TSI_FRAC_SECOND
- SQL_TSI_SECOND
- SQL_TSI_MINUTE
- SQL_TSI_HOUR
- SQL_TSI_DAY
- SQL_TSI_MONTH
- SQL_TSI_YEAR

Fractional seconds are expressed in millionths of a second (microseconds).

timestamp-exp1

is the first timestamp expression in the comparison. It is the name of a column; the result of another scalar function; or a time, date, or timestamp literal.

timestamp-exp2

is the second timestamp expression in the comparison. It is the name of a column. The date portion of this timestamp is set to the current date before the resulting timestamp is calculated.

If either timestamp expression is a time value and *interval* is days, weeks, months, quarters, or years; the data portion of *timestamp-exp* is set to the current date before the resulting timestamp is calculated.

If either timestamp expression is a date value and *interval* is fractional seconds, seconds, minutes, or hours; the time portion of that timestamp expression is set to 0 before the resulting timestamp is calculated.

The following SQL statement returns the names of employees and the number of years they have been employed:

```
select name, {fn timestampdiff (SQL_TSI_YEAR,
  { fn curdate() }, hire_date)} from employees
```

YEAR

The YEAR function has the following syntax:

```
YEAR ( date-expression )
```

The YEAR function returns the current date.

For more information, see the following documents:

For Information About	**See**
CORE SQL date functions	*Microsoft ODBC 2.1 Programmer's Reference and SDK Guide*
Datetime data	Data Types on page 3-8

Mathematical Functions

The NonStop ODBC Server supports the following CORE SQL mathematical functions:

- EXP
- MOD
- PI

EXP

The EXP function has the following syntax:

```
EXP ( floating-expression )
```

The EXP (exponential) function returns the value of `floating-expression` multiplied by the mathematical constant *e* (2.1718...).

MOD

The MOD function has the following syntax:

```
MOD ( int-expression-1 , int-expression-2 )
```

The MOD function returns the remainder (modulus) expressed by:

int-expression-1 – (*int-expression-1* / *int-expression-2*) * *int-expression-2*

PI

The PI function has the following syntax:

```
PI ( )
```

The PI function returns the value 3.1415926535897936.

String Functions

The NonStop ODBC Server supports the following CORE SQL string functions:

- CONCAT
- LENGTH
- LOCATE
- LTRIM
- RTRIM
- SUBSTRING
- UCASE

CONCAT

```
CONCAT ( string-expression1, string-expression2 )
```

The CONCAT function returns a character string that is the result of concatenating *`string-expression2`* to *`string-expression1`*. For example:

```
SELECT { fn CONCAT(first_name, last_name) FROM employee
```

or

```
SELECT --(*vendor(Microsoft),product(ODBC)
          fn CONCAT(first_name, last_name)*)-- FROM employee
```

LENGTH

```
LENGTH ( string-expression )
```

The LENGTH function returns the number of characters in *`string-expression`*, excluding trailing blanks. For example:

```
SELECT last_name FROM employee
  WHERE { fn LENGTH(last_name) } = 10
```

LOCATE

```
LOCATE ( string-expression1, string-expression2 )
```

The LOCATE function returns the starting position of the first occurrence of *`string-expression1`* within *`string-expression2`*. The search begins with the first position in *`string-expression2`* . For example:

```
SELECT * FROM dept
  WHERE { fn LOCATE("Marketing",deptname) } > 0
```

LTRIM

```
LTRIM ( string-expression )
```

The LTRIM function returns the characters of *`string-expression`* with any leading blanks removed. For example:

```
SELECT * FROM employee WHERE { fn LTRIM(last_name) } = "Wang"
```

RTRIM

```
RTRIM ( string-expression )
```

The RTRIM function returns the characters of *`string-expression`* with any trailing blanks removed. See LTRIM for an example.

SUBSTRING

```
SUBSTRING ( string-expression, start, length )
```

The SUBSTRING function returns a character string from *`string-expression`* beginning with *`start`* for *`length`* characters. For example:

```
SELECT * FROM employee
  WHERE { fn SUBSTRING(first_name,1,4) } = "Bill"
```

UCASE

```
UCASE ( string-expression )
```

The UCASE function returns *`string-expression`* shifted to uppercase letters. For example:

```
SELECT * FROM employee
  WHERE { fn UCASE(last_name) } = "SMITH"
```

System Functions

The NonStop ODBC Server supports the following CORE SQL system functions:

- DATABASE
- USER

DATABASE

The DATABASE function has the following syntax:

```
DATABASE ( )
```

The DATABASE function returns the current database name. The current database reflects the user's current session values. These values are set at the time the user session is first established, using values from the associated profile (ZNSPROF) and the user's login request. The current values change during the course of a session when the user performs actions to set or reset the current environment. The value returned by DATABASE is always a valid database name as determined by its existence in the system mapping table ZNSDB.

USER

The USER function has the following syntax:

```
USER ( )
```

The USER function returns the logical username of the current user. The current user is the user who is logged on.

Conversion Function

The NonStop ODBC Server supports the CORE SQL conversion function, CONVERT. The CONVERT function has the following syntax:

```
CONVERT ( value-expression , data-type )
```

`data-type`

the data type into which the expression is to be converted. It can be any valid ODBC data type except a binary type.

`value-expression`

The value to be converted.

The CONVERT function returns the value of *`value-expression`*, converted to the data type specified by *`data-type`*.

If *`data-type`* is BIT or TINYINT, it is treated as SMALLINT type. No range checking is performed.

The CONVERT function cannot appear in the view definition.

Converting from a Datetime Value to a Character Value

When converting a datetime value to a character value, the result is displayed in the format year to fraction 6, as shown in the following example:

```
select {fn convert ({fn now()},char(26))} ...
```

Year to fraction 6 values require 26 characters, so the *`data-type`* must be at least 26 characters long.

For information about year to fraction data, see the *NonStop SQL/MP Reference Manual*.

Converting a Character Column Value to a Datetime Value

When converting a character column value to a datetime value, the format of the data in the column must be year to fraction 3. For example, if the START_DATE column in the following example is a character field, the data must be in the form year to fraction 3 (such as “1994-01-05:12:29:00.123”).

```
select {fn convert (table1.start_date,timestamp)} ...
```

Blank Padding of Character Data

When you convert a character expression to character data of a longer size, the value is blank-padded on the right.

Truncating Significant Digits

When converting between types with a different number of decimal points, the value is truncated if trailing significant digits will be lost. If, however, leading significant digits will be lost, an error message is issued and the statement does not execute.

Search Conditions

Use CORE SQL syntax in search conditions when using the NonStop ODBC Server. All supported search conditions work the same as in CORE SQL.

The syntax of a search condition is any of the following:

```
expression-1 comparison-operator  { expression-2 }
                                  { (subquery)   }

expression-1 [ NOT ] BETWEEN expression-2
  AND expression-3

expression comparison-operator  { ANY }  ( subquery )
                                { ALL }

expression [ NOT ] IN (  { value-list }  )
                         { subquery   }

[ NOT ] column-name [ NOT ] LIKE "match-string"

[ NOT ] column-name IS [ NOT ] NULL

[ NOT ] EXISTS ( subquery )

boolean-expression

expression   { AND }  expression
             { OR  }
```

For more information about search conditions, see the X/Open *CAE Specification*.

Expressions and Operators

Use CORE SQL syntax in expressions when using the NonStop ODBC Server.

The syntax of an expression is:

```
operand [ arithmetic-operator operand
   [ arithmetic-operator operand ] ... ]

operand is:

   { column-name                      }
   { dynamic-parameter                }
   { literal                          }
   { ODBC-scalar-function-extension }
   { aggregate-function-reference   }
   { USER)                            )

arithmetic-operator is:

   ( + }
   { - }
   { * }
   { / }
```

dynamic-parameter

is represented by a question mark (?); it identifies a parameter in a dynamically prepared SQL statement

literal

is an exact representation of a numeric or character value.

USER

is a keyword that represents a character string containing a logical username.

The *column-name* is described under Names on page 3-3 and *ODBC-scalar-function-extension* is described under Escape Clauses on page 3-11; *aggregate-function-reference* is described under Aggregates on page 3-27.

The syntax of a Boolean expression is as follows:

```
expression comparison-operator  [ ANY ]  expression
                                [ ALL ]
expression [ NOT ] IN expression

[ NOT ] EXISTS expression

expression [ NOT ] BETWEEN expression AND expression

expression [ NOT ] LIKE expression

NOT expression LIKE expression

expression IS [ NOT ] NULL

NOT boolean-expression

boolean-expression AND/OR boolean-expression

[ NOT ] boolean-function
```

Table 3-6 summarizes the supported CORE SQL operators.

Table 3-6. NonStop ODBC Server Support of CORE SQL Operators

CORE SQL Operator Type	Symbol	NonStop ODBC Server Support
Arithmetic	+	x
	-	x
	*	x
	/	x
Comparison	=	x
	<>	x
Comparison	>	x
	>=	x
	<	x
	<=	x

Arithmetic operators in an expression are evaluated according to their order of precedence. The following list of operators is arranged in descending order:

+, –	(unary operators)
*, /	multiplication, division
+, –	addition, subtraction

Aggregates

The NonStop ODBC Server supports all of the CORE SQL aggregate functions (AVG, COUNT, MAX, MIN, and SUM).

AVG

The AVG aggregate function has the following syntax:

```
AVG (  { DISTINCT column-name }  )
       { expression           }
```

The AVG function returns the average of the values in the column, in the same data type as the argument you supply.

If you specify DISTINCT, the AVG function calculates the result after excluding duplicate and null values of the argument. If you do not specify DISTINCT, duplicate values are retained; *expression* must contain at least one *column-name* and must not contain any aggregate functions.

COUNT

The COUNT aggregate function has the following syntax:

```
COUNT (  { DISTINCT column-name }  )
         { column-name          }
         { *                    }
```

The COUNT function returns the total number of values in the column, in numeric data type.

If only *column-name* is specified, the column data type must be a numeric data type such as NUMERIC and not a nonnumeric type such as CHAR.

If you specify DISTINCT, the COUNT function calculates the result after excluding duplicate values of the argument. COUNT (*) counts all values, including duplicate and null values.

(NonStop SQL/MP does not support the COUNT function when only *column-name* is specified in a query; however, NonStop ODBC has implemented the COUNT function for ODBC clients that need this function.)

MAX

The MAX aggregate function has the following syntax:

```
MAX (  { DISTINCT column-name }  )
       { expression           }
```

The MAX function returns the largest value in the column, in the same data type as the argument you supply.

If you specify DISTINCT, the MAX function calculates the result after excluding duplicate and null values of the argument. If you do not specify DISTINCT, duplicate values are retained; *expression* must contain at least one *column-name* and must not contain any aggregate functions.

MIN

The MIN aggregate function has the following syntax:

```
MIN (   { DISTINCT column-name }   )
        { expression           }
```

The MIN function returns the smallest value in the column, in the same data type as the argument you supply.

If you specify DISTINCT, the MIN function calculates the result after excluding duplicate and null values of the argument. If you do not specify DISTINCT, duplicate values are retained; *expression* must contain at least one *column-name* and must not contain any aggregate functions.

SUM

The SUM aggregate function has the following syntax:

```
SUM (   { DISTINCT column-name }   )
        { expression           }
```

The SUM function returns a numeric value that is the sum of the values in the column.

If you specify DISTINCT, the SUM function calculates the result after excluding duplicate values of the argument. If you do not specify DISTINCT, duplicate values are retained; *expression* must contain at least one *column-name* and must not contain any aggregate functions.

For more information, see the following documents:

For Information About	**See**
Aggregates in CORE SQL	X/Open *CAE Specification*
Aggregates in NonStop SQL/MP	*NonStop SQL/MP Reference Manual*

Wild-Card Characters

Wild-card characters can be used in the LIKE clause operand, which is a quoted string, to represent any character in a string. Table 3-7 summarizes NonStop ODBC Server support of CORE SQL wild-card characters.

Table 3-7. NonStop ODBC Server Support of Wild-Card Characters

CORE SQL Wild-Card Character	NonStop ODBC Server Support
%	x
_	x

Character comparisons are case sensitive. For example, the following comparison would locate the string “sanitary” but not the string “San Francisco.”

```
%san%
```

For more information, see the following documents:

For Information About	See
Wild-card characters in CORE SQL	X/Open *CAE Specification*
Wild-card characters in NonStop SQL/MP	*NonStop SQL/MP Reference Manual*

CASE Expression

The NonStop ODBC Server supports the CASE expression in queries. The syntax is the same as the NonStop SQL/MP syntax, except that you use NonStop ODBC logical names instead of NonStop SQL/MP names:

Simple CASE is:

```
CASE case-expression
  WHEN expression-1 THEN {result-expression-1 | NULL}
  WHEN expression-2 THEN {result-expression-2 | NULL}
  ...
  WHEN expression-n THEN {result-expression-n | NULL}
  [ELSE {result-expression | NULL}]
END
```

Searched CASE is:

```
CASE
  WHEN search-condition-1 THEN {result-expression-1 | NULL}
  WHEN search-condition-2 THEN {result-expression-2 | NULL}
  ...
  WHEN search-condition-n THEN {result-expression-n | NULL}
  [ELSE {result-expression | NULL}]
END
```

`case-expression`

specifies a value expression that is compared to the value expressions in each WHEN clause of a simple CASE. The data type of each *`expression`* in the WHEN clause must be comparable to the data type of *`case-expression`*.

`expression-1 ... expression-n`

specifies a value associated with each *`result-expression`*. If the value of an *`expression`* in a WHEN clause matches *`case-expression`*, then simple CASE returns the associated *`result-expression`* value. If there is no match, the CASE expression returns the value expression specified in the ELSE clause, or NULL if the ELSE value is not specified.

`result-expression-1 ... result-expression-n`

specifies the result value expression associated with each *`expression`* in a WHEN clause of a simple CASE, or with each *`search-condition`* in a WHEN clause of a searched CASE. All of the *`result-expressions`* must have comparable data types, and at least one of the *`result-expressions`* must return a nonnull.

`result-expression`

follows the ELSE keyword and specifies the value returned if none of the expressions in the WHEN clause of a simple CASE are equal to the case expression, or if none of the search conditions in the WHEN clause of a searched CASE are true. If the ELSE *`result-expression`* clause is not specified, then CASE returns NULL. The data type of *`result-expression`* must be comparable to the other results.

`search-condition-1 ... search-condition-n`

specifies conditions to test for in a searched CASE. If a *`search-condition`* is true, the CASE expression returns the associated *`result-expression`* value. If no *`search-condition`* is true, the CASE expression returns the value expression specified in the ELSE clause, or NULL if the ELSE value is not specified.

Examples

```
SELECT last_name, first_name,
  CASE jobcode
    WHEN 100 THEN "Manager"
    WHEN 250 THEN "Information Designer"
    WHEN 420 THEN "Software Engineer"
    WHEN 600 THEN "Database Administrator"
    ELSE NULL
   END
 FROM employee
```

```
SELECT last_name, first_name,
  CASE
    WHEN deptnum=1000 THEN salary * 1.1
    WHEN deptnum=2000 THEN salary * 1.5
    WHEN deptnum=3000 THEN salary * 1.8
    ELSE salary
```

```
   END
 FROM employee
```

For more information about the CASE expression, see the *NonStop SQL/MP Reference Manual*.

CORE SQL Statements

The SQL statements you can use with the NonStop ODBC Server are a subset of the ODBC CORE SQL statements. This section lists the supported CORE SQL statements and summarizes how each statement differs when used with the NonStop ODBC Server.

Ordinarily, if you use an unsupported statement with the NonStop ODBC Server, an error message is generated. You can establish a user profile option, however, to ignore errors from unsupported features. This option can be useful when you are running an application that uses a feature whose action is not required, such as GRANT and REVOKE.

Table 3-8. NonStop ODBC Server Support of SQL Statements (page 1 of 2)

CORE SQL Statement	**NonStop ODBC Server Support**	**Comments**
ALTER TABLE	x	Reduced syntax.
CALL	x	Fully supported
CREATE [UNIQUE] INDEX	x	You cannot create a unique index on a column that allows null values.
		The fully expanded index name must be unique in the network.
CREATE TABLE	x	The NonStop ODBC Server maps some data types to NonStop SQL/MP data types. See Data Types on page 3-8.
		The fully expanded table name must be unique in the network.
CREATE VIEW	x	You can modify data in a view only if the view is derived from one table.
		The fully expanded view name must be unique in the network.
DELETE	x	Fully supported.

x Indicates that the statement is supported
– Indicates that the statement is not supported

Table 3-8. NonStop ODBC Server Support of SQL Statements (page 2 of 2)

CORE SQL Statement	NonStop ODBC Server Support	Comments
DROP INDEX	x	The following associated objects must be accessible: ● The underlying table ● Catalogs containing the description of the index
DROP TABLE	x	Dependent views are automatically dropped, but you should drop them first because they are not dropped from the mapping tables. Use REFRESH afterward.
DROP VIEW	x	Dependent views are automatically dropped. Use REFRESH afterward.
GRANT	x	Recognized, but not executed.
INSERT	x	Fully supported.
REVOKE	x	Recognized, but not executed.
SELECT	x	Fully supported.
UPDATE	x	Supported with one exception: you cannot set a new value for a column that is part of the primary key for the table.

x Indicates that the statement is supported
– Indicates that the statement is not supported

CORE SQL statements are divided into the following categories:

- Data Definition Language (DDL) statements
- Data Manipulation Language (DML) statements
- Transaction management statements
- Stored procedure execution statement

Table 3-9. Supported CORE SQL Statements by Type (page 1 of 2)

CORE SQL Statement	Description	Corresponding NonStop SQL/MP Statement
DDL Statements		
ALTER TABLE	Adds new columns to an existing table.	ALTER TABLE ADD COLUMN
CREATE INDEX	Creates indexes.	CREATE INDEX
CREATE TABLE	Creates new tables.	CREATE TABLE
CREATE VIEW	Creates views.	CREATE VIEW

Table 3-9. Supported CORE SQL Statements by Type (page 2 of 2)

CORE SQL Statement	Description	Corresponding NonStop SQL/MP Statement
DROP INDEX	Removes an index.	DROP INDEX
DROP TABLE	Removes a table, along with all data, indexes, dependent views, and permission specifications for that table.	DROP TABLE
DROP VIEW	Removes views.	DROP VIEW
DML Statements		
DELETE	Removes rows from a table.	DELETE
INSERT	Adds new rows to a table or view.	INSERT
SELECT	Retrieves rows from tables.	SELECT
UPDATE	Changes data in existing rows, either by adding new data or modifying existing data.	UPDATE
Stored Procedure Execution Statement		
CALL	Invokes a stored procedure (a Pathway server class).	None

ALTER TABLE

Use ALTER TABLE to add new columns to an existing table. The new column appears as the last column of the table.

The ALTER TABLE statement has the following syntax:

```
ALTER TABLE table-name

[ ADD  column-identifier data-type        ]

[ ADD  ( column-identifier data-type      ]
[   [, column-identifier data-type] ... ) ]
```

table-name

is the table to alter. The table name can be qualified with the database name and owner name. The fully expanded table name must be unique in the network.

column-identifier

is the column to add. The column identifier must be unique for columns defined in the table.

If the statement contains errors, the results can be different than if executed using ODBC. See the following discussion of “Adding Multiple Columns.”

data-type

the CORE SQL data type for the column. Data types are converted to NonStop SQL/MP data types. Data types are described under Language Elements on page 3-2.

Example

The following statement adds two columns to the DEPT table. The columns contain BIT type data and are initialized as NULL:

```
alter table dept
   (add deptnew   int,
        deptclose int)
```

Adding Multiple Columns

Because NonStop SQL/MP does not allow you to add multiple columns with one statement, the NonStop ODBC Server translates an ALTER TABLE statement with multiple column specifications to multiple NonStop SQL/MP ALTER TABLE statements. If one of those statements contains an error, depending on where the error occurs, other statements can be executed and the specified columns will be added. However, if a multiple-column ALTER TABLE statement is bound by a user transaction, and an error occurs, no columns are added.

CORE SQL Compared With NonStop SQL/MP

The corresponding NonStop SQL/MP statement is ALTER TABLE ADD COLUMN.

ALTER TABLE in CORE SQL differs from the NonStop SQL/MP implementation in the following ways:

Feature	**In NonStop SQL/MP**	**In CORE SQL**
Number of columns you can add with ALTER TABLE	One	Multiple
Can you specify default values?	Yes	No. You cannot specify a default value unless you use pass-through mode.
Is NOT NULL available?	Yes	No. New columns always allow null values. You cannot specify the NOT NULL attribute unless you use pass-through mode.
Can you alter attributes of a table or column using ALTER TABLE?	Yes	No. You cannot alter the attributes of a column unless you use pass-through mode.

CALL

Use CALL to invoke a previously defined Pathway server class program—a stored procedure—to access the database. Stored procedures are described in detail in Section 5, Stored Procedures.

Stored procedure execution is not included in ODBC CORE SQL but it is one of the elements of ODBC Extended SQL that are supported by the NonStop ODBC Server. Because it is an extension, the CALL statement must be entered as an escape clause.

The CALL statement has the following syntax:

```
--*( VENDOR(Microsoft),PRODUCT(ODBC)

   [ ?= ] CALL procedure-name [ ( parameter [, ... ] ) ] )*--

       or

{ [?=] CALL procedure-name [ ( parameter [, ... ] ) ] }
```

Note. In the escape clause format, the --*(...)*-- or { ... } enclosure is a required element.

?=

represents the return status, to which the procedure returns its completion status code. The codes are determined by the server class programmer.

If you specify ?=, but the data source does not return a status code, the HP NonStop ODBC/MP driver sets the return status to NULL (SQL_NULL_DATA in ODBC).

If you omit ?=, and the data source returns a status code, the driver ignores the return value.

procedure-name

identifies the procedure being called. The form of *procedure-name* is:

```
[ [database.]owner.   ]  procedure-identifier
[ database..          ]
```

The NonStop ODBC Server supports up to 60 characters in the name, but a maximum of 30 is recommended to allow sharing the procedure with TSQL/ODBC or TSQL/DBLIB users.

parameter

is either a literal or a dynamic parameter marker, represented by a question mark (?). In the latter case, you must use the ODBC function SQLSetParam to specify the value.

Considerations

- If the procedure call supplies more values than the number of parameters defined in the procedure declaration, the excess values are ignored. No error or warning is issued.
- If default values are specified for parameters of a procedure (using the NonStop ODBC Server catalog utility ADD PROCEDURE), you can execute a stored procedure without giving a parameter value. The missing values can appear anywhere in the input parameter list. For example, the procedure DEBIT_ACCOUNT has two input parameters, NAME and AMOUNT, and can be called as following:

  ```
  { call debit_account( , 100) }
  ```

  In this case, the account holder's name is whatever is defined for that parameter in the procedure declaration; for example, "SMITH".

  If a default value is not specified for a parameter, an error is issued if that parameter is missing.
- If the data type of a parameter and its corresponding parameter definition in the procedure declaration are not compatible, a data conversion error occurs. If data conversion results in data truncation, a warning is issued. It is the responsibility of the Pathway server class program to take the appropriate action.

Example

The following statement invokes a stored procedure named DEBIT_ACCOUNT:

```
{ call debit_account(?,100) }
```

One parameter is a dynamic parameter marker and the other is a literal. The value for the dynamic parameter must be set by the SQLSetParam function before the stored procedure is executed.

CORE SQL Compared With NonStop SQL/MP

NonStop SQL/MP has no statement corresponding to CALL.

CREATE INDEX

Use CREATE INDEX to create indexes.

The CREATE INDEX statement has the following syntax:

```
CREATE [UNIQUE] INDEX index-name

   ON base-table-name

   ( column-identifier [ ASC | DESC ]
     [, column-identifier [ ASC | DESC ] ... )
```

`UNIQUE`

specifies that two or more rows of the table cannot have the same values for the indexed columns.

NonStop SQL/MP does not allow unique indexes on nullable columns.

`index-name`

specifies the index to create. The named index must not already exist.

`base-table-name`

is the table in which the indexed column or columns are located. It must be an existing table. The *`table-name`* can be qualified with the database and owner.

`column-identifier`

specifies the column or columns to which the index applies.

ODBC does not limit the number of columns you can specify. In NonStop SQL/MP, however, the number of columns that can be indexed depends on the length of the index key. For more information, see the *NonStop SQL/MP Reference Manual*.

`[ ASC | DESC ]`

specifies whether the order of the referenced column is ascending or descending. ASC is the default.

Examples

The following statements create indexes on the EMPLOYEE table:

```
create unique index xempnum on employee (empnum)
create index xempname on employee (last_name, first_name)
```

CORE SQL Compared With NonStop SQL/MP

The corresponding NonStop SQL/MP statement is CREATE INDEX.

CREATE INDEX in CORE SQL differs from the NonStop SQL/MP implementation in the following ways:

Feature	**In NonStop SQL/MP**	**In CORE SQL**
Can you create partitioned indexes?	Yes	No, unless you use pass-through mode.
Can you specify partition attributes?	Yes	No, NonStop SQL/MP defaults are taken unless you use pass-through mode.
Can you specify file attributes?	Yes	No, NonStop SQL/MP defaults are taken unless you use pass-through mode.
Can you specify the physical location of the index?	Yes	No, unless you use pass-through mode. You can create an index only on the subvolume that holds the catalog in which the index will be registered.
Can you create a clustered index?	Yes	No, unless you use pass-through mode.
Can you create a UNIQUE index on a nullable column?	No	Yes

CREATE TABLE

Use CREATE TABLE to create new tables. The file security for the table is obtained from the default permissions associated with the user's logon name.

The CREATE TABLE statement has the following syntax:

```
CREATE TABLE base-table-name

  ( column-element [, column-element ] ... )

column-element is:

  column-definition | table-constraint-definition

column-definition is:

  column-identifier data-type [ DEFAULT default-value ]

  [ column-constraint-definition
    [, column-constraint-definition ] ... ]

default-value is:

  [  literal  ]
  [  NULL     ]
  [  USER     ]

column-constraint-definition is:

  [ NOT NULL                                               ]
  [                                                        ]
  [ UNIQUE                                                 ]
  [ PRIMARY KEY                                            ]

table-constraint-definition is:

  [ PRIMARY KEY                                            ]
  [    ( column-identifier [, column-identifier ] ...)     ]
  [                                                        ]
  [ UNIQUE                                                 ]
  [    ( column-identifier [, column-identifier ] ...)     ]
  [ CHECK ( search-condition )                             ]
```

base-table-name

specifies the table to be created. The *table-name* can be qualified with the database name and owner name. The fully expanded table name must be unique in the network.

If the *owner-name* is omitted, the current user's name is used.

column-definition

defines the characteristics of a single column in the table being created, as follows:

column-identifier

specifies a column in the table. The *column-identifier* must be unique for columns defined in the table.

data-type

is the CORE SQL data type for the column. Data types are converted to NonStop SQL/MP data types. Data types are described under Language Elements on page 3-2.

DEFAULT *default-value*

specifies a default value to be placed in each row in the referenced column when the table is created. The default value can be any of the following:

default-value	**Value Assigned**
literal	The specified value
NULL	A null value
SYSTEM	A timestamp issued at the time the default is used
USER	A character string containing the logical username of the current user

Because of NonStop SQL/MP restrictions, a *literal* used as a *default-value* cannot be longer than 8 characters. If you specify the USER option, the logical username also cannot be longer than 8 characters.

If you omit this entry, NULL is used.

column-constraint-definition

specifies any constraints to be placed on the referenced column.

NOT NULL specifies that the column cannot contain null values. NOT NULL can appear not more than once in a column definition.

No other column constraints are supported.

table-constraint-definition

specifies constraints to be placed on one or more columns.

PRIMARY KEY (*column-identifier* [, *column-identifier*] ...)

states that each row is constrained to contain a different value in the specified column or columns. Each specified column must also be constrained by NOT NULL.

PRIMARY KEY can appear not more than once in a CREATE TABLE statement.

No other table constraints are supported.

Examples

The following statement creates a table called EMPLOYEE:

```
create table employee
   (empnum       smallint,
    first_name    char(15),
    last_name     char(20),
    deptnum       smallint  default null,
    jobcode       smallint  default null,
    salary        float     default null)
```

The following statement creates a table called DEPT:

```
create table dept
   (deptnum       smallint  not null,
    deptname      char(12),
    manager       smallint,
    rptdept       smallint  default null,
    location      char (18) default null,
    primary key (deptnum))
```

The table is created on the PERSNL subvolume and is recorded in the NonStop SQL/MP catalog on that subvolume.

File Attributes and Security Considerations

The following NonStop SQL/MP file attributes and security considerations apply to the use of CREATE TABLE:

- The default physical file attributes for the table are the same as in NonStop SQL/MP.
- The default table organization is the same as in NonStop SQL/MP: KEY SEQUENCED.
- The table definition must comply with NonStop SQL/MP limits. See the *NonStop SQL/MP Reference Manual* for the limits.
- The ownership and security of the table affect dependent indexes and views. These dependencies are discussed in the CREATE VIEW, CREATE INDEX, and ALTER security statements.

For further information about these NonStop SQL/MP features, see the CREATE TABLE statement description in the *NonStop SQL/MP Reference Manual.*

Creation of Partitioned Tables

You can create an SQL/MP partitioned table by using a partition overlay template with the CREATE TABLE statement. For details, see Appendix F, Creating Partitioned Tables.

CORE SQL Compared With NonStop SQL/MP

The corresponding NonStop SQL/MP statement is CREATE TABLE.

CREATE TABLE in CORE SQL differs from the NonStop SQL/MP implementation in the following ways:

Feature	In NonStop SQL/MP	In CORE SQL
Can you specify the physical location of a table?	Yes	No. a table created using the NonStop ODBC Server resides in the same location as the NonStop SQL/MP catalog in which it is registered, unless the profile attribute DEFAULT_ LOCATION names an alternative volume. If it does, the table is created in that alternative location, but registered in the current catalog.
Can you create partitioned tables?	Yes	No, unless you use pass-through mode.
Can you create nonaudited tables?	Yes	No, all tables created through the NonStop ODBC Server are audited. To create a nonaudited table, use pass-through mode.
Data types available	All NonStop SQL/MP data types	Some NonStop SQL/MP data types. For information about the data types, see Language Elements on page 3-2.
Can you specify table attributes?	Yes	No, unless you use pass-through mode. Tables created through the NonStop ODBC Server use default NonStop SQL/MP table attributes.

CREATE VIEW

Use CREATE VIEW to create a view.

The file security for the view is obtained from the default permissions associated with the user's logon name.

The CREATE VIEW statement has the following syntax:

```
CREATE VIEW table-name

  [ ( column-identifier [, column-identifier ] ... ) ]

  AS query-expression

query-expression is:

  { query-expression UNION [ALL] query-expression }
  { ( query-expression )                          }
  { query-specification                            }
```

`table-name`

identifies the view to create.

You can specify the owner of a view, but you cannot specify the database. (You can create a view only in the current database.)

`column-identifier`

specifies the alternate column names for the columns of the view. The *`column-identifier`* must be unique for columns defined in the view.

If you omit the *`column-identifier`* list, the column names of the view take the names from the columns in the select list of the *`query-specification`*.

`query-expression`

is a *`query-specification`* or a union of multiple *`query specification`*s.

`query-specification`

is a SELECT statement that defines the columns for the view and sets the selection criteria. Unless you declare alternate column names in the *`column-identifier`* list, the column names you specify in the *`query-specification`* define the column names of the view.

Tables in the *`query-specification`* must be in the current database or be qualified by their database name.

If there is only one table referenced by the view, NonStop ODBC automatically creates a protection view.

The *query-specification* cannot include the ORDER BY clause, but it can include the OUTER JOIN and UNION clauses.

For more information, see SELECT on page 3-54.

Examples

The following statement creates a view called MGRLIST:

```
create view mgrlist
  (first_name,
   last_name,
   department)
   as select first_name,
             last_name,
             deptname
   from dept,
     employee
     where dept.manager = employee.empnum
```

This view includes columns from the DEPT table and the EMPLOYEE table of the current database.

The following statement creates a single view from all of the columns in the TAB1 and TAB2 tables.

```
create view view1
  as select *
  from tab1
  union select *
    from tab2
```

Shorthand Views and Protection Views

NonStop SQL/MP has two types of views:

Shorthand	Derived from one or more tables or other views and defined without the protection attribute. A shorthand view can be read but not updated.
Protection	Derived from a single table by taking a projection of the columns of the table, or a selection of the rows of the table, or both. For more information about the rules for protection views, see CREATE VIEW in the *NonStop SQL/MP Reference Manual*. A protection view can be secured, updated, and read, with certain restrictions; see the *NonStop SQL/MP Reference Manual*.

When you create a view using the NonStop ODBC Server, the NonStop ODBC Server attempts to create a protection view. If the create fails, however, the NonStop ODBC Server creates a shorthand view. The NonStop ODBC Server returns an informational message in this case.

The view definition text indicates whether a view is a shorthand or a protection view. You can list this text by querying the NonStop SQL/MP VIEWS table, which resides on the subvolume of your NonStop SQL/MP catalog. The following SQLCI statement lists the view definition text for a view:

```
select viewtext from views where viewname =
   "\TEST.$VOL2.PERSNL.EMPLIST";
```

In this example, both the VIEWS table and the EMPLIST view are in the subvolume \TEST.$VOL2.PERSNL.

CORE SQL Compared With NonStop SQL/MP

The corresponding NonStop SQL/MP statement is CREATE VIEW.

CREATE VIEW in CORE SQL differs from the NonStop SQL/MP implementation in the following ways:

Feature	In NonStop SQL/MP	In CORE SQL
Where is the view created?	In the user’s current node, volume, and subvolume	In the same subvolume as the catalog in which the view is registered
Can you specify protection view or shorthand view?	Yes	No, unless you use pass-through mode.
Is WITH CHECK OPTION available?	Yes	No, unless you use pass-through mode.

DELETE

Use DELETE to remove rows from a table. DELETE has two forms: the positioned DELETE and the searched DELETE.

The positioned DELETE deletes the row from which the current row of the active set of a cursor is derived.

The searched DELETE deletes any rows that satisfy a search condition.

Positioned DELETE

The positioned DELETE statement has the following syntax:

```
DELETE FROM table-name

  WHERE CURRENT OF cursor-name
```

table-name

is the table from which rows are to be deleted. The *table-name* can be qualified with the database name and owner name. If *table-name* is a viewed table, DELETE deletes the corresponding row of the base table from which the viewed table is derived.

The *table-name* must be the (single) table referenced by the FROM clause of the *query-specification* that defines the result table of the cursor.

If the specified table is audited, the positioned DELETE statement cannot be used unless a user transaction is in progress (the SQL_AUTOCOMMIT option of the ODBC function SQLSetConnectOption must be off).

cursor-name

specifies the cursor for which the active row is to be deleted. The cursor specification for *cursor-name* must include the FOR UPDATE clause, and the result table of the cursor must be updatable.

The *search-condition* cannot contain subqueries that refer to the table or view from which the rows are being deleted.

If you omit the WHERE clause, all rows are deleted.

Searched DELETE

The searched DELETE statement has the following syntax:

```
DELETE FROM table-name

  [ WHERE search-condition ]
```

table-name

is the table from which rows are to be deleted. The *table-name* can be qualified with the database name and owner name. If *table-name* is a viewed table, DELETE deletes the corresponding row of the base table from which the viewed table is derived.

The *table-name* must be updatable and must not be referenced, directly or indirectly, by a FROM clause of any subquery contained in *search-condition*.

search-condition

specifies the criteria for the rows to delete. DELETE deletes every row that satisfies *search-condition*. If no row satisfies *search-condition*, DELETE does not delete any rows.

If you omit the WHERE clause, DELETE deletes all rows in the table.

Examples

The following statements delete one record from the EMPLOYEE table and one from the DEPARTMENT table:

```
delete employee
   where last_name = "Hall" and first_name = "Dana"

delete department
   where deptnum = 1030
```

CORE SQL Compared With NonStop SQL/MP

The corresponding NonStop SQL/MP statement is DELETE.

DELETE in CORE SQL differs from the NonStop SQL/MP implementation in the following ways:

Feature	In NonStop SQL/MP	In CORE SQL
Can you specify the locking mode?	Yes	No, unless you use pass-through mode.
Default locking mode	STABLE ACCESS	Depends on the setting of the SQL_TXN_ISOLATION item in the user profile table, ZNSPROF. The default is STABLE ACCESS.

DROP INDEX

Use DROP INDEX to remove an index from the database.

The DROP INDEX statement has the following syntax:

```
DROP INDEX index-name
```

index-name

specifies the index to drop.

Example

The following statement drops the index XDEPTNUM from the database:

```
drop index xdeptnum
```

CORE SQL Compared With NonStop SQL/MP

The corresponding NonStop SQL/MP statement is DROP INDEX, which is the same as the CORE SQL statement.

DROP TABLE

Use DROP TABLE to remove a table from a database, along with all data, indexes, and dependent views for that table.

The DROP TABLE statement has the following syntax:

```
DROP TABLE base-table-name

  [ CASCADE | RESTRICT ]
```

base-table-name

specifies the table to be dropped. The table name can be qualified with the database name and owner name.

DROP TABLE also drops any indexes based on the base table.

`CASCADE`

specifies that constraints and viewed tables that reference the base table are also dropped. CASCADE is the default specification.

`RESTRICT`

is not supported. The setting of the SQL_UNSUPPORTED flag in the ZNSPROF mapping table determines whether the NonStop ODBC Server reports an error, issues a warning message, or ignores the RESTRICT parameter.

Example

The following statement drops the EMPLOYEE table from the database:

```
drop table employee
```

The DROP TABLE statement also drops any constraints, indexes, and views that reference the EMPLOYEE table.

Dropping Dependent Views and Indexes

You should drop dependent views and indexes before dropping a table using the NonStop ODBC Server.

When you drop a table using the NonStop ODBC Server, NonStop SQL/MP drops the table and all dependent views and indexes. The NonStop ODBC Server, however, does not clear the mapping table entries for the views or indexes.

Note. When the mapping table entries are not cleared and you later attempt to create a view or index by the same name, you will receive error messages saying that the object already exists when in fact it does not.

The only way to clear the mapping table entries in this situation is to run the catalog utility statement REFRESH.

For information about REFRESH, see Section 7, Managing Customized Catalogs.

Dropping Partitioned Tables

NonStop SQL/MP allows creation of partitioned tables. Each partition has an entry in the mapping table. Dropping a table that is partitioned causes all the partitions to be dropped, but the mapping entries will remain. To clear the entries, run the catalog utility statement REFRESH.

CORE SQL Compared With NonStop SQL/MP

The corresponding NonStop SQL/MP statement is DROP TABLE, which, although it does not include the CASCADE clause, functions the same as the CORE SQL statement.

DROP VIEW

Use DROP VIEW to remove views from the database.

The DROP VIEW statement has the following syntax:

```
DROP VIEW viewed-table-name

  [ CASCADE | RESTRICT ]
```

viewed-table-name

the view to drop. The *viewed-table-name* can be qualified with the owner name.

CASCADE

specifies that all viewed tables based on *viewed-table-name* are also dropped. CASCADE is the default specification.

RESTRICT

is not supported. The setting of the SQL_UNSUPPORTED flag in the ZNSPROF mapping table determines whether the NonStop ODBC Server reports an error, issues a warning message, or ignores the RESTRICT parameter.

Example

The following statement drops the MGRLIST view owned by PAYROLL_ADMIN:

```
drop view emplist payroll_admin.mgrlist cascade
```

The DROP VIEW statement also drops any views based on that view.

Dropping Dependent Views and Indexes

You should drop dependent views and indexes before dropping a view using the NonStop ODBC Server.

When you drop a view using the NonStop ODBC Server, NonStop SQL/MP drops the view and all dependent views and indexes. The NonStop ODBC Server, however, does not clear the mapping table entries for the views or indexes.

Note. When the mapping table entries are not cleared and you later attempt to create a view or index by the same name, you will receive error messages saying that the object already exists when in fact it does not.

The only way to clear the mapping table entries in this situation is to run the catalog utility statement REFRESH.

For information about REFRESH, see Section 7, Managing Customized Catalogs.

CORE SQL Compared With NonStop SQL/MP

The corresponding NonStop SQL/MP statement is DROP VIEW, which functions the same as the CORE SQL statement, although it does not contain the CASCADE clause.

GRANT

The NonStop ODBC Server recognizes the syntax of the GRANT statement, but does not perform its activities.

The GRANT statement has the following syntax:

```
GRANT  { ALL                                      }
       { grant-privilege [, grant-privilege ] ... }

  ON table-name

  TO  { PUBLIC                       }
      { user-name [, user-name ] ... }
```

The setting of the SQL_UNSUPPORTED flag in the ZNSPROF mapping table determines whether the NonStop ODBC Server reports an error, issues a warning message, or ignores the GRANT statement.

INSERT

Use INSERT to add new rows to a table or a view.

The INSERT statement has the following syntax:

```
INSERT [ INTO ] table-name

  [ ( column-identifier [, column-identifier ] ... ) ]

  { VALUES ( insert-value [, insert-value ] ... ) }
  { query-specification                            }

insert value is

  [ dynamic-parameter ]
  [ literal           ]
  [ NULL              ]
  [ USER              ]
```

table-name

is the table into which the rows are inserted. The *table-name* can be qualified with the database name and owner name. If *table-name* is a viewed table, the INSERT statement inserts rows into the base table from which *table-name* is derived; the view must be a NonStop SQL/MP protection view. For information about protection views, see Shorthand Views and Protection Views on page 3-44.

The table *table-name* must be updatable.

column-identifier

specifies a column for which a value will be supplied. Omitting all *column-identifiers* is equivalent to specifying all the columns of *table-name*, in ascending order of their position in the table.

query-specification

is a SELECT statement that selects values from other tables and views to be inserted in the table or view specified in the INTO clause.

If the INSERT statement contains a *query-specification*, it inserts one row of values into *table-name* for each row of the derived table. The value of the first column of a row in the derived table is inserted into the first specified column, the second row value into the second specified column, and so on. The row must contain an element for each column that you specify in the *column-identifier* list.

The *select-statement* cannot refer to the table, view, or underlying table of the view into which the rows are being inserted.

`insert-value`

represents a value to be inserted into the corresponding column. It can be any of the following:

`insert-value`	**Value Assigned**
`dynamic-parameter`	Represented by a question mark (?), it identifies a parameter in a dynamically prepared SQL statement
`literal`	The exact value specified
NULL	A null value
USER	A character string containing the logical username of the current user

If the VALUES clause is present, INSERT inserts a single row with the specified *`insert-values`* as column values. It assigns the first *`insert-value`* to the first specified column, the second value to the second column, and so on. There must be one *`insert-value`* for each *`column-identifier`*.

Example

The following example adds one row to the EMPLOYEE table:

```
insert into test_disk01_persnl..employee
   (empnum, first_name, last_name, deptnum, jobcode, salary)
   values (1234, "Georgia", "Brown", 1001, 3004, 52300)
```

CORE SQL Compared With NonStop SQL/MP

The corresponding NonStop SQL/MP statement is INSERT.

INSERT in CORE SQL differs from the NonStop SQL/MP implementation in the following ways:

Feature	**In NonStop SQL/MP**	**In CORE SQL**
Can you specify the locking mode?	Yes	No, unless you use pass-through mode.
Default locking mode	STABLE ACCESS	Depends on the setting of the SQL_TXN_ISOLATION item in the user profile table, ZNSPROF. The default is STABLE ACCESS.

POWER

Use POWER to retrieve the value of a numeric expression to a specified power.

The POWER scalar function has the following syntax:

```
POWER (numeric-exp, integer-exp)
```

numeric-exp

is a number. It can be the name of a column, the result of another scalar function, or a numeric literal.

integer-exp

is the power to which the number is raised. It can be the name of a column, the result of another scalar function, or a numeric literal.

Example

The following SQL statement returns the value 5.2 squared:

```
select {fn power (5.2,2) }
   from tab1
```

REVOKE

The NonStop ODBC Server recognizes the syntax of the REVOKE statement, but does not perform its activities.

The REVOKE statement has the following syntax:

```
REVOKE  { ALL                                       }
        { grant-privilege [, grant-privilege ] ...) }

  ON table-name

  FROM  { PUBLIC                        }
        { user-name [, user-name ] ...  }

  [ CASCADE | RESTRICT ]
```

The setting of the SQL_UNSUPPORTED flag in the ZNSPROF mapping table determines whether the NonStop ODBC Server reports an error, issues a warning message, or ignores the REVOKE statement.

SELECT

Use SELECT to retrieve rows from database tables or to access rows for use in updating columns in the same or another table. The SELECT statement has two forms: the standard SELECT and SELECT for update.

Standard SELECT

The SELECT statement has the following syntax:

```
SELECT  [ ALL      ]  select-item [, select-item ] ...
        [ DISTINCT ]

  FROM table-reference-list

  [ WHERE search-condition ]

  [ GROUP BY column-identifier | column-alias
    [,column-identifier | column-alias ] ... ]

  [ HAVING search-condition ]

  [ UNION query-specification ]

  [ ORDER BY sort-specification [, sort-specification ] ... ]

select-item is:

  {  [ table-name        ]  *                  }
  {  [ correlation-name  ]                     }
  {                                            }
  {  [ table-name       ]  column-identifier  }
  {  [correlation-name  ]                      }
  {                                            }
  {  expression                                }
  {                                            }

table-reference-list is:

  table-reference [, table-reference ] ...
```

```
table-reference is:

  table-name [ correlation-name ] | [outer-join-extension ]

outer-join-extension is

  { OJ outer-join } | outer-join

outer-join is:

  table-name [ correlation-name ] LEFT [OUTER] JOIN
    oj-table-reference
    ON search-condition

oj-table-reference is:
  table-name [ correlation-name ] | outer-join

sort-specification is:

  [ unsigned-integer                  ]
  [ column-identifier  [ ASC | DESC ] ]
```

Note. The braces ({ and }) in the `outer-join-extension` in the preceding syntax are not the usual syntax convention indicating choice, but instead are actual characters that you must enter.

Examples

The following statement retrieves all rows and all columns from the EMPLOYEE table:

```
select * from employee
```

The following statement retrieves all employees in department 1030:

```
select * from employee
   where deptnum = 1030
```

This statement finds the average salary for each department in the EMPLOYEE table:

```
select AVG ( salary ) from employee
   GROUP BY deptnum ORDER BY deptnum
```

The preceding syntax diagram shows the entire syntax of the SELECT statement. The following diagrams show the syntax of each primary clause.

ALL/DISTINCT Clause

The ALL/DISTINCT clause of the SELECT statement specifies the columns to be included in the result of the query. The clause has the following syntax:

```
{ ALL      }  select-item [, select-item ] ...
{ DISTINCT }

select-item is:

  { [ table-name      }  *
}
  { [ correlation-name}
}
  {
}
  { [ table-name      ]column-identifier [as[column-alias ]]
}
  { [ correlation-name]
}
  {
}
  {  expression [as column-alias ]]
}
  {
}
```

ALL or DISTINCT

specifies whether all rows or only distinct rows are to be included in the result of the query:

ALL	Include all rows of the result table
DISTINCT	Include only unique rows of the result table. Duplicate rows are removed from the result table.

ALL is the default.

[*table-name*.]*

retrieves all columns of a table or view. An asterisk (*) specifies all columns, in CREATE TABLE order.

[*table-name*.]*column-identifier*

retrieves the specified columns in the order specified.

expression

is an expression. Expressions are described under Language Elements on page 3-2.

FROM Clause

The FROM clause of the SELECT statement specifies the tables and views used in the SELECT statement; it has the following syntax:

```
table-reference is

  table-name [ correlation-name ] | outer-join-extension

outer-join-extension is

  { OJ outer-join } | outer-join

outer-join is

  table-name [ correlation-name ] LEFT [OUTER] JOIN
    oj-table-reference
    ON search-condition

oj-table-reference is table-name [correlation-name ] | outer-
join
```

table-name

identifies a table or view used in the SELECT statement. The *table-name* or *view-name* can be qualified with the database name and owner name.

correlation-name

is a name that identifies the table or view for use within the SELECT statement. Each *correlation-name* in the FROM clause must be unique.

outer-join

is an ODBC extension that permits return of rows from a table even if no matching rows exist in the other table. See Escape Clauses on page 3-11.

Note that, unlike NonStop SQL, outer joins are nested. That is, the first ON clause is matched with the second outer join and the second ON clause is matched with the first outer join.

WHERE Clause

The WHERE clause of the SELECT statement specifies a search condition to be applied to each row of the FROM clause result table; it has the following syntax:

```
WHERE search-condition
```

Search conditions are described under Language Elements on page 3-2.

GROUP BY Clause

The GROUP BY clause of the SELECT statement specifies the groups into which the table will be divided; it has the following syntax:

```
GROUP BY column-identifier [,column-identifier ] ...
```

column-identifier

is the name of a column from one of the tables in the FROM clause.

HAVING Clause

The HAVING clause of the SELECT statement sets conditions for the GROUP BY clause and has the following syntax:

```
HAVING search-condition
```

Search conditions are described under Language Elements on page 3-2.

UNION Clause

The UNION clause of the SELECT statement specifies that the result table is to contain all the rows of the tables defined by both the *query-specification* in the WHERE clause and the *query-specification* in the UNION clause; it has the following syntax:

```
UNION query-specification
```

ORDER BY Clause

The ORDER BY clause of the SELECT statement specifies the order in which to sort the rows of the result table; it has the following syntax:

```
ORDER BY sort-specification [, sort-specification ] ...

sort-specification is:

  { unsigned-integer }  [ ASC | DESC ]
  { column-name      }
```

unsigned-integer

is a number from 1 to *n* that specifies the position of an item in the *select-item* list; the sorting order of that item is used as the sorting order of the result table rows. For example:

```
select col_1,col_2 from table_1 order by 1
```

specifies that the result table is to be sorted in the same sequence as COL_1.

column-name

specifies a column in a table in the FROM clause; the sorting order of that column is used as the sorting order of the result table rows. For example:

```
select col_1,col_2 from table_1 order by col_1
```

specifies that the result table is to be sorted in the same sequence as COL_1.

`ASC or DESC`

explicitly specifies whether to sort in ascending or descending order; the default is ascending.

The actual sorting sequence used, numeric or ASCII, depends on the data type of the column chosen by *unsigned-integer* or *column-name*.

If the SELECT statement contains a UNION clause, the ORDER BY clause must be the last clause in the statement, as it defines the ordering of the result of that union.

SELECT for Update

The SELECT for update statement is used to associate a list of updatable columns with a dynamic SQL statement; a cursor can then be declared for the statement. The SELECT for update statement has the following syntax:

```
SELECT  [ ALL      ]  select-item [, select-item ] ...
        [ DISTINCT ]

  FROM table-reference [, table-reference ]...

  [ WHERE search-condition ]

  FOR UPDATE OF column-name [, column-name ] ...
```

column-name

is the name of a column in a table specified in the FROM clause. It must be updatable.

CORE SQL Compared With NonStop SQL/MP

The corresponding NonStop SQL/MP statement is SELECT.

SELECT in CORE SQL differs from the NonStop SQL/MP implementation in the following ways:

Feature	In NonStop SQL/MP	In CORE SQL
Is BROWSE ACCESS locking mode available?	Yes	No, unless you use pass-through mode.
Is the INTO clause available?	Yes	No, unless you use pass-through mode.
Is the IN EXCLUSIVE MODE clause available?	Yes	No, unless you use pass-through mode. See the following discussion of “IN EXCLUSIVE MODE Clause.”

IN EXCLUSIVE MODE Clause

In programs used with the NonStop ODBC Server, you cannot specify the NonStop SQL/MP IN EXCLUSIVE MODE clause; you can, however, obtain exclusive mode by locking the table using pass-through mode as shown in the following example:

```
begin work

select "tdm: sql lock table \test.$disk01.persnl.employee
   in exclusive mode"

select jobcode, deptnum, first_name, last_name
   from employee
   where jobcode > 500 and deptnum <= 3000"

select "tdm: sql unlock table \test.$disk01.persnl.employee"

commit transaction
```

Remember to unlock the table when you no longer need the lock on it, as illustrated in the preceding example.

UPDATE

Use UPDATE to change data in existing rows, either by adding new data or by modifying existing data. UPDATE has two forms: the positioned UPDATE and the searched UPDATE.

The positioned UPDATE changes data in the row from which the current row of the active set of a cursor is derived.

The searched UPDATE changes data in any rows that satisfy a search condition.

Positioned UPDATE

The positioned UPDATE statement has the following syntax:

```
UPDATE table-name

  SET column-identifier =  { expression }
                           { NULL       }

     [, column-identifier =  { expression }  ] ...
                             { NULL       }

  WHERE CURRENT OF cursor-name
```

table-name

is the table in which rows are to be changed. The *table-name* can be qualified with the database name and owner name. If *table-name* is a viewed table, UPDATE changes the corresponding row of the base table from which the viewed table is derived.

The *table-name* must be the (single) table referenced by the FROM clause of the *query-specification* that defines the result table of the cursor.

column-identifier

specifies a column in which the row is to be changed.

expression

is a valid expression to be assigned. Expressions are described under Language Elements on page 3-2.

NULL

sets the value to NULL.

cursor-name

specifies the cursor for which the active row is to be changed. The cursor specification for *cursor-name* must include the FOR UPDATE clause and the result table of the cursor must be updatable.

Searched UPDATE

The searched UPDATE statement has the following syntax:

```
UPDATE table-name

  SET column-identifier =  { expression }
                           { NULL       }

     [, column-identifier =  { expression }  ] ...
                             { NULL       }

  WHERE CURRENT OF search-condition
```

table-name

is the table in which rows are to be changed. The *table-name* can be qualified with the database name and owner name. If *table-name* is a viewed table, UPDATE changes the corresponding row of the base table from which the viewed table is derived.

The *table-name* must be the (single) table referenced by the FROM clause of the *query-specification* that defines the result table of the cursor.

column-identifier

specifies a column in which a row is to be changed.

expression

is a valid expression to be assigned. Expressions are described under Language Elements on page 3-2.

NULL

sets the value to NULL.

search-condition

specifies criteria for selecting the rows to be updated.

The *search-condition* cannot have a subquery that references the table or view in which the rows are being updated.

Example

The following example updates the salary of employee number 1002:

```
update test_disk01_persnl..employee
   set salary = salary * 1.1
   where empnum = 1002
```

CORE SQL Compared With NonStop SQL/MP

The corresponding NonStop SQL/MP statement is UPDATE.

UPDATE in CORE SQL differs from the NonStop SQL/MP implementation in the following ways:

Feature	In NonStop SQL/MP	In CORE SQL
Can you specify the locking mode?	Yes	No, unless you use pass-through mode.
Default locking mode	STABLE ACCESS	REPEATABLE ACCESS

4 Transact-SQL Language

Applications used with the HP NonStop ODBC Server can contain Transact-SQL statements. The NonStop ODBC Server accepts these Transact-SQL statements, translates them to HP NonStop SQL/MP statements, and sends the statements to NonStop SQL/MP. The NonStop ODBC Server can also execute some NonStop SQL/MP statements directly.

This section provides the following information:

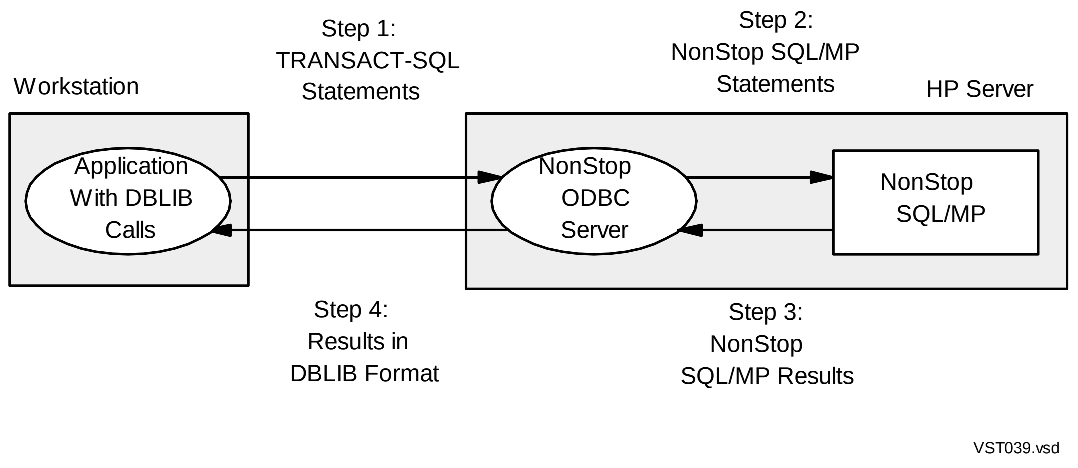
- Summarizes the general differences between executing Transact-SQL statements in SQL Server and executing them using the NonStop ODBC Server.
- Describes the language elements, such as names, data types, functions, and variables, used to construct Transact-SQL statements.
- Summarizes the supported Transact-SQL statements.
- Alphabetically lists and describes the Transact-SQL statements that you can use in programs that interact with the NonStop ODBC Server.
- Gives examples of each supported statement.
- Explains the differences for each supported statement between:
    - The Transact-SQL statement and the NonStop ODBC Server implementation of the statement.
    - The NonStop SQL/MP statement and the NonStop ODBC Server implementation of the statement.

NonStop ODBC Server Translation

Application programs used with the NonStop ODBC Server contain DB-LIBRARY calls and Transact-SQL statements.

Figure 4-1 illustrates the relationships among Transact-SQL, the NonStop ODBC Server, and NonStop SQL/MP.

Figure 4-1. Relationships Among Transact-SQL, the NonStop ODBC Server, and NonStop SQL/MP



A Transact-SQL statement is submitted to NonStop SQL/MP as follows:

1. An application program issues a DB-LIBRARY call that sends a Transact-SQL statement to the NonStop ODBC Server.
2. The NonStop ODBC Server translates the Transact-SQL statement to a NonStop SQL/MP statement and sends the statement on to NonStop SQL/MP.
3. NonStop SQL/MP processes the statement and returns NonStop SQL/MP data (or a diagnostic message) to the NonStop ODBC Server.
4. The NonStop ODBC Server translates the NonStop SQL/MP output to Transact-SQL format and sends the result to the application program.
5. The application retrieves the data using DB-LIBRARY calls.

Using Transact-SQL Through the NonStop ODBC Server

This subsection summarizes some differences between using Transact-SQL in SQL Server and using it in the NonStop ODBC Server. It covers the following topics:

- Unsupported features
- Mapping of table, view, and index names
- DDL statements allowed in user-defined transactions
- Specifying the database location
- Batch queries

Unsupported Features

As stated in “Notation Conventions,” unsupported SQL Server features are underlined in this manual. For example, in the following syntax, the ON DEFAULT clause is unsupported.

```
CREATE DATABASE database-name

  ON  { DEFAULT)        }  [ = size ]
      { database-device }
```

Mapping of Table, View, and Index Names

When you create tables, views, and indexes, you specify Transact-SQL identifiers for the table, view, or index names. The NonStop ODBC Server maps these identifiers to eight-character Guardian file names. Even though the names are mapped, you refer to the object by its Transact-SQL name when accessing it using the NonStop ODBC Server.

You need to be concerned with the name mapping only if you are locating the object in the NonStop ODBC Server mapping tables. The NonStop ODBC Server maps table, view, and index names to NonStop SQL/MP simple names as follows:

Character	**Replaced With**
Number sign (#)	Zero (0)
Dollar sign ($)	Zero (0)
Underscore (_)	Zero (0)
Leading underscore (_)	The letter U (uppercase)

The `table-name` for a temporary table begins with a number sign (#). When temporary tables are mapped, the resulting name always begins with an uppercase U.

DDL Statements Allowed in User-Defined Transactions

SQL Server and NonStop SQL/MP have different rules about which DDL statements are allowed in user-defined transactions.

Table 4-1 lists the supported DDL statements and indicates whether they are allowed in user-defined transactions in programs executed using SQL Server and programs executed using the NonStop ODBC Server.

Table 4-1. Statements Allowed in User-Defined Transactions (page 1 of 2)

Statement	**Allowed in SQL Server**	**Allowed in NonStop ODBC Server**
ALTER TABLE	–	x (if the table is audited)
CREATE DATABASE	–	x
CREATE INDEX	–	x

Table 4-1. Statements Allowed in User-Defined Transactions (page 2 of 2)

Statement	Allowed in SQL Server	Allowed in NonStop ODBC Server
CREATE TABLE	–	x
CREATE VIEW	–	x
DROP DATABASE	–	x
DROP INDEX	–	x
DROP TABLE	–	x (if the table is audited)
DROP VIEW	–	x
TRUNCATE TABLE	–	x

x Indicates the statement can be used in a user-defined transaction
– Indicates the statement cannot be used in a user-defined transaction

Specifying the Database Location

With SQL Server, you specify the database location as a parameter of the CREATE DATABASE statement. In programs written for the NonStop ODBC Server, however, the location is implied in the structure of the database name you specify.

The following statement, written for the NonStop ODBC Server, creates a NonStop SQL/MP catalog called PERSNL on the disk volume DISK01 on the node called \TEST:

```
create database test_disk01_persnl
```

In a program written for SQL Server, you would specify the database location using the ON parameter. The following statement, written for SQL Server, creates a database, PERSONNEL, and allocates three megabytes to it on FILE1 and two megabytes to it on FILE2:

```
create database personnel
on file1 = 3, file2 = 2
```

Batch Queries

The NonStop ODBC Server supports batch queries.

The following table summarizes how batches differ in SQL Server and the NonStop ODBC Server.

Feature	In SQL Server	In the NonStop ODBC Server
Can you execute USE DATABASE inside a batch?	Does not take effect until the batch is executed	Takes effect immediately
Are Control-of-Flow statements supported?	Yes	No

If any statement in a batch contains a syntax error, none of the statements in the batch is executed.

Language Elements

Language elements differ for Transact-SQL and NonStop SQL/MP. This subsection describes how to use the following Transact-SQL language elements when using the NonStop ODBC Server:

- Names
- Data types
- Functions
- Variables
- Search conditions
- Expressions and operators
- Aggregates
- Wild-card characters
- NULL values
- Comments

Names

When using the NonStop ODBC Server, the names you specify for objects such as databases, tables, indexes, and columns must be in Transact-SQL syntax; however, there are restrictions on some names, because the NonStop ODBC Server maps some names to Guardian names.

This subsection summarizes name usage, then describes the following types of names:

- Database names
- Owner names
- Table, view, and index names
- Column and correlation names
- Procedure names
- Transaction names
- Variable names

Summary of Name Usage

In Transact-SQL, most names are Transact-SQL identifiers. NonStop SQL/MP does not have corresponding identifiers. Therefore, the NonStop ODBC Server maps these identifiers to NonStop SQL/MP simple names or Guardian names.

Table 4-2 summarizes name format and how the NonStop ODBC Server handles names.

Table 4-2. How the NonStop ODBC Server Handles Transact-SQL Object Names (page 1 of 2)

Name Type	Name Format	How NonStop ODBC Server Handles Name
Database name	NonStop ODBC-database-name	Maps it to \node.$*volume*.*subvolume*, which corresponds to a NonStop SQL/MP catalog.
Owner name	`logical-username`	Maps it to *group-name*.*user-name*, which corresponds to a Guardian logon name.
Table name	Transact-SQL identifier	Maps it to an 8-character Guardian file name. Replaces $ and # and underscore (_) with zero. Replaces a leading underscore with the letter u. Might resort to generating a random name.
View name	Transact-SQL identifier	Maps it to an 8-character Guardian file name. Replaces $ and # and underscore with zero. Replaces a leading underscore with the letter u. Might resort to generating a random name.
Index name	Transact-SQL identifier	Maps it to an 8-character Guardian file name. Replaces $ and # and underscore with zero. Replaces a leading underscore with the letter u. Might resort to generating a random name.
Column heading	Transact-SQL identifier	Retains it, but does not map it. The client can specify the column heading only in the select list of a SELECT command. The value is returned as the column heading in the descriptor that precedes the result data.
Column name	Transact-SQL identifier	Maps it to a NonStop SQL/MP simple name. A column name cannot contain a dollar sign ($), number sign (#), or leading underscore character (_). The server replaces $ and # with an underscore and replaces a leading underscore with the letter u.

Table 4-2. How the NonStop ODBC Server Handles Transact-SQL Object Names (page 2 of 2)

Name Type	Name Format	How NonStop ODBC Server Handles Name
Correlation name	Transact-SQL identifier	Maps it to a NonStop SQL/MP simple name. Replaces $ and # with an underscore. Replaces a leading underscore with the letter u.
Procedure name	Transact-SQL identifier	Maps it to a Pathway server class, but the user sees the name as identified to Transact-SQL.
Transaction name	Transact-SQL identifier	Retains it, but does not map it. The server ignores transaction names and does not support Transact-SQL functions operating on transaction-name.
Variable name	Transact-SQL identifier (no more than 29 characters) preceded by an @ symbol.	Supported directly; not handled by the ODBC server.

All names created using the NonStop ODBC Server are converted to uppercase letters before any SQL statement processing occurs.

Table 4-3 summarizes the rules for creating Transact-SQL identifiers, NonStop SQL/MP simple names, and Guardian names.

Table 4-3. Rules for Object Names

			Guardian Name			
	Transact-SQL Identifier	**NonStop SQL/MP Simple Name**	**Node Name**	**Volume Name**	**Subvolume Name**	**ObjectName**
Length (in characters)	1–30	1–30	1–8	1–7	1–8	1–8
First character	Letter Number sign (#) Underscore (_)	Letter	Backslash (\)	$	Letter	Letter
Remaining characters	Letters Digits Number sign Dollar sign ($) Underscore	Letters Digits Underscore	Letters Digits	Letters Digits	Letters Digits	Letters Digits
Example	table_a	second_val	\finance \system	$persnl $disk01	pay1994	table1

Identifiers used with the NonStop ODBC Server are not case sensitive (My_Table is the same as my_table). Identifiers used with Transact-SQL are case sensitive only if SQL Server is configured to have case-sensitive identifiers.

For more information on identifiers, see the following documents:

For Information On	**See**
Transact-SQL identifiers	*Microsoft SQL Server Transact-SQL Reference*
NonStop SQL/MP identifiers	*NonStop SQL/MP Reference Manual*

Database Names

A logical database name may be any valid NonStop SQL/MP identifier.

You must separate the parts of the name with underscores (_).

Considerations for database names are as follows:

- Each portion of the name begins with a letter and consists of letters and digits. The maximum number of characters for each portion is as follows:

Portion	Characters
Node	7
Volume	6
Subvolume	8

- The NonStop ODBC Server maps the database name to a Guardian identifier as follows:

  ```
  \node.$volume.subvolume
  ```

  For example:

  `corp_vol2_sales` maps to `\corp1.$vol2.sales`

For more information about Guardian names, see the *Guardian User's Guide*.

Owner Names

Each SQL object belongs to an owner, identified by the logical username associated with a Guardian logon name by an ADD USER statement or the logical username associated with an alias username (and with a Guardian logon name) by an ADD ALIAS statement. Owner name is not specifically limited to group_user; it is stored as an identifier in column NOS_USERNAME in the ZNSUS table.

An owner name may be any valid NonStop SQL/MP identifier.

Considerations for owner names are as follows:

- An owner name is a Transact-SQL identifier.
- An owner name is stored in the mapping tables with the associated object name. When you create an object under a specific owner name, you must refer to that object using the owner name (or the associated logical username for an alias username) as a qualifier.

  Note. Do not specify another owner when you create objects. If you do so, the other owner is listed in the NonStop ODBC Server mapping tables, while you are the owner of the object.

- You specify an owner name as part of an index, table, view, or stored procedure name as follows:

  ```
  [ [database.]owner. ]  object
  [ database..        ]
  ```

  If you include the database name but omit the owner name, you must include an extra period to show the omission.

- All object-name references are fully qualified before the SQL statement is executed. If you do not specify an owner name, the logical username established at connection time is used to qualify the object. If the database name is missing, the default database indicated in the ZNSPROF profile record for the current logical username is used.

Some additional considerations for alias usernames are as follows:

- They can be used in place of their associated logical usernames for logging on.
- They cannot be used for qualifying object names.
- They are system-wide, not restricted to a given database.
- If the owner name is missing from an object reference, the logical username associated with the alias username is used.

The NonStop ODBC Server authenticates a logon username as follows:

1. NOSUTIL first checks the length of the logon username. If the name is longer than 17 characters, it cannot be a Guardian name, so NOSUTIL sets the corresponding Guardian username (G_USERNAME) to NULL. If the name is 17 or fewer characters, NOSUTIL does not set G_USERNAME.
2. The NonStop ODBC server then searches the username alias table, ZNSALT, for a mapping entry for the logon username:
    - If a mapping entry is found, the NonStop ODBC server determines the PROFILE_NAME and NOS_USERNAME from the ZNSALT table and the corresponding Guardian username (G_USERNAME) from the ZNSUS table. The NonStop ODBC server uses this information and the corresponding password to authenticate the user.
    - If a mapping entry is not found, the NonStop ODBC server does not consider the user to be an ODBC user. The NonStop ODBC server tries to authenticate the user; however, even if the authentication is successful, the NonStop ODBC server does not allow the user to perform DDL operations.
3. If the authentication fails and the logon username is 17 or fewer characters, the NonStop ODBC server converts the name to a Guardian format name (`group.user`) by substituting the first underscore with a period. The NonStop ODBC server then tries to authenticate the user using the converted Guardian name.

Note. If Safeguard is in effect for the system and the username is an alias, the NonStop ODBC server always uses the alias name for the logon. If the username is not an alias, the NonStop ODBC server uses the Guardian name for the logon. If Safeguard is not in effect, the NonStop ODBC server always uses the Guardian name for the logon.

For more information, see the following documents:

For Information On	See
Transact-SQL usernames	*Microsoft SQL Server System Administrator's Guide*
Name mapping	Section 7, Managing Customized Catalogs
Guardian logon name	*Guardian User's Guide*

Table, View, and Index Names

For table, view, and index names, you specify Transact-SQL identifiers. The NonStop ODBC Server maps these identifiers to eight-character Guardian file names. Even though the names are mapped, you refer to the object by its Transact-SQL name when accessing it using the NonStop ODBC Server.

For more information, see the following documents:

For Information On	See
Transact-SQL object names	*Microsoft SQL Server Transact-SQL Reference*
Mapping of table names	Section 7, Managing Customized Catalogs
Mapping tables	Section 8, HP NonStop ODBC Server Mapping Tables

Column and Correlation Names

For column names and correlation names, you specify Transact-SQL identifiers. The NonStop ODBC Server maps the identifiers to NonStop SQL/MP simple names as follows:

Character	Replaced With
Number sign (#)	Underscore (_)
Dollar sign ($)	Underscore (_)
Leading underscore (_)	The letter U (uppercase)

Even though the names are mapped, you refer to the column or correlation name by its Transact-SQL name when accessing it using the NonStop ODBC Server.

Procedure Names

For procedure names, you specify Transact-SQL identifiers. The NonStop ODBC Server maps these identifiers to the names of Pathway server class programs. Even though the names are mapped, you refer to the object by its Transact-SQL name when accessing it using the NonStop ODBC Server.

Transaction Names

For transaction names, you specify Transact-SQL identifiers. The NonStop ODBC Server retains the transaction name, but does not map it because transaction names are not used in NonStop SQL/MP.

Variable Names

For variable names, you specify Transact-SQL variable names, which are identifiers of no more than 29 characters preceded by an “at” symbol (@). The NonStop ODBC Server retains the variable name but does not map it because variable names are not used in NonStop SQL/MP.

Data Types

When creating and manipulating objects and data items, you must specify Transact-SQL data types. Because Transact-SQL and NonStop SQL/MP data types differ, the data types or values you specify are converted to corresponding NonStop SQL/MP data types.

Even if you are retrieving or manipulating data created with NonStop SQL/MP, you must specify Transact-SQL, not NonStop SQL/MP, data types and values.

This subsection summarizes NonStop ODBC Server support of Transact-SQL data types. Data type support falls in two categories:

- Creating objects and data
- Retrieving and manipulating NonStop SQL/MP objects and data

Creating Objects and Data

When you create objects and data, you specify Transact-SQL data types and values. The NonStop ODBC Server handles the type or value in one of three ways:

- Fully supports it—the data type or value is processed as you input it.
- Converts it—the data type or value is converted to a corresponding NonStop SQL/MP data type or value.
- Flags it as an error—the data type is not supported.

Table 4-4 summarizes the Transact-SQL data types, the corresponding NonStop SQL/MP data types, and the conversion information.

Table 4-4. Conversion of Transact-SQL Data Types to NonStop SQL/MP Data Types

Transact-SQL Data Type	NonStop ODBC Server Support	Corresponding NonStop SQL/MP Data Type
BINARY	–	–
BIT	x	SMALLINT
CHAR	x	CHAR
DATETIME	x	DATETIME year to fraction (3)
FLOAT	x	DOUBLE PRECISION

Table 4-4. Conversion of Transact-SQL Data Types to NonStop SQL/MP Data Types

Transact-SQL Data Type	NonStop ODBC Server Support	Corresponding NonStop SQL/MP Data Type
IMAGE	–	–
INT	x	INTEGER
MONEY	x	DECIMAL
SMALLINT	x	SMALLINT
SYSNAME	x	VARCHAR (30)
TEXT	x	VARCHAR (The default is 512)
TINYINT	x	SMALLINT
USER_TYPE	–	–
VARBINARY	–	–
VARCHAR	x	VARCHAR

x Indicates a supported data type
– Indicates an unsupported data type

Supported DATETIME Formats

The NonStop ODBC Server supports the following DATETIME formats, which are a subset of the Transact-SQL DATETIME formats:

```
datetime

date

time

time date

mixed datetime
```

The *date* can be any of the following:

```
monthn  { . }  day  { . }  year
        { - }       { - }
        { / }       { / }

year4

YYYYMMDD

YYMMDD

day  { . }  day  { . }  year
     { - }       { - }
     { / }       { / }

month [ , ] day [ , ] year

day [ , ] month [ , ] year

day [ , ] year month

year month

year4 month [ , ] day

month [ , ] year4
```

The *time* can be any of the following:

```
hour  { AM }
      { PM }

hour:minute  [ AM ]
             [ PM ]

hour:minute:second  [ AM ]
                    [ PM ]
hour:minute:second:millisecond  [ AM ]
                                [ PM ]

hour:minute:second:millisecond  [ AM ]
                                 [ PM ]
```

The `mixed` *datetime* can be any of the following:

```
month [ , ] day [ , ] time year

day [ , ] month [ , ] time year
```

Table 4-5 lists the datetime fields.

Table 4-5. Datetime Fields

Field	Meaning	Comments and Valid Values
day	Day within a month	1 or 01 to 31
month	Month within a year (in alphabetic format)	January to December
monthn	Month in numeric format	1 or 01 to 12
year	Year	1 to 4 digits
year4	4-digit year	Must be 4 digits
hour	Hour within a day	0 or 00 to 23
minute	Minute within an hour	0 or 00 to 59
second	Second within a minute	0 or 00 to 59
millisecond	Millisecond within a second	0 or 00 to 999

For more information, see the following documents:

For Information On	See
Transact-SQL data types	*Microsoft SQL Server Transact-SQL Reference*
NonStop SQL/MP data types	*NonStop SQL/MP Reference Manual*
Transact-SQL dateparts	Date Functions on page 4-19
	Microsoft SQL Server Transact-SQL Reference

Retrieving and Manipulating NonStop SQL/MP Data and Objects

If you use the NonStop ODBC Server to access data and objects that were created with NonStop SQL/MP, the NonStop ODBC Server converts the NonStop SQL/MP data types and values to Transact-SQL data types and values.

Table 4-6 summarizes the NonStop SQL/MP data types, corresponding Transact-SQL data types, and conversion information.

Table 4-6. Converting NonStop SQL/MP Data Types to Transact-SQL Data Types (page 1 of 3)

NonStop SQL/MP Data Type	Does the Object Accept Null Values?	Corresponding Transact-SQL Data Type
CHARACTER	–	CHAR if value is 0–255 VARCHAR if value is 256–4050
	x	VARCHAR
DATE	–	DATETIME

x Indicates the column accepts null values
– Indicates the column does not accept null values

Table 4-6. Converting NonStop SQL/MP Data Types to Transact-SQL Data Types (page 2 of 3)

NonStop SQL/MP Data Type	Does the Object Accept Null Values?	Corresponding Transact-SQL Data Type
	x	DATETIMN
DATETIME	–	DATETIME
	x	DATETIMN
DECIMAL (l,s)		
l <= 4 and s=0	–	SMALLINT
4 < l <= 9 and s=0		INT
l > 9 or s <> 0		FLOAT
l <= 4 and s=0	x	INTN
4 < l <= 9 and s=0		INTN
l > 9 or s <> 0		FLOATN
DOUBLE PRECISION	– x	FLOAT FLOATN
FLOAT	–	FLOAT
	x	FLOATN
INTEGER signed	–	INT
	x	INTN
INTEGER unsigned	–	FLOAT
	x	FLOATN
INTERVAL	Not supported	
LARGEINT	–	FLOAT
	x	FLOATN
NUMERIC (l,s)		
l <= 4 and s=0	–	SMALLINT
4 < l <= 9 and s=0		INT
l > 9 or s <> 0		FLOAT
l <= 4 and s=0	x	INTN
4 < l <= 9 and s=0		INTN
l > 9 or s <> 0		FLOATN
PIC S9V9 COMP to PIC S9(18) COMP		
l <= 4 and s=0	–	SMALLINT
4 < l <= 9 and s=0		INT

x Indicates the column accepts null values
– Indicates the column does not accept null values

Table 4-6. Converting NonStop SQL/MP Data Types to Transact-SQL Data Types (page 3 of 3)

NonStop SQL/MP Data Type	Does the Object Accept Null Values?	Corresponding Transact-SQL Data Type
l > 9 or s <> 0		FLOAT
l <= 4 and s=0	x	INTN
4 < l <= 9 and s=0		INTN
l > 9 or s <> 0		FLOATN
PIC S9V9 DISPLAY to PIC S9(18) DISPLAY		
l <= 4 and s=0	–	SMALLINT
4 < l <= 9 and s=0		INT
l > 9 or s <> 0		FLOAT
l <= 4 and s=0	x	INTN
4 < l <= 9 and s=0		INTN
l > 9 or s <> 0		FLOATN
PIC X DISPLAY	–	CHAR if value is 0–255 VARCHAR if value is 256–4050
	x	VARCHAR
REAL	–	FLOAT
	x	FLOATN
SMALLINT signed	–	SMALLINT
	x	INTN
SMALLINT unsigned	–	INT
	x	INTN
TIME	–	DATETIME
	x	DATETIMN
TIMESTAMP	Not supported	
VARCHAR	–	VARCHAR
	x	VARCHAR

x Indicates the column accepts null values
– Indicates the column does not accept null values

Functions

Transact-SQL provides several types of functions for manipulating and retrieving data. This subsection summarizes NonStop ODBC Server support of Transact-SQL functions, then describes, by the following function types, how to use the supported functions:

- Date functions
- Mathematical functions
- String functions
- System functions

Summary of Transact-SQL Functions

Table 4-7 summarizes the supported Transact-SQL functions. Descriptions of each function follow the table.

Table 4-7. Supported Transact-SQL Functions

Function Type	Function	Comments
Date	DATEADD	See Date Functions on page 4-19 for descriptions of all date functions.
	DATEDIFF	
	DATEPART	
	GETDATE	
Mathematical	EXP	Fully supported
	PI	Fully supported
	POWER	Fully supported
String	UPPER	Fully supported
System	DB_NAME	All parameters are ignored; the function always returns the current database name.
	SUSER_NAME	All parameters are ignored; the function always returns the current username.
	USER_ID	All parameters are ignored; the function always returns an integer used internally to identify a user.
	USER_NAME	All parameters are ignored; the function always returns the current username.
Conversion	CONVERT	The style parameter is ignored, and there are restrictions when converting to or from a datetime value. See Conversion Function on page 4-24.

For a complete list of the Transact-SQL functions, see Appendix B, Summary of Support for SQL Server Features.

Date Functions

The NonStop ODBC Server supports the following Transact-SQL date functions, which manipulate datetime data:

- DATEADD
- DATEDIFF
- DATEPART
- GETDATE

Dateparts used within the functions have some restrictions, which are described following the individual function descriptions.

DATEADD

The DATEADD function has the following syntax:

```
DATEADD ( datepart , number , date )
```

DATEADD returns a datetime value equal to the sum of *`date`* plus the *`number`* of *`dateparts`*. Do not use a variable for *`date`*.

DATEDIFF

The DATEDIFF function has the following syntax:

```
DATEDIFF ( datepart , date-1 , date-2 )
```

DATEDIFF returns a signed integer value equal to *`date-2`* minus *`date-1`* in units of the specified *`datepart`*.

DATEPART

The DATEPART function has the following syntax:

```
DATEPART ( datepart , date )
```

DATEPART returns the specified *`datepart`* of the specified *`date`*, expressed as an integer.

GETDATE

The GETDATE function has the following syntax:

```
GETDATE ( )
```

GETDATE returns the current date and time in SQL Server’s standard internal format for datetime values. GETDATE takes no arguments.

Dateparts in Date Functions

Transact-SQL date functions contain arguments called dateparts. Table 4-8 summarizes the dateparts supported by the NonStop ODBC Server.

Table 4-8. NonStop ODBC Server Support of Transact-SQL Dateparts

Datepart	Abbreviation	NonStop ODBC Server Support
day	dd	x
dayofyear	dy	–
hour	hh	x
millisecond	ms	x
minute	mi	x
month	mm	x
quarter	qq	–
second	ss	x
week	wk	–
weekday	wd	–
year	yy	x

x Indicates a supported datepart
– Indicates an unsupported datepart

If you use an unsupported datepart, an error message is generated and the statement does not execute.

Results of Month Operations in Date Functions

Date functions used through the NonStop ODBC Server are executed by NonStop SQL/MP. The results of date operations, therefore, follow NonStop SQL/MP rules when Transact-SQL and NonStop SQL/MP have different rules for processing dates.

For example, the following statement is valid in Transact-SQL, but generates an error message when executed through the NonStop ODBC Server:

```
select dateadd (month, 1, '1/31/1980')

Msg 18001, Level 16, State 1:
NonStop SQL message (8425) : Either an invalid date was input or the result
of the date-time expression produced an invalid date.
```

When executed in Transact-SQL, this statement produces the date 2/29/1980.

For information on NonStop SQL/MP datetime data, see “Language Elements” in the *NonStop SQL/MP Reference Manual.*

Generating Dates Earlier Than 1753

Transact-SQL and NonStop SQL/MP allow different year ranges:

Transact-SQL	1753-9999
NonStop SQL/MP	0001-9999

Although the NonStop ODBC Server allows you to specify dates only in the range allowed by Transact-SQL, you can obtain dates earlier than 1753 by using negative numeric literals in a datetime function.

For example, the following statement produces a 1752 date when executed using the NonStop ODBC Server:

```
select dateadd (day, -1, 'jan 1 1753')

 --------------------
  Dec 30 1752 12:00AM
(1 row affected)
```

This statement would cause an overflow error message if executed in Transact-SQL.

Limitations on Specifying Milliseconds

Transact-SQL and NonStop SQL/MP allow different millisecond ranges:

Transact-SQL	Up to 3 digits
NonStop SQL/MP	Up to 6 digits

The NonStop ODBC Server allows you to specify milliseconds only in the range allowed by NonStop SQL/MP. For example, the following statement is valid in Transact-SQL but generates an error when executed through the NonStop ODBC Server:

```
select dateadd (millisecond, 9999, 'jan 2 1994')

Msg 18001, Level 15, State 1:
NonStop SQL message (3096) : INTERVAL value is out of range.
```

The following statement, however, specifies a valid NonStop SQL/MP millisecond range and can be executed through the NonStop ODBC Server:

```
select dateadd (millisecond, 999, 'jan 2 1994')

 --------------------
  Jan  2 1994 12:00AM

(1 row affected)
```

Variable Expressions in Date Functions

The NonStop ODBC Server does not support variable expressions in date functions. For example, the following statement is not allowed:

```
select dateadd ( month, @v1 + @v2, getdate() )
```

Simple variables, however, are allowed in date functions.

For more information, see the following documents:

For Information On	See
Transact-SQL date functions	*Microsoft SQL Server Transact-SQL Reference*
Datetime data	Data Types on page 4-12.

Mathematical Functions

The NonStop ODBC Server supports the following Transact-SQL mathematical functions:

- EXP
- PI
- POWER

All of these functions are fully supported.

EXP

The EXP function has the following syntax:

```
EXP ( floating-expression )
```

EXP returns the exponential value of the specified argument.

PI

The PI function has the following syntax:

```
PI ( )
```

PI returns the constant value 3.1415926535897936.

POWER

The POWER function has the following syntax:

```
POWER ( numeric-expression-1,numeric-expression-2 )
```

POWER returns the value of *`numeric-expression-1`* raised to the power of *`numeric-expression-2`*. The expressions can be of integer, float, or money type. The result is of the same data type as *`numeric-expression-1`*.

For more information on Transact-SQL mathematical functions, see the *Microsoft SQL Server Transact-SQL Reference*.

String Functions

The NonStop ODBC Server supports one Transact-SQL string function:

```
UPPER ( character-expression )
```

UPPER returns the value of *`character-expression`*, shifted to uppercase characters.

For more information on Transact-SQL string functions, see the *Microsoft SQL Server Transact-SQL Reference*.

System Functions

The NonStop ODBC Server supports the following Transact-SQL system functions:

- DB_NAME
- SUSER_NAME
- USER_ID
- USER_NAME

All parameters of these functions are ignored.

DB_NAME

The DB_NAME function has the following syntax:

```
DB_NAME ( [ database-id ] )
```

`database-id`

is not supported. The NonStop ODBC Server ignores this parameter.

When executed using the NonStop ODBC Server, DB_NAME returns the current database name in the format *`node_volume_subvolume.`* The current database is the database specified on the most recent USE statement. When you log on to the NonStop ODBC Server, the database name specified for the current user in the ZNSUS table; if no rows exist in ZNSUS that map to the current logon name, the master database is used by default.

SUSER_NAME

The SUSER_NAME function has the following syntax:

```
SUSER_NAME ( [ server-user-id ] )
```

server-user-id

is not supported. The NonStop ODBC Server ignores this parameter.

When executed using the NonStop ODBC Server, SUSER_NAME returns the logical username of the current user. The current user is the user who is logged on.

USER_ID

The USER_ID function has the following syntax:

```
USER_ID ( [ user-name ] )
```

user-name

is not supported. The NonStop ODBC Server ignores this parameter.

When executed using the NonStop ODBC Server, USER_ID returns an integer that is the numeric user ID of the current user.

USER_NAME

The USER_NAME function has the following syntax:

```
USER_NAME ( [ user-id ] )
```

user-id

is not supported. The NonStop ODBC Server ignores this parameter.

When executed using the NonStop ODBC Server, USER_NAME returns the logical username of the current user. The current user is the user who is logged on.

For more information on Transact-SQL system functions, see the *Microsoft SQL Server Transact-SQL Reference*.

Conversion Function

The NonStop ODBC Server supports the Transact-SQL conversion function, CONVERT. The CONVERT function has the following syntax:

```
CONVERT ( data-type , expression [ , style ] )
```

data-type

is the data type into which the expression is to be converted. It can be any supported data type.

expression

is the value to be converted.

style

is not supported. The NonStop ODBC Server ignores this parameter.

Converting From a Datetime Value to a Character Value

When converting a datetime value to a character value, the result is displayed in the format year to fraction 6, as shown in the following example:

```
select convert (char(26), getdate ())
```

Year to fraction 6 values require 26 characters, so the *data-type* must be at least 26 characters long.

For information on year to fraction data, see the *NonStop SQL/MP Reference Manual*.

Converting a Character Column Value or Local Variable to a Datetime Value

When converting a character column value or local variable to a datetime value, the format of the data in the column or local variable must be year to fraction 3. For example, if the START_DATE column in the following example is a character field, the data must be in the form year to fraction 3 (such as “1994-01-05:12:29:00.123”).

```
select convert (datetime, table1.start_date)
```

Converting a Character String to a Datetime Value

When converting a character literal to a datetime value, the *expression* can be any supported datetime format.

```
select convert (datetime, '1994')
```

Blank Padding of Character Data

When you convert a character expression to character data of a longer size, the value is blank-padded on the right.

Truncating Significant Digits

When converting between types with a different number of decimal points, the value is truncated if trailing significant digits will be lost. If, however, leading significant digits will be lost, an error message is issued and the statement does not execute.

Variables

You can use both local and global variables when using the NonStop ODBC Server. Local variables, however, have some usage restrictions, and not all global variables are supported. Also, you cannot declare values for global variables.

This subsection contains information on both local and global variables.

Local Variables

You can use local variables when using the NonStop ODBC Server, naming the variables according to Transact-SQL naming rules. Using variables in the NonStop ODBC Server, however, differs from using them in Transact-SQL in the following ways.

Referencing Null Variables

A variable cannot be null when it is referenced. For example, the following statements will result in an error:

```
declare @v1 int, @v2 int
select @v2 = @v1
```

Nonaggregate and Aggregate Assignments to Variables

Variable assignments cannot contain a mix of nonaggregate and aggregates. For example, the following statement is not allowed:

```
select @v1 = c1, @v2 = max (c2) from t1
```

An alternative is to divide the statement into two statements:

```
select @v1 = c1 from t1
select @v2 = max (c2) from t1
```

For more information, see the following documents:

For Information On	See
Declaring variables	DECLARE on page 4-61.
Assigning values to variables	SELECT on page 4-82.
Using variables in Transact-SQL	*Microsoft SQL Server Transact-SQL Reference*

Global Variables

You can use Transact-SQL global variables when using the NonStop ODBC Server; however, not all global variables are supported.

Table 4-9 summarizes the global variables and whether they are supported by the NonStop ODBC Server.

Table 4-9. Global Variables

Global Variable	NonStop ODBC Server Support	Comments or value assigned
@@connections	x	The value is always 1.
@@cpu_busy	–	-1
@@error	x	No differences.
@@idle	–	-1
@@io_busy	–	-1
@@max_connections	x	Depends on the system resources.
@@next_level	–	-1
@@pack_received	–	-1
@@pack_sent	–	-1
@@packet_errors	–	-1
@@procid	–	-1
@@rowcount	–	0
@@textsize	x	The default is 512. (In Transact-SQL, it is 4058.)
@@timeticks	–	-1
@@total_errors	–	-1
@@total_read	–	-1
@@total_write	–	-1
@@trancount	x	The value will be either 0 or 1 because nested transactions are not supported. The default is 0.
@@version	–	NonStop ODBC Server T8666D20

x Indicates a supported global variable
– Indicates an unsupported global variable

If you use an unsupported global variable, the NonStop ODBC Server returns the value indicated in Table 4-9.

Global Variables in a Batch

In Transact-SQL, global variables must be used within a batch. In the NonStop ODBC Server, global variables do not need to be in a batch.

Declaring Values for Global Variables

In Transact-SQL, you can declare values for global variables (for the current batch only). In the NonStop ODBC Server, you cannot declare values for global variables.

Search Conditions

Use Transact-SQL syntax in search conditions when using the NonStop ODBC Server. All supported search conditions work the same as in Transact-SQL.

The syntax of a search condition is any of the following:

```
WHERE expression comparison-operator expression

WHERE [ NOT ] column-name [ NOT ] LIKE "match-string"

WHERE [ NOT ] column-name IS [ NOT ] NULL

WHERE expression [ NOT ] BETWEEN expression AND expression

WHERE expression [ NOT ] IN  (  { value-list }  )
                                { subquery   }

WHERE [ NOT ] EXISTS ( subquery )

WHERE expression comparison-operator  { ANY }  ( subquery )
                                      { ALL }

WHERE [ NOT ] column-name join-operator column-name

WHERE boolean-expression

WHERE expression  { AND }  expression
                  { OR  }
```

For more information on search conditions, see the *Microsoft SQL Server Transact-SQL Reference*.

Expressions and Operators

Use Transact-SQL syntax in expressions when using the NonStop ODBC Server.

The syntax of an expression is:

```
operand operator operand [ operator operand ] ...

operand is:

  { constant      }
  { column-name   }
  { function      }
  { ( subquery )  }
  { variable      }

operator is:

  [ arithmetic-operator ]
  [ bitwise-operator    ]
  [ string-operator     ]
```

Operators are summarized on the following pages.

The syntax of a Boolean expression is:

```
expression comparison-operator  [ ANY ]  expression
                                [ ALL ]

expression [ NOT ] IN expression

[ NOT ] EXISTS expression

expression [ NOT ] BETWEEN expression AND expression

expression [ NOT ] LIKE expression

NOT expression LIKE expression

expression IS [ NOT ] NULL

NOT boolean-expression

boolean-expression AND/OR boolean-expression

[ NOT ] boolean-function
```

Table 4-10 summarizes the supported and unsupported Transact-SQL operators.

Table 4-10. NonStop ODBC Server Support of Transact-SQL Operators

Transact-SQL Operator Type	Symbol	NonStop ODBC Server Support
Arithmetic	+	x
	-	x
	*	x
	/	x
	%	x
Bitwise	&	–
	|	–
	^	–
	~	–
Comparison	=	x
	>	x
	<	x
	>=	x
	<=	x
	!=	x
	!>	x
	!<	x
	*=	–
	=*	–
Other	ALL	x
	AND	x
	ANY	x
	BETWEEN	x
	EXISTS	x
	IN	x
	IS NULL	x
	LIKE	x
	NOT	x
	OR	x

x Indicates a supported operator
– Indicates an unsupported operator

LIKE Operator—Differences From SQL Server

When used through the NonStop ODBC Server, the behavior of the LIKE operator differs from its behavior in SQL Server in the following ways:

Feature	In SQL Server	In the NonStop ODBC Server
Are trailing blanks significant?	No	Yes
Do character comparisons distinguish between uppercase and lowercase?	No	Yes

For more information, see the following documents:

For Information On	See
Expressions	*Microsoft SQL Server Transact-SQL Reference*
Boolean expressions	*Microsoft SQL Server Transact-SQL Reference*

Aggregates

The NonStop ODBC Server supports all of the Transact-SQL aggregate functions (AVG, COUNT, MAX, MIN, and SUM).

Table 4-11 summarizes the primary differences between using aggregate functions with SQL Server and using them with the NonStop ODBC Server.

Table 4-11. NonStop ODBC Server Support of Aggregate Functions

Feature	In SQL Server	In the NonStop ODBC Server
Data type of the results of AVG, COUNT, and SUM	Depends on the expression being evaluated.	Always floating point
Data type of the results of MAX and MIN	Depends on the expression being evaluated.	Depends on the expression being evaluated. An INT expression returns a floating point value.
What is the default if the DISTINCT clause is omitted from COUNT?	All rows are counted	Only distinct rows are counted.
Are aggregates and row aggregates distinguished?	Yes	No, because the row aggregates are part of the Transact-SQL COMPUTE BY clause, which is not supported by the NonStop ODBC Server.

AVG

The AVG aggregate function has the following syntax:

```
AVG ( [ DISTINCT ] expression )
```

AVG returns a floating point value that represents the average of the values in the column identified by *expression*; null values are ignored. If you include DISTINCT, AVG eliminates duplicate values before calculating the average. AVG can be used with numeric columns only.

If you include DISTINCT when executing AVG using the NonStop ODBC Server, *expression* cannot be a constant. (The *expression* can be a constant when using Transact-SQL.)

COUNT

The COUNT aggregate function has the following syntax:

```
COUNT ( { DISTINCT column-name } )
        { column-name          }
        { *                    }
```

COUNT returns a floating-point value that represents the number of nonnull values in the column identified by *column-name*. If you include DISTINCT, COUNT eliminates duplicate values before making the count.

If only *column-name* is specified, the column data type must be a numeric data type such as NUMERIC and not a nonnumeric type such as CHAR.

If you use * instead of *column-name*, COUNT returns the number of rows, regardless of the presence of null values; * and DISTINCT cannot be used together.

If you omit the DISTINCT keyword when using the NonStop ODBC Server, the default is DISTINCT. If you omit DISTINCT when executing COUNT using SQL Server, however, all rows are counted.

If you include DISTINCT when using the NonStop ODBC Server, *column-name* cannot be a constant. (The *column-name* can be a constant when using Transact-SQL.)

(NonStop SQL/MP does not support the COUNT function when only *column-name* is specified in a query; however, NonStop ODBC has implemented the COUNT function for ODBC clients that need this function.)

MAX

The MAX aggregate function has the following syntax:

```
MAX ( expression )
```

MAX returns the maximum value in the column identified by *expression*. MAX can be used with numeric, character, and date columns, but not with bit columns. With character columns, MAX returns the value that is highest in the collating sequence. MAX ignores null values.

MAX returns a floating-point value for INT expressions. For other types of expressions, the value returned depends on the value of the expression being evaluated.

MIN

The MIN aggregate function has the following syntax:

```
MIN ( expression )
```

MIN returns the minimum value in the column identified by *`expression`*. MIN can be used with numeric, character, and date columns, but not with bit columns. With character columns MIN returns the value that is lowest in the collating sequence. MIN ignores null values.

MIN returns a floating-point value for INT expressions. For other types of expressions, the value returned depends on the value of the expression being evaluated.

SUM

The SUM aggregate function has the following syntax:

```
SUM ( [ DISTINCT ] expression )
```

SUM returns a floating-point value that represents the sum of all values in the column identified by expression. If you include DISTINCT, SUM eliminates duplicate values before adding the values. SUM ignores null values. SUM can be used with numeric columns only.

If you include DISTINCT when using the NonStop ODBC Server, the *`expression`* cannot be a constant. (The *`expression`* can be a constant when using Transact-SQL.)

For more information, see the following documents:

For Information On	**See**
Aggregates in Transact-SQL	*Microsoft SQL Server Transact-SQL Reference*
Aggregates in NonStop SQL/MP	*NonStop SQL/MP Reference Manual*

Wild-Card Characters

Wild-card characters are used with the LIKE operator to represent any character in a string. Table 4-12 summarizes NonStop ODBC Server support of Transact-SQL wild-card characters.

Table 4-12. NonStop ODBC Server Support of Wild-Card Characters

Transact-SQL Wild-Card Character	NonStop ODBC Server Support
%	x
_	x
[*character-range*]	–
[*character-set*]	–
[^*character-range*]	–
[^*character-set*]	–

x Indicates a supported wild-card character
– Indicates an unsupported wild-card character

When used with the NonStop ODBC Server, the % wild-card character works differently than it does in Transact-SQL:

- In the NonStop ODBC Server, character comparisons are case sensitive. In Transact-SQL, however, character comparisons are not case sensitive.

  For example, the following comparison would locate the string “San Francisco” if done with Transact-SQL, but not if done with the NonStop ODBC Server:

  ```
  %san%
  ```

- Because NonStop SQL/MP recognizes trailing blanks as part of a string, you must place the % wild-card character at the end of a comparison; this is not necessary in Transact-SQL.

  For example, when searching a 12-character field containing “ABC”, Transact-SQL sees the value as “ABC” and NonStop SQL/MP sees it as “ABC   ” (with nine trailing blanks). Consequently, when searching for the string using the NonStop ODBC Server, you must use the following comparison:

  ```
  %ABC%
  ```

For more information, see the following documents:

For Information On	See
Wild-card characters in Transact-SQL	*Microsoft SQL Server Transact-SQL Reference*
Wild-card characters in NonStop SQL/MP	*NonStop SQL/MP Reference Manual*

NULL Values

The only difference between NULL values in Transact-SQL and NULL values in the NonStop ODBC Server is the order in which data is displayed.

Table 4-13 summarizes how NULL values affect the order in which data is displayed in Transact-SQL and in statements executed using the NonStop ODBC Server.

Table 4-13. The Order in Which NULL Values Are Displayed

Clause	In Transact-SQL	When Executed Using the NonStop ODBC Server
GROUP BY	NULL values form their own group.	Same as Transact-SQL
ORDER BY	NULL values come before all others.	NULL values come after all others.
SELECT statement with the DISTINCT keyword	NULL values are considered duplicates of each other. Only one NULL is selected, no matter how many are encountered.	Same as Transact-SQL

Comments

User-written comments annotate SQL statements and statement blocks.

The syntax of a comment is as follows:

```
/* text-of-the-comment */
```

Comments in the NonStop ODBC Server work the same as comments in Transact-SQL except that comments in the NonStop ODBC Server cannot be nested.

Summary of Statements

The SQL statements you can use with the NonStop ODBC Server are a subset of the SQL Server Transact-SQL statements. This section lists the Transact-SQL statements and summarizes how each supported statement differs when used with the NonStop ODBC Server. The statements are divided into the following categories:

- SQL statements
- Control-of-flow statements
- Defaults, rules, and triggers
- Stored procedures
- Storage allocation
- Transaction logging
- Miscellaneous statements

Ordinarily, if you use an unsupported statement with the NonStop ODBC Server, an error message is generated. You can establish a user profile option, however, to ignore errors from unsupported features. This could be useful when running an application that uses a feature whose action is not required, such as GRANT and REVOKE.

Table 4-14 summarizes NonStop ODBC Server support of SQL statements.

Table 4-14. NonStop ODBC Server Support of Transact-SQL Statements (page 1 of 3)

Transact-SQL Statement	NonStop ODBC Server Support	Comments
ALTER TABLE	x	Supports the following syntax: ALTER TABLE ADD (*column-identifier data-type* [,*column-identifier data-type*])
BEGIN TRANSACTION	x	Nested transactions are not allowed. Transaction names can be included but are not meaningful. DDL operations are allowed within a transaction.
COMMIT TRANSACTION	x	Transaction savepoints are ignored. Transaction names can be included but are ignored.
CREATE DATABASE	x	The ON DEFAULT clause is not supported. You cannot specify the amount of space to allocate for the database. NonStop SQL/MP allocates disk space for tables and other objects as you create them and enter data.
CREATE [UNIQUE] INDEX	x	The following clauses are not supported: ● CLUSTERED ● WITH index-option You cannot create a unique index on a column that allows null values. The fully expanded index name must be unique in the network.
CREATE TABLE	x	Some data types are unsupported, and the NonStop ODBC Server maps some data types to NonStop SQL/MP data types. The fully expanded table name must be unique in the network. The following clauses are not supported: ● UNIQUE and PRIMARY KEY for column-constraint-definition ● Column-constraint-definition of REFERENCES ● Table-constraint-definition of UNIQUE, CHECK, or FOREIGN KEY

x Indicates that the statement is supported
– Indicates that the statement is not supported

Table 4-14. NonStop ODBC Server Support of Transact-SQL Statements (page 2 of 3)

Transact-SQL Statement	NonStop ODBC Server Support	Comments
CREATE VIEW	x	You can modify data in a view only if the view is derived from one table. The fully expanded view name must be unique in the network.
DELETE	x	The FROM clause is not supported. You cannot specify more than one table name—you cannot delete rows based on data stored in other tables. DELETE must be inside a transaction if the table is audited.
DROP DATABASE	x	Fully supported.
DROP INDEX	x	The following associated objects must be accessible: ● The underlying table ● Catalogs containing the description of the index
DROP TABLE	x	Dependent views are automatically dropped, but you should drop them first because they are not dropped from the mapping tables. Use REFRESH afterward.
DROP VIEW	x	Dependent views are automatically dropped. Use REFRESH afterward.
EXECUTE	x	Execution of stored procedures is supported, but procedures must be created in the Pathway environment.
INSERT	x	Fully supported.
ROLLBACK TRANSACTION	x	Transaction savepoints are ignored.
SAVE TRANSACTION	x	If used in the NonStop ODBC Server, the SAVE TRANSACTION statement is ignored. No error or warning messages are generated.

x Indicates that the statement is supported
– Indicates that the statement is not supported

Table 4-14. NonStop ODBC Server Support of Transact-SQL Statements (page 3 of 3)

Transact-SQL Statement	NonStop ODBC Server Support	Comments
SELECT	x	The following clauses are not supported: ● INTO ● GROUP BY ALL ● COMPUTE BY ● FOR BROWSE You cannot include expressions in an aggregate-free-expression. Vector aggregates are not supported. You cannot mix aggregate and nonaggregate expressions in the select-list.
TRUNCATE TABLE	x	TRUNCATE TABLE works at the same speed as DELETE. Changes are written to the audit log. TRUNCATE TABLE must be inside a transaction if the table is audited.
UPDATE	x	The FROM clause is not supported. You can update data in a view only if the view is derived from one table. UPDATE must be inside a transaction if the table is audited. Aggregate functions are not allowed in the expression.
UPDATE STATISTICS	x	You cannot specify the index for which to update statistics.
USE	x	The new database takes effect as soon as the statement is executed, not when the current batch finishes.

x Indicates that the statement is supported
– Indicates that the statement is not supported

Table 4-15 summarizes NonStop ODBC Server support of the control-of-flow statements.

Table 4-15. NonStop ODBC Server Support of Control-of-Flow Statements

Transact-SQL Statement	NonStop ODBC Server Support	Comments
BEGIN...END	x	These statements are fully supported; however, in SQL Server, BEGIN and END are used primarily to enclose control-of-flow statements (such as IF, THEN, ELSE, and WHILE), which are not supported by the NonStop ODBC Server.
BREAK	–	–
CONTINUE	–	–
DECLARE	x	Some data types are unsupported, and the NonStop ODBC Server maps some data types to NonStop SQL/MP data types. For information on usage differences, see Variables on page 4-26.
GOTO	–	–
IF...ELSE	–	–
PRINT	x	Fully supported.
RAISERROR	–	–
RETURN	–	–
WAITFOR	–	–
WHILE	–	–

x Indicates that the statement is supported
– Indicates that the statement is not supported

Defaults, Rules, and Triggers

The NonStop ODBC Server does not support defaults, rules, or triggers. The unsupported Transact-SQL statements are:

CREATE DEFAULT
CREATE RULE
CREATE TRIGGER
DROP DEFAULT
DROP RULE
DROP TRIGGER

You can use the NonStop SQL/MP CREATE CONSTRAINT and DROP CONSTRAINT statements in pass-through mode as a partial alternative to CREATE RULE and DROP RULE. As an alternative to CREATE DEFAULT, you can use the DEFAULT parameter on the NonStop SQL/MP CREATE TABLE and ALTER TABLE statements in pass-through mode.

Miscellaneous Transact-SQL Statements

The NonStop ODBC Server does not support the following Transact-SQL statements:

GRANT
KILL
READTEXT
REVOKE
SETUSER
SHUTDOWN
WRITETEXT

GRANT and REVOKE can be processed and ignored by setting a user profile option.

For the SET statement, only two parameters are supported:

PARSEONLY
TEXTSIZE

SQL statements are divided into the following categories:

- Control-of-flow statements
- Data Control Language (DCL) statements
- Data Definition Language (DDL) statements
- Data Manipulation Language (DML) statements
- Transaction management statements
- Miscellaneous statements

Table 4-16 summarizes the supported Transact-SQL statements by statement type.

Table 4-16. Supported Transact-SQL Statements by Type (page 1 of 2)

Transact-SQL Statement	Description	Corresponding NonStop SQL/MP Statement
Control-of-Flow Statements		
BEGIN...END	Executes the set of statements within the BEGIN and END statements.	None
DECLARE	Defines a local variable.	None
PRINT	Returns a user-defined message to the user’s application.	None
DDL Statements		
ALTER TABLE	Adds new columns to an existing table.	ALTER TABLE ADD COLUMN
CREATE DATABASE	Creates a new NonStop SQL/MP catalog.	CREATE CATALOG

Table 4-16. Supported Transact-SQL Statements by Type (page 2 of 2)

Transact-SQL Statement	Description	Corresponding NonStop SQL/MP Statement
CREATE INDEX	Creates indexes.	CREATE INDEX
CREATE TABLE	Creates new tables.	CREATE TABLE
CREATE VIEW	Creates views.	CREATE VIEW
DROP DATABASE	Removes one or more catalogs.	DROP CATALOG
DROP INDEX	Removes an index.	DROP INDEX
DROP TABLE	Removes a table, along with all data, indexes, dependent views, and permission specifications for that table.	DROP TABLE
DROP VIEW	Removes views.	DROP VIEW
DELETE	Removes rows from a table.	DELETE
DML Statements		
INSERT	Adds new rows to a table or view.	INSERT
SELECT	Retrieves rows from tables.	SELECT
TRUNCATE TABLE	Removes all rows from a table without logging deletions.	None
UPDATE	Changes data in existing rows, either by adding new data or modifying existing data.	UPDATE
Transaction Management Statements		
BEGIN TRANSACTION	Marks the starting point of a user-specified transaction.	BEGIN WORK
COMMIT TRANSACTION	Marks the ending point of a user-specified transaction. All database changes since the preceding BEGIN TRANSACTION statement become permanent.	COMMIT WORK
ROLLBACK TRANSACTION	Rolls back all changes to the database that occurred since the last BEGIN TRANSACTION statement was executed.	ROLLBACK WORK
SAVE TRANSACTION	Sets a savepoint within a transaction.	None
Miscellaneous Statements		
EXECUTE	Invokes a stored procedure.	None
SET	Sets query-processing options for the duration of a work session.	None
UPDATE STATISTICS	Updates all statistics associated with all columns of a specified table.	UPDATE STATISTICS
USE	Causes SQL Server to use the specified database.	CATALOG or VOLUME

None of the Transact-SQL DCL statements are supported.

ALTER TABLE

Use ALTER TABLE to add new columns to an existing table. The new column appears as the last column of the table.

The ALTER TABLE statement has the following syntax:

```
ALTER TABLE  [ [database.]owner. ]  table-name

ADD column-name data-type NULL

[ , column-name data-type NULL ] ...
```

table-name

is the table to alter. The table name can be qualified with the database name and owner name. The fully expanded table name must be unique in the network.

column-name

is the column to add. The column name must be unique for columns defined in the table.

If the statement contains errors, the results may be different than if executed using SQL Server. See Adding Multiple Columns on page 4-43.

data-type

is the SQL Server data type for the column. Data types are converted to NonStop SQL/MP data types. Data types are described under Language Elements on page 4-5.

If you specify an unsupported data type, an error message is generated and the statement is not executed.

NULL

is a required keyword that sets the initial values in the new column to NULL in existing rows.

Examples

The following statement adds two columns to the DEPT table. The columns contain BIT type data and are initialized as NULL:

```
use test_disk01_persnl

alter table dept
   add deptnew   int null,
       deptclose int null
```

The DEPT table is registered in the NonStop SQL/MP catalog PERSNL on the disk volume DISK01 on the node TEST.

Adding Multiple Columns

If you add multiple columns with one ALTER TABLE statement and the statement contains errors, the results can be different than if the statement were executed using SQL Server.

Because NonStop SQL/MP does not allow you to add multiple columns with one statement, the NonStop ODBC Server translates the ALTER TABLE statement to multiple NonStop SQL/MP ALTER TABLE statements. Therefore, even if one of the statements contains an error, the other statements will be executed and will add the specified columns. In SQL Server, if one of the *`column-name`* clauses contains an error, the statement fails and none of the columns are added.

NonStop ODBC Server Compared With SQL Server

When used through the NonStop ODBC Server, ALTER TABLE differs from the SQL Server implementation in the following ways:

Feature	**In SQL Server**	**In Programs Used With the NonStop ODBC Server**
Can ALTER TABLE be inside a transaction?	No	Yes, if the table is audited.
Can you add BIT type columns?	No	Yes. The NonStop ODBC Server maps the BIT data type to the NonStop SQL/MP SMALLINT data type.
Maximum sum of the lengths of all columns for a table.	1962 bytes	The maximum row length allowed in NonStop SQL/MP, as given in the *NonStop SQL/MP Reference Manual.*

NonStop ODBC Server Compared With NonStop SQL/MP

The corresponding NonStop SQL/MP statement is ALTER TABLE ADD COLUMN.

When used through the NonStop ODBC Server, ALTER TABLE differs from the NonStop SQL/MP implementation in the following ways:

Feature	**In NonStop SQL/MP**	**In Programs Used With the NonStop ODBC Server**
Number of columns you can add with ALTER TABLE	One	Multiple
Can you specify default values?	Yes	No. You cannot specify a default value unless you use pass-through mode.
Is NOT NULL available?	Yes	No. New columns always allow null values. You cannot specify the NOT NULL attribute unless you use pass-through mode.
Can you alter attributes of a table or column using ALTER TABLE?	Yes	No. You cannot alter the attributes of a column unless you use pass-through mode.

BEGIN...END

Use BEGIN...END to enclose a series of statements.

You can include BEGIN and END in programs used with the NonStop ODBC Server; however, BEGIN and END are used primarily to enclose control-of-flow statements, which are not supported by the NonStop ODBC Server.

The BEGIN...END statement has the following syntax:

```
BEGIN

statement-block

END
```

statement-block

is a series of statements. Statement blocks can be nested (BEGIN...END within another BEGIN...END).

Examples

The following BEGIN and END statements enclose two INSERT statements:

```
begin
```

```
insert into employee
   (empnum, first_name, last_name, deptnum,
    jobcode, salary)
    values (2001, "Dana", "Hall", 1010, 3001, 40000)

insert into employee
   (empnum, first_name, last_name, deptnum,
    jobcode, salary)
    values (2002, "Fiona", "Stenhouse", 1020, 3001, 38000)

end
```

NonStop ODBC Server Compared With SQL Server

When used through the NonStop ODBC Server, BEGIN and END do not differ from the SQL Server implementation. You cannot, however, use SQL Server control-of-flow statements in programs used with the NonStop ODBC Server. BEGIN and END are implemented to provide compatibility with SQL Server programs.

NonStop ODBC Server Compared With NonStop SQL/MP

NonStop SQL/MP does not have a statement that corresponds to BEGIN...END.

BEGIN TRANSACTION

Use BEGIN TRANSACTION to mark the starting point of a user-specified transaction.

The BEGIN TRANSACTION statement has the following syntax:

```
BEGIN TRAN[SACTION] [ transaction-name ]
```

transaction-name

is the name associated with the transaction.

If you omit the *transaction-name*, no name is associated with this transaction.

Examples

The following transaction creates a table and inserts one row:

```
begin transaction

use test_disk01_persnl

create table employee
   (empnum       smallint,
    first_name   char(15),
    last_name    char(20),
    deptnum      smallint,
    jobcode      smallint,
    salary       money     null)
```

```
insert into employee
   (empnum, first_name, last_name, deptnum,
    jobcode, salary)
    values (2001, "Dana", "Hall", 1010, 3001, 40000)

commit transaction
```

NonStop ODBC Server Compared With SQL Server

When used through the NonStop ODBC Server, BEGIN TRANSACTION differs from the SQL Server implementation in the following ways:

Feature	**In SQL Server**	**In Programs Used With the NonStop ODBC Server**
Are nested transactions allowed?	Yes	No
Are transaction savepoints allowed?	Yes	Yes, but they are ignored. If you want to roll back a transaction, you must roll back the entire transaction.
Are DDL operations on a table allowed within a user-defined transaction?	No	Yes, if the table is audited. See DDL Statements Allowed in User-Defined Transactions on page 4-3.

NonStop ODBC Server Compared With NonStop SQL/MP

The corresponding NonStop SQL/MP statement is BEGIN WORK.

The primary difference between NonStop ODBC Server transactions and NonStop SQL/MP transactions is that, in programs used with the NonStop ODBC Server, you can specify a transaction name. The transaction name is provided primarily for compatibility with SQL Server and is not meaningful in NonStop SQL/MP (because transactions cannot be nested in NonStop SQL/MP).

COMMIT TRANSACTION

Use COMMIT TRANSACTION to mark the ending point of a user-specified transaction. All database changes since the preceding BEGIN TRANSACTION statement become permanent.

The COMMIT TRANSACTION statement has the following syntax:

```
COMMIT TRAN[SACTION] [ transaction-name ]
```

transaction-name

is the name associated with the transaction. The *transaction-name* is optional on COMMIT TRANSACTION, and is ignored.

In SQL Server, transaction names are needed when transactions are nested; however, since transactions cannot be nested in programs used with the NonStop ODBC Server, the *`transaction-name`* is not meaningful in these programs.

When used through the NonStop ODBC Server, the transaction is committed even if the *`transaction-name`* does not match the *`transaction-name`* on the previous BEGIN TRANSACTION statement. No error or warning messages are issued.

Examples

The following transaction creates a table and inserts one row:

```
begin transaction

use test_disk01_persnl

create table employee
   (employee_number    smallint      not null,
    first_name         char(15)      not null,
    last_name          char(20)      not null,
    deptnum            smallint      not null,
    job_code           smallint      not null,
    salary             money         null)

insert into employee
   (empnum, first_name, last_name, deptnum,
    jobcode, salary)
    values (2001, "Dana", "Hall", 1010, 3001, 40000)

commit transaction
```

NonStop ODBC Server Compared With SQL Server

When used through the NonStop ODBC Server, COMMIT TRANSACTION differs from the SQL Server implementation in the following ways:

Feature	**In SQL Server**	**In Programs Used With the NonStop ODBC Server**
Are transaction names allowed?	Yes	Yes, but they are ignored
Are nested transactions allowed?	Yes	No
Are transaction savepoints allowed?	Yes	Yes, but they are ignored.
Are DDL operations on a table allowed within a transaction?	No	Yes, if the table is audited See DDL Statements Allowed in User-Defined Transactions on page 4-3.

NonStop ODBC Server Compared With NonStop SQL/MP

The corresponding NonStop SQL/MP statement is COMMIT WORK.

The primary difference between NonStop ODBC Server transactions and NonStop SQL/MP transactions is that in programs used with the NonStop ODBC Server, you can specify a transaction name. The transaction name is provided primarily for compatibility with SQL Server and is not meaningful in NonStop SQL/MP (because transactions cannot be nested in NonStop SQL/MP).

In programs used with the NonStop ODBC Server, you cannot specify the NonStop SQL/MP AUDIT ONLY clause on the COMMIT TRANSACTION statement; therefore, all locks are released whenever COMMIT TRANSACTION is used.

CREATE DATABASE

Use CREATE DATABASE to create a new NonStop SQL/MP catalog, customized for use with the NonStop ODBC Server.

When you create a database using CREATE DATABASE in the NonStop ODBC Server, the NonStop ODBC Server creates a NonStop SQL/MP catalog and the appropriate mapping tables for that catalog. For information on customized catalogs, see Section 7, Managing Customized Catalogs.

The CREATE DATABASE statement has the following syntax:

```
CREATE DATABASE database-name

  [ ON  { DEFAULT         }  [ = size ]
        { database-device }

     [ ,  { DEFAULT         }  [ = size ] ] ...   ]
          { database-device }
```

database-name

is the subvolume in which the NonStop SQL/MP catalog is to be created.

The *database-name* is a subvolume optionally qualified with a node and volume. If you omit the node and volume, the *database-name* is expanded using the current volume and subvolume.

If the specified subvolume already contains a catalog, the catalog is customized.

ON DEFAULT = *size*

is not supported. In the NonStop ODBC Server, this clause is ignored. No error messages are generated.

`ON` *`database-device = size`*

is not supported. In the NonStop ODBC Server, this clause is ignored. No error messages are generated.

Examples

The following statement creates a NonStop SQL/MP catalog PERSNL on the disk volume DISK01 on the node called TEST:

```
create database test_disk01_persnl
```

NonStop ODBC Server Compared With SQL Server

When used through the NonStop ODBC Server, CREATE DATABASE differs from the SQL Server implementation in the following ways:

Feature	**In SQL Server**	**In Programs Used With the NonStop ODBC Server**
Can CREATE DATABASE be inside a transaction?	No	Yes See DDL Statements Allowed in User-Defined Transactions on page 4-3.
Are specific permissions needed?	Yes, you must have permission to create a database on the server.	Yes, you need the appropriate Guardian and Safeguard permissions. The user must have the ability to update the USAGES and CATALOGS tables in the system catalog, and to create files on the target subvolume.
Is a specific database required?	Yes, you must be in the MASTER database to create a database.	No
Who owns the database?	The user who creates it.	The user who creates it, unless the specified subvolume already contains a catalog. In that case, the database owner is the Guardian name of the owner of the NonStop SQL/MP catalog.
What security does the database have?	Only the database owner can access the database. The owner can grant other users permission to use the database.	Uses file-system security, which is obtained from the default permissions associated with the user's logon name

Feature	In SQL Server	In Programs Used With the NonStop ODBC Server
Must the database owner grant access permission to other users?	Yes, the owner must grant permission to other users before they can access the database.	No. The catalog uses file-system security. Any user with the appropriate security can access the catalog. For information on file-system security, see the *Guardian User's Guide*.
Can you specify the amount of space to allocate for the database?	Yes	No, NonStop SQL/MP allocates space on an as-needed basis. The only space initially allocated is the space for the NonStop SQL/MP catalog. Disk space for tables and other objects is allocated as you create them and enter data. If the disk is full, NonStop SQL/MP generates an error message.
Are there limits on the size and number of objects?	Yes, as follows: ● 32,767 databases per server ● 2 billion tables per database ● 250 columns per table ● 1962 bytes per row	Yes, the NonStop SQL/MP limits are used, which can affect the size of the database. See information on limits in the *NonStop SQL/MP Reference Manual.*
Do you need to allocate space for transaction logs?	Yes	No, the system manager allocates space on a system-wide basis.
Is auditing of the volume necessary?	N.A.	The volume on which the catalog is created must be audited by TMF.

NonStop ODBC Server Compared With NonStop SQL/MP

The corresponding NonStop SQL/MP statement is CREATE CATALOG.

CREATE INDEX

Use CREATE INDEX to create indexes.

The CREATE INDEX statement has the following syntax:

```
CREATE [ UNIQUE ]  { CLUSTERED    }  INDEX index-name
                   { NONCLUSTERED }

  ON  [  [database.]owner. ]  table-name

      ( column-name [ , column-name ] ...  )

[ WITH index-option [ , index-option ] ...  ]

index-option is:

     { FILLFACTOR = fill }
     { IGNORE_DUP_KEY    }
     { IGNORE_DUP_ROW    }
     { ALLOW_DUP_ROW     }
```

`UNIQUE`

specifies that two or more rows of the table cannot have the same values for the indexed columns.

`CLUSTERED, NONCLUSTERED`

indicates whether the physical order of rows in the table is controlled by the index.

Clustered indexes are not supported. CLUSTERED, if specified, is ignored. No error messages are generated, and the index remains nonclustered.

`index-name`

is the index to create.

`table-name`

the table in which the indexed column or columns are located. The *`table-name`* can be qualified with the database and owner.

`column-name`

is the column or columns to which the index applies.

In SQL Server, you can specify up to 16 columns. In NonStop SQL/MP, however, the number of columns that can be indexed depends on the length of the index key. For more information, see the *NonStop SQL/MP Reference manual.*

WITH FILLFACTOR = *fill*

is not supported. In the NonStop ODBC Server, this clause is ignored. No error messages are generated.

WITH IGNORE_DUP_KEY

is not supported. In the NonStop ODBC Server, this clause is ignored. No error messages are generated. Rows are rejected if they have a duplicate primary key.

WITH IGNORE_DUP_ROW, WITH ALLOW_DUP_ROW

is not supported. In the NonStop ODBC Server, this clause is ignored. No error messages are generated. Rows are rejected if they are not unique with respect to the index and the index is created with the UNIQUE option.

Examples

The following statements create indexes on the EMPLOYEE table:

```
use test_disk01_persnl
create unique index xempnum on employee (empnum)
create index xempname on employee (last_name, first_name)
```

The indexes are created on the subvolume PERSNL on the disk volume DISK01 on the node TEST. The indexes are registered in the NonStop SQL/MP catalog on that same subvolume.

NonStop ODBC Server Compared With SQL Server

When used through the NonStop ODBC Server, CREATE INDEX differs from the SQL Server implementation in the following ways:

Feature	**In SQL Server**	**In Programs Used With the NonStop ODBC Server**
Can CREATE INDEX be inside a transaction?	No	Yes See DDL Statements Allowed in User-Defined Transactions on page 4-3.
Is auditing of the volume necessary?	N.A.	Yes, the volume on which the index is created must be audited by TMF.
What locking mode is used?	Shared	The table is locked with a shared table lock while the index is created.
Maximum number of indexes that can exist for a table	250	Follows the rules for NonStop SQL/MP limits. See information on limits in the *NonStop SQL/MP Reference Manual.*

Feature	In SQL Server	In Programs Used With the NonStop ODBC Server
Can you create a unique index on a column that allows null values?	Yes	No
How many columns can be indexed?	Up to 16	Depends on the length of the index key. See information about “Limits” in the *NonStop SQL/MP Reference Manual.*
Can you index BIT type columns?	No	Yes, but the NonStop ODBC Server maps the BIT data type to the NonStop SQL/MP SMALLINT data type.
Can you index TEXT type columns?	No	Yes, but the NonStop ODBC Server maps the TEXT data type to the NonStop SQL/MP VARCHAR data type.
Can you create a clustered index?	Yes	No
Should index names be unique?	Yes, within a table, but not within a database.	Yes, the fully expanded name must be unique in the network.

NonStop ODBC Server Compared With NonStop SQL/MP

The corresponding NonStop SQL/MP statement is CREATE INDEX.

When used through the NonStop ODBC Server, CREATE INDEX differs from the NonStop SQL/MP implementation in the following ways:

Feature	In NonStop SQL/MP	In Programs Used With the NonStop ODBC Server
Can you specify descending keys?	Yes	No, unless you use pass-through mode.
Can you create partitioned indexes?	Yes	No, unless you use pass-through mode.
Can you specify partition attributes?	Yes	No, NonStop SQL/MP defaults are taken.
Can you specify file attributes?	Yes	No, NonStop SQL/MP defaults are taken.
Can you specify the physical location of the index?	Yes	No, unless you use pass-through mode. You can create an index only on the subvolume that holds the catalog in which the index will be registered.
Can you create a clustered index?	Yes	No

Specifying the Physical Location

In NonStop SQL/MP, you can specify the physical location of the index. In programs used with the NonStop ODBC Server, you cannot; you can create an index only on the subvolume that holds the catalog in which the index will be registered. You can, however, specify the physical location by using the pass-through mode.

The following statements create three indexes on the PERSNL subvolume on the disk volume DISK01 on the node TEST:

```
use test_disk01_persnl

create unique index xdeptnum
   on dept (deptnum)

create index xdeptmgr
   on dept (manager)

create index xdeptrpt
   on dept (rptdept)
```

The indexes are registered in the catalog on that subvolume.

CREATE TABLE

Use CREATE TABLE to create new tables.

The file security for the table is obtained from the default permissions associated with the user's logon name.

The CREATE TABLE statement has the following syntax:

```
CREATE TABLE [  [database.]owner. ]  table-name

  ( column-name data-type  [ NOT NULL ]
                           [ NULL     ]

  [ , column-name data-type [ NOT NULL ]  ] ...  )
                            [ NULL     ]
```

table-name

is the table to be created. The *table-name* can be qualified with the database name and owner name. The fully expanded table name must be unique in the network.

If the *owner-name* is omitted, the current user's name is used.

You can specify a temporary table by preceding the table name with a number sign (#). Temporary tables are dropped at the end of the NonStop ODBC Server session.

`column-name`

identifies a column in the table. The *`column-name`* must be unique for columns defined in the table.

`data-type`

specifies the SQL Server data type for the column. Data types are converted to NonStop SQL/MP data types. Data types are described under Language Elements on page 4-5.

If you specify an unsupported data type, an error message is generated and the statement is not executed.

`NOT NULL, NULL`

specifies whether the column can contain null values.

NOT NULL is the default.

Examples

The following statement creates a table called EMPLOYEE in the NonStop SQL/MP catalog called PERSNL:

```
use test_disk01_persnl

create table employee
   (empnum        smallint,
    first_name    char(15),
    last_name     char(20),
    deptnum       smallint  null,
    jobcode       smallint  null,
    salary        float     null)
```

The EMPLOYEE table definition is recorded in the PERSNL NonStop SQL/MP catalog on the disk volume DISK01 on the node called TEST.

The following statement creates a table called DEPT:

```
create table dept
   (deptnum       smallint,
    deptname      char(12),
    manager       smallint,
    rptdept       smallint   null,
    location      char (18)  null)
```

The table is created on the PERSNL subvolume and is recorded in the NonStop SQL/MP catalog on that subvolume.

NonStop ODBC Server Compared With SQL Server

When used through the NonStop ODBC Server, CREATE TABLE differs from the SQL Server implementation in the following ways:

Feature	In SQL Server	In Programs Used With the NonStop ODBC Server
Can CREATE TABLE be inside a transaction?	No	Yes See DDL Statements Allowed in User-Defined Transactions on page 4-3.
Is auditing of the volume necessary?	N.A.	Yes, the volume on which the table is created must be audited by TMF, even if the table itself is not audited.
Is the created table audited?	N.A.	Yes, all tables created using the NonStop ODBC Server are audited by TMF.
Are user-defined data types allowed?	Yes	No, NonStop SQL/MP does not support user-defined data types.
How much data can the table hold?	Depends on the space allocated for the database in the CREATE DATABASE or DISK INIT statements.	A limited amount set by system defaults. For information, see the CREATE TABLE statement in the *NonStop SQL/MP Reference Manual*. To create a table that holds more data, use pass-through mode.
Should table names be unique?	Yes, for a specific database and owner.	Yes, the fully expanded name must be unique in the network. The NonStop ODBC Server preserves uniqueness when the user specifies and uses a logical name.
Can you name another user as the owner of the new table?	No	Yes, if the specified owner name is a valid Guardian logon name.

File Attributes and Security Considerations

The following NonStop SQL/MP file attributes and security considerations apply to the use of CREATE TABLE:

- The default physical file attributes for the table are the same as in NonStop SQL/MP.
- The default table organization is the same as in NonStop SQL/MP: KEY SEQUENCED.
- The table definition must comply with NonStop SQL/MP limits. See the *NonStop SQL/MP Reference Manual* for the limits.

- The ownership and security of the table affect dependent indexes and views. These dependencies are discussed in the CREATE VIEW, CREATE INDEX, and ALTER security statements.

For further information on these NonStop SQL/MP features, see the CREATE TABLE statement description in the *NonStop SQL/MP Reference Manual*.

NonStop ODBC Server Compared With NonStop SQL/MP

The corresponding NonStop SQL/MP statement is CREATE TABLE.

When used through the NonStop ODBC Server, CREATE TABLE differs from the NonStop SQL/MP implementation in the following ways:

Feature	**In NonStop SQL/MP**	**In Programs Used With the NonStop ODBC Server**
Can you specify the physical location of a table?	Yes	No, a table created using the NonStop ODBC Server is created in the same location as the NonStop SQL/MP catalog in which it is registered, unless you use the pass-through mode.
Default column attribute	NULL	NOT NULL
Can you specify default values for columns?	Yes	No
Can you create partitioned tables?	Yes	No
Can you create nonaudited tables?	Yes	No, all tables created through the NonStop ODBC Server are audited. To create a nonaudited table, use pass-through mode.
Can you create a table with a primary key?	Yes	No, unless you use pass-through mode.
Data types available	All NonStop SQL/MP data types	Some NonStop SQL/MP data types. For information on the data types, see Language Elements on page 4-5.
Can you specify table attributes?	Yes	No, unless you use pass-through mode. Tables created through the NonStop ODBC Server use default NonStop SQL/MP table attributes.

CREATE VIEW

Use CREATE VIEW to create a view.

The file security for the view is obtained from the default permissions associated with the user’s logon name.

The CREATE VIEW statement has the following syntax:

```
CREATE VIEW view-name

  [ ( column-name [, column-name ] ... ) ]

  AS query-expression

query-expression is:

  { query-expression UNION [ALL] query-expression }
  { ( query-expression )                          }
  { query-specification                            }
```

view-name

specifies the name of the view to create.

You can specify the owner of a view, but you cannot specify the database or qualify the object name (view name). You can create a view for the current database only.

column-name

specifies an alternate column name for a column of the view. The *column-name* must be unique for columns defined in the view.

If you omit the *column-name* list, the column names of the view take the names from the columns in the select list of the SELECT statement.

query-expression

defines the columns for the view and sets the selection criteria. Unless you declare alternate column names in the *column-name* clause, the column names you specify in the query expression define the column names of the view.

A query expression consists of a single *query-specification* or a union of multiple *query specification*s.

query-specification

is a SELECT statement that defines the columns for the view and sets the selection criteria. Unless you declare alternate column names in the *column-identifier* list, the column names you specify in the *query-specification* define the column names of the view.

Tables in the *`query-specification`* must be in the current database or be qualified by their database name.

If there is only one table referenced by the view, NonStop ODBC automatically creates a protection view.

The *`query-specification`* cannot include the ORDER BY clause, but it can include the OUTER JOIN and UNION clauses.

See SELECT on page 3-54 for further information.

Examples

The following statement creates a view called MGRLIST on the disk volume DISK01 on the node TEST:

```
use test_disk01_persnl

create view mgrlist
  (first_name,
   last_name,
   department)
   as select first_name,
             last_name,
             deptname
   from dept,
     employee
     where dept.manager = employee.empnum
```

This view includes columns from the DEPT table and the EMPLOYEE table of the PERSNL database.

The MGRLIST view definition will be recorded in the PERSNL NonStop SQL/MP catalog on the disk volume DISK01 on the node called TEST.

NonStop ODBC Server Compared With SQL Server

When used through the NonStop ODBC Server, CREATE VIEW differs from the SQL Server implementation in the following ways:

Feature	**In SQL Server**	**In Programs Used With the NonStop ODBC Server**
Can CREATE VIEW be inside a transaction?	No	Yes See DDL Statements Allowed in User-Defined Transactions on page 4-3.
Can you modify data in a view?	Yes, with some restrictions.	Yes, if the view is a protection view. See the following discussion of “Shorthand Views and Protection Views.”
Can you select columns from several databases?	Yes	Yes, as long as the NonStop SQL/MP catalogs of all the databases are customized.

Feature	In SQL Server	In Programs Used With the NonStop ODBC Server
Is the view definition text visible?	Yes, in the SYSCOMMENTS system table.	Yes, but not in the SYSCOMMENTS catalog. The text is visible in the NonStop SQL/MP VIEWS catalog table if the user can determine the NonStop SQL/MP name used for the view. See Locating Objects Using the Mapping Tables on page 7-4.
Is auditing of the volume necessary?	N.A.	Yes, the volume on which the view is created must be audited by TMF.
Should view names be unique?	Yes, for a specific database and owner.	Yes, the fully expanded name must be unique in the network.
Can you name another user as the owner of the new view?	No	Yes, if the specified owner name is a valid Guardian logon name.

Shorthand Views and Protection Views

NonStop SQL/MP has two types of views:

Shorthand	Derived from one or more tables or other views and defined without the protection attribute. A shorthand view can be read but not updated.
Protection	Derived from a single table by taking a projection of the columns of the table, or a selection of the rows of the table, or both. For more information about the rules for protection views, see CREATE VIEW in the *NonStop SQL/MP Reference Manual*. A protection view can be secured, updated, and read.

When you create a view using the NonStop ODBC Server, it attempts to create a protection view. If the create fails, however, the NonStop ODBC Server creates a shorthand view. No messages are generated indicating the type of view created.

You will know that the NonStop ODBC Server created a shorthand view if you get an error message when you try to insert or update data in the view.

The view definition text indicates whether a view is a shorthand or a protection view. You can list this text by querying the NonStop SQL/MP VIEWS table, which resides on the subvolume of your NonStop SQL/MP catalog. The following SQLCI statement lists the view definition text for a view:

```
select viewtext from views where viewname =
   "\TEST.$VOL2.PERSNL.EMPLIST";
```

In this example, both the VIEWS table and the EMPLIST view are in the subvolume \TEST.$VOL2.PERSNL.

NonStop ODBC Server Compared With NonStop SQL/MP

The corresponding NonStop SQL/MP statement is CREATE VIEW.

When used through the NonStop ODBC Server, CREATE VIEW differs from the NonStop SQL/MP implementation in the following ways:

Feature	**In NonStop SQL/MP**	**In Programs Used With the NonStop ODBC Server**
Where is the view created?	In the user's current node, volume, and subvolume	On the same subvolume as the catalog in which the view is registered
Can you specify protection view or shorthand view?	Yes	No, unless you use pass-through mode.
Is WITH CHECK OPTION available?	Yes	No

DECLARE

Use DECLARE to declare the name and type of local variables for a batch. Assign values to local variables with SELECT statements. You must use variables in the batch in which they are declared.

Variables are described under Language Elements on page 4-5.

The DECLARE statement has the following syntax:

```
DECLARE @variable-name data-type

[ , @variable-name data-type ] ...
```

@variable-name

is the variable to declare. The *variable-name* must be preceded by a commercial "at" sign (@).

data-type

is the SQL Server data type for the variable. Data types are treated as declared in the NonStop ODBC Server; intermediate calculated results are performed with NonStop SQL/MP data types, which can lead to slightly different arithmetic truncation.   Data types are described under Language Elements on page 4-5.

If you specify an unsupported data type, an error message is generated and the statement is not executed.

Examples

The following statements declare variables and assign them values.

```
declare @veryhigh money
select @veryhigh = max(price) from titles
```

NonStop ODBC Server Compared With SQL Server

When used through the NonStop ODBC Server, DECLARE differs from the SQL Server implementation in the following way:

Feature	**In SQL Server**	**In Programs Used With the NonStop ODBC Server**
Can you reference a variable when its value is null?	Yes	No

Referencing Null Variables

When you declare a variable, its value is NULL. You must assign a value with a SELECT statement before referencing the variable.

For example, the following statements declare the variables @v1 and @v2, then assign the value of variable @v1 (which is null) to the variable @v2, causing an error:

```
declare @v1 int, @v2 int
select @v2 = @v1
```

If, however, you assign a value to @v1, you can assign its value to @v2. The following statements do not cause an error:

```
declare @v1 int, @v2 int
select @v1 = 55
select @v2 = @v1
```

Variables are described under Language Elements on page 4-5.

NonStop ODBC Server Compared With NonStop SQL/MP

NonStop SQL/MP does not have a statement that corresponds to DECLARE.

DELETE

Use DELETE to remove rows from a table.

The DELETE statement has the following syntax:

```
DELETE [ FROM ] table-reference

[ FROM table-reference [ , table-reference ...  ] ]

[ WHERE search-condition ]

table-reference is:

   [ [database.]owner. ]  {table-name }
                           {view-name) }
```

FROM

is not supported. The NonStop ODBC Server ignores this clause.

table-name or *view-name*

identifies the table or view from which rows are to be deleted. The *table-name* or *view-name* can be qualified with the database name and owner name.

You cannot delete rows from a view that references more than one table, even if the rows being deleted belong to only one of the tables.

FROM *table-reference*

is not supported. In the NonStop ODBC Server, this clause generates an error message. You cannot delete rows based on data stored in other tables.

WHERE *search-condition*

specifies the criteria for the rows to delete.

The *search-condition* cannot contain subqueries that refer to the table or view from which the rows are being deleted.

If you omit the WHERE clause, all rows are deleted.

Examples

The following statements delete one record from the EMPLOYEE table and one from the DEPARTMENT table:

```
delete employee
   where last_name = "Hall" and first_name = "Dana"

delete department
   where deptnum = 1030
```

NonStop ODBC Server Compared With SQL Server

When used through the NonStop ODBC Server, DELETE differs from the SQL Server implementation in the following ways:

Feature	In SQL Server	In Programs Used With the NonStop ODBC Server
Is the FROM clause available?	Yes	No. If you use it, the NonStop ODBC Server generates an error message. See the following discussion of the FROM clause.
Locking mode used	Exclusive	The NonStop SQL/MP REPEATABLE ACCESS mode, which is functionally similar to the SQL Server default of exclusive mode.

FROM Clause

In programs used with the NonStop ODBC Server, you cannot specify the SQL Server FROM clause with the DELETE statement. You can, however, sometimes achieve the same results by using a subquery.

The following DELETE statement is used with the NonStop ODBC Server. It deletes employees that work for manager number 1001. The statement uses a subquery to select all departments for which the manager is number 1001:

```
delete test_disk01_persnl..employee
where deptnum in
   (select deptnum from dept where manager = 1001)
```

In a program used with SQL Server, you could execute the previous statement by using a FROM clause:

```
delete employee
from dept
where employee.deptnum = dept.deptnum and dept.manager = 1001
```

NonStop ODBC Server Compared With NonStop SQL/MP

The corresponding NonStop SQL/MP statement is DELETE.

When used through the NonStop ODBC Server, DELETE differs from the NonStop SQL/MP implementation in the following ways:

Feature	In NonStop SQL/MP	In Programs Used With the NonStop ODBC Server
Can you specify the locking mode?	Yes	No, unless you use pass-through mode.
Default locking mode	STABLE ACCESS	REPEATABLE ACCESS

DROP DATABASE

Use DROP DATABASE to remove one or more NonStop SQL/MP catalogs.

DROP DATABASE removes the NonStop ODBC Server mapping tables and the NonStop SQL/MP catalog.

The DROP DATABASE statement has the following syntax:

```
DROP DATABASE database-name [ , database-name ] ...
```

database-name

is the subvolume name that identifies the database.

The *database-name* is a subvolume optionally qualified with a node and volume. If you omit the node and volume, the *database-name* is expanded using the user's current node and volume.

Note. If you execute a DROP DATABASE statement that names the database you are currently using, you must execute a USE statement specifying that database before attempting any SQL operations. Otherwise, you might receive unexpected results.

Examples

The following statement drops the database PERSNL:

```
drop database persnl
```

The database is in the user's current node and volume. For example, if the user's current node and volume is \RELY.$DISK01, the catalog PERSNL must be in \RELY.$DISK01.PERSNL.

The following statement drops three databases, PERSNL, SALES, and INVENT:

```
drop database persnl, sales, invent
```

All three of these databases must be in the user's current node and volume.

How the NonStop ODBC Server Drops Database Objects

When an object is registered in a customized catalog, the NonStop ODBC Server mapping tables keep track of whether the object was created using the NonStop ODBC Server or using NonStop SQL/MP.

When you drop a database using the NonStop ODBC Server, the NonStop SQL/MP catalog is dropped only if all objects in it were created through the NonStop ODBC Server. When you drop a mapped table, view, or index, the underlying NonStop SQL/MP object is also dropped and its mapping table removed.

DROP DATABASE fails if the mapping tables contain any NonStop ODBC Server objects that cannot be dropped.

DROP DATABASE is equivalent to USERCAT DEINSTALL with the CASCADE parameter (described under USERCAT DEINSTALL Statement on page 7-24).

Table 4-17 summarizes the objects, mapping tables, and catalogs that are dropped by DROP DATABASE.

Table 4-17. Dropping Mapping Tables, Objects, and Catalogs

	Which Items Are Dropped			
If the Database Contains	**NonStop ODBC Server Mapping Tables**	**Objects Created Using the NonStop ODBC Server**	**NonStop SQL/MP Catalog**	**Objects Created Using NonStop SQL/MP and Refreshed for the NonStop ODBC Server**
Only objects created using the NonStop ODBC Server	x	x	x	–
Objects created using the NonStop ODBC Server and objects created using NonStop SQL/MP	x	x	–	–

x Indicates the item is dropped
– Indicates the item is not dropped

NonStop ODBC Server Compared With SQL Server

When used through the NonStop ODBC Server, DROP DATABASE differs from the SQL Server implementation in the following ways:

Feature	**In SQL Server**	**In Programs Used With the NonStop ODBC Server**
Can DROP DATABASE be inside a user-defined transaction?	No	Yes See DDL Statements Allowed in User-Defined Transactions on page 4-3.

NonStop ODBC Server Compared With NonStop SQL/MP

The corresponding NonStop SQL/MP statement is DROP CATALOG.

The NonStop SQL/MP catalog is dropped only if all objects registered in it were created through the NonStop ODBC Server. See How the NonStop ODBC Server Drops Database Objects on page 4-65.

DROP INDEX

Use DROP INDEX to remove an index from the database.

The DROP INDEX statement has the following syntax:

```
DROP INDEX table-name.index-name

[ , table-name.index-name ] ...
```

table-name

identifies the table in which the indexed column is located. The table must be in the current catalog—you cannot drop an index from a table in another catalog.

index-name

specifies the index to drop.

If the statement contains errors, the results can be different than if executed using SQL Server. See Dropping Multiple Indexes.

Examples

The following statement drops three indexes from the EMPLOYEE table:

```
use test_disk01_persnl

drop index employee.xdeptnum,
   employee.xempname,
   employee.xempdept
```

The EMPLOYEE table is registered in the NonStop SQL/MP catalog that is on the PERSNL subvolume on the disk volume DISK01 on the node TEST.

Dropping Multiple Indexes

If you drop multiple indexes with one DROP INDEX statement and the statement contains errors, the results may be different than if the statement were executed using SQL Server.

Because NonStop SQL/MP does not allow you to drop multiple indexes with one statement, the NonStop ODBC Server translates the DROP INDEX statement to multiple NonStop SQL/MP DROP INDEX statements. Therefore, even if one of the statements contains an error, the other statements will be executed and will drop the specified indexes. In SQL Server, if one of the *index-name* clauses contains an error, the statement fails and none of the indexes are dropped.

NonStop ODBC Server Compared With SQL Server

When used through the NonStop ODBC Server, DROP INDEX differs from the SQL Server implementation in the following ways:

Feature	In SQL Server	In Programs Used With the NonStop ODBC Server
Can DROP INDEX be inside a transaction?	No	Yes See DDL Statements Allowed in User-Defined Transactions on page 4-3.
Which objects must be accessible?	N.A.	These associated objects: ● The underlying table ● All object program files using the underlying table (if any) ● Catalogs containing the description of the index
Should partitions be available?	N.A. SQL Server does not have partitioned indexes.	Yes, if the index is a partitioned index.
Security restrictions	SQL Server security restrictions apply.	NonStop SQL/MP security restrictions apply for the NonStop SQL/MP process accessor ID and for catalogs that describe indexes affected by the drop.

NonStop ODBC Server Compared With NonStop SQL/MP

The corresponding NonStop SQL/MP statement is DROP INDEX.

When used through the NonStop ODBC Server, DROP INDEX differs from the NonStop SQL/MP implementation in the following ways:

Feature	In NonStop SQL/MP	In Programs Used With the NonStop ODBC Server
Number of indexes you can drop with the DROP INDEX statement	One	Multiple
How you qualify the index name	Specify the Guardian file name that identifies the index.	Specify the table name that the index is associated with, followed by the index name.

DROP TABLE

Use DROP TABLE to remove a table from a database, along with all data, indexes, and dependent views for that table.

The DROP TABLE statement has the following syntax:

```
DROP TABLE  [ [database.]owner.]  table-name
              database..       ]

  [  ,  [ [database.]owner. ]  table-name  ] ...
  [     [ database..        ]              ]
```

table-name

identifies the table to be dropped; `table-name` can be qualified with the database name and owner name.

If the statement contains errors, the results can be different than if executed using SQL Server. See Dropping Multiple Tables.

Examples

The following statement drops three tables from the PERSNL database:

```
use test_disk01_persnl
```

```
drop table employee, job, dept
```

The tables are registered in the NonStop SQL/MP catalog that is on the PERSNL subvolume on the disk volume DISK01 on the node TEST.

Dropping Multiple Tables

If you drop multiple tables with one DROP TABLE statement and the statement contains errors, the results may be different than if the statement were executed using SQL Server.

Because NonStop SQL/MP does not allow you to drop multiple tables with one statement, the NonStop ODBC Server translates the DROP TABLE statement to multiple NonStop SQL/MP DROP TABLE statements. Therefore, even if one of the statements contains an error, the other statements will be executed and will drop the specified tables. In SQL Server, if one of the `table-name` clauses contains an error, the statement fails and none of the tables are dropped.

Dropping Dependent Views and Indexes

You should drop dependent views and indexes before dropping a table using the NonStop ODBC Server.

When you drop a table using the NonStop ODBC Server, NonStop SQL/MP drops the table and all dependent views and indexes. The NonStop ODBC Server, however, does not clear the mapping table entries for the views or indexes.

Note. When the mapping table entries are not cleared and you later attempt to create a view or index by the same name, you will receive error messages saying that the object already exists when in fact it does not.

The only way to clear the mapping table entries in this situation is to run the catalog utility statement REFRESH.

For information on REFRESH, see Maintaining Catalogs on page 7-27.

Dropping Partitioned Tables

NonStop SQL/MP allows creation of partitioned tables. Each partition has an entry in the mapping table. Dropping a table that is partitioned causes all the partitions to be dropped, but the mapping entries will remain. To clear the entries, run the catalog utility statement REFRESH.

For information on REFRESH, see Maintaining Catalogs on page 7-27.

NonStop ODBC Server Compared With SQL Server

When used through the NonStop ODBC Server, DROP TABLE differs from the SQL Server implementation in the following ways:

Feature	**In SQL Server**	**In Programs Used With the NonStop ODBC Server**
Can DROP TABLE be inside a transaction?	No	Yes, if the table is audited.
Must partitions be available?	N.A.	Yes, and although all partitions are dropped, mapping entries for them are not. See Dropping Partitioned Tables.
Are dependent programs automatically invalidated?	N.A.	Yes

Feature	In SQL Server	In Programs Used With the NonStop ODBC Server
Are constraints automatically dropped?	SQL Server does not have constraints, but any rules on the table lose their binding, and any triggers associated with the table are automatically dropped.	Yes
Are dependent views and indexes automatically dropped?	No	Yes, but the mapping entries for the dependent objects are not dropped. See Dropping Dependent Views and Indexes on page 4-69.
Security requirements imposed	SQL Server security requirements	NonStop SQL/MP security requirements. See the information on DROP TABLE in the *NonStop SQL/MP Reference Manual.*

NonStop ODBC Server Compared With NonStop SQL/MP

The corresponding NonStop SQL/MP statement is DROP TABLE.

When used through the NonStop ODBC Server, DROP TABLE differs from the NonStop SQL/MP implementation in the following way:

Feature	In NonStop SQL/MP	In Programs Used With the NonStop ODBC Server
Number of tables you can drop with a DROP TABLE statement	One	Multiple

DROP VIEW

Use DROP VIEW to remove views from the database.

The DROP VIEW statement has the following syntax:

```
DROP VIEW [ owner.]view-name

[ , [ owner.]view-name ] ...
```

view-name

identifies the view to drop; *view-name* can be qualified with the owner name.

Examples

The following statement drops a view owned by PAYROLL_MGR and one owned by PAYROLL_ADMIN:

```
use test_disk01_persnl

drop view payroll_mgr.emplist,
          payroll_admin.mgrlist
```

Dropping Dependent Views and Indexes

You should drop dependent views and indexes before dropping a view using the NonStop ODBC Server.

When you drop a view using the NonStop ODBC Server, NonStop SQL/MP drops the view and all dependent views and indexes. The NonStop ODBC Server, however, does not clear the mapping table entries for the views or indexes.

Note. When the mapping table entries are not cleared and you later attempt to create a view or index by the same name, you will receive error messages saying that the object already exists when in fact it does not.

The only way to clear the mapping table entries in this situation is to run the catalog utility statement REFRESH.

For information on REFRESH, see Maintaining Catalogs on page 7-27.

NonStop ODBC Server Compared With SQL Server

When used through the NonStop ODBC Server, DROP VIEW differs from the SQL Server implementation in the following ways:

Feature	In SQL Server	In Programs Used With the NonStop ODBC Server
Can DROP VIEW be inside a transaction?	No	Yes See DDL Statements Allowed in User-Defined Transactions on page 4-3.

Feature	In SQL Server	In Programs Used With the NonStop ODBC Server
Are dependent views and indexes dropped?	No	Yes, but the mapping entries for the dependent objects are not dropped. See Dropping Dependent Views and Indexes on page 4-69.
Format of an owner name	SQL Server identifier	A Guardian logon name. For information on specifying owner names, see Language Elements on page 4-5.
Security requirements imposed	SQL Server security requirements	NonStop SQL/MP security requirements; anyone with the proper file-system security can drop a view. For information, see the description of the DROP VIEW statement in the *NonStop SQL/MP Reference Manual.*

NonStop ODBC Server Compared With NonStop SQL/MP

The corresponding NonStop SQL/MP statement is DROP VIEW.

When used through the NonStop ODBC Server, DROP VIEW differs from the NonStop SQL/MP implementation in the following way:

Feature	In NonStop SQL/MP	In Programs Used With the NonStop ODBC Server
Number of views you can drop with a DROP VIEW statement	One	Multiple

EXECUTE

Use EXECUTE to invoke a previously defined stored procedure (a Pathway server class program).

The EXECUTE statement has the following syntax:

```
[ EXEC[UTE] ] [ @return-status = ]

   [server.]  [ [database.]owner.   ]  procedure-name[;number]
              [ database..          ]

   [ [@parameter-name =]  { value,                  } [, ...]
]
                          { @variable [ OUT[PUT] ] }

   [ WITH RECOMPILE ]
```

@return-status

is a local variable that returns a code indicating the completion status of the stored procedure execution. The completion codes are defined by the Pathway programmer. The data type of *@return-status* is as declared in the batch prior to the EXECUTE command. The data type must be of type INTEGER or SMALLINT. The default data type is SMALLINT.

If the procedure returns a value, the batch where the stored procedure is called should include a *@return-status* local variable declared as numeric. Note that a data conversion error might occur if the local variable declared is not exactly of type INTEGER. The reply to the NonStop ODBC Server from the Serverclass_Send message indicates the return variable's value.

The EXEC[UTE] keyword may be omitted if the statement is the first one in a batch.

```
[server.]  [ [database.]owner. ]  procedure-name[;number]
           [ database..        ]
```

identifies the stored procedure. If the *server* name is included, the NonStop ODBC Server ignores it; the NonStop ODBC Server does not support four-part names. The *database*, *owner*, and *procedure-name* portions of the name are described under Names on page 4-5. A procedure name may optionally begin with an underscore (_) character.

The *number* that can optionally be appended to the procedure name identifies a specific procedure within a group of procedures having the same name; if omitted, it defaults to 1. It is recommended that *number* not be used as part of the procedure name if the procedure is intended to be shared by both Transact-SQL and CORE SQL users.

`@parameter-name`

is a Transact-SQL identifier. According to SQL Server rules, parameter values without associated parameter names must be given in the order defined in the procedure declaration. If *`@parameter-name`* is used, any sequence is allowed, but when this form is used, it must be used for all subsequent parameters in the EXECUTE statement.

`value`

is either a literal or a local variable. No other expression is permitted.

`@variable` `[ OUT[PUT] ]`

is a local variable of the procedure, defined in the same batch in which the EXECUTE call is issued; it is a Transact-SQL identifier. If the keyword OUTPUT is specified, the parameter is an output variable that receives a value when the procedure is executed. By default, all parameters are considered to be input parameters unless the keyword OUTPUT (or OUT) is specified.

`WITH RECOMPILE`

indicates that a less-than-optimal access plan is acceptable; it has no meaning on the HP server and is ignored (the NonStop ODBC Server always uses the optimal plan from among those available).

Considerations

- Named and unnamed parameters can be mixed in a procedure call, but if a named parameter is used, all subsequent parameters in the statement must also be named or an error occurs.
- If a procedure call supplies more values than the number of parameters defined in the procedure declaration, the excess values are ignored. No error or warning message is issued.
- If values are assigned to the same named parameter more than once, only the first value is used. No error or warning message is issued.
- If a default value is specified for a parameter of a procedure (using the NonStop ODBC Server catalog utility ADD PROCEDURE), you can execute a stored procedure without giving a parameter value; however, the missing values must be at the end of the parameter list. If a default value is not specified, an error message is issued for each missing parameter. For example, the DEBIT_ACCOUNT stored procedure can be called without any parameters, provided a default value is defined for each of its parameters:

  ```
  execute debit_account
  ```
- An error occurs if a parameter is used as an OUTPUT parameter and its corresponding definition in the procedure declaration is not INPUT/OUTPUT.

- OUTPUT variables are not part of a transaction, so if a parameter is changed in a transaction that is rolled back, the parameter's value does not revert to its previous value.
- If the data type of a parameter and its corresponding parameter definition in the procedure declaration are not compatible, a data conversion error occurs. If data conversion results in data truncation, a warning is issued. It is the responsibility of the Pathway server class program to take the appropriate action.

Example

The following statement invokes a stored procedure named DEBIT_ACCOUNT:

```
exec debit_account (@name = "smith", @amount = 50.00)
```

The statement passes the name of the account to access and the amount to be debited.

NonStop ODBC Server Compared With SQL Server

When used through the NonStop ODBC Server, EXECUTE operates in the same way as it does with SQL Server.

NonStop ODBC Server Compared With NonStop SQL/MP

NonStop SQL/MP has no statement corresponding to EXECUTE.

Stored procedure execution is described in detail in Execution of Stored Procedures on page 5-4.

INSERT

Use INSERT to add new rows to a table or a view.

The INSERT statement has the following syntax:

```
INSERT [ INTO ]  [ [database.]owner. ]  { table-name }
                                        { view-name) }

[ ( column-name [, column-name] ... ) ]

  { VALUES ( constant-expr [, constant-expr ] ... )  }
  {    select-statement                                }
```

table-name

identifies the table into which the rows are inserted; *table-name* can be qualified with the database name and owner name.

view-name

identifies the view into which the rows are inserted; *view-name* can be qualified with the database name and owner name.

The view must be an updatable NonStop SQL/MP protection view. For information on protection views, see “Shorthand Views and Protection Views” in the CREATE VIEW on page 4-58.

column-name

specifies a column for which a value will be supplied.

constant-expr

is a value (or the keyword NULL) for a column in the row being inserted. This list of values must match the *column-name* list.

select-statement

is a SELECT statement that selects values from other tables and views to be inserted in the table or view specified in the INTO clause.

The select list must contain an element for each column that you specify in the *column-name* list.

The *select-statement* cannot refer to the table, view, or underlying table of the view into which the rows are being inserted.

Examples

The following example adds one row to the EMPLOYEE table:

```
insert into test_disk01_persnl..employee
   (empnum, first_name, last_name, deptnum, jobcode, salary)
   values (1234, "Georgia", "Brown", 1001, 3004, 52300)
```

NonStop ODBC Server Compared With SQL Server

When used through the NonStop ODBC Server, INSERT differs from the SQL Server implementation in the following ways:

Feature	In SQL Server	In Programs Used With the NonStop ODBC Server
Can you insert rows into a view that references more than one table?	Yes, if the columns being inserted belong to only one table.	No, even if the columns being inserted belong to only one table.
Locking mode used	Exclusive	The NonStop SQL/MP REPEATABLE ACCESS mode, which is functionally similar to the SQL Server default of exclusive mode.
Rules the SELECT statement must follow	SQL Server rules	These NonStop SQL/MP rules: The *`select-list`* must contain an element for each column that you specify in the *`column-name`* list. A subquery cannot refer to a table, view, or underlying table of the view into which rows are being inserted.

NonStop ODBC Server Compared With NonStop SQL/MP

The corresponding NonStop SQL/MP statement is INSERT.

When used through the NonStop ODBC Server, INSERT differs from the NonStop SQL/MP implementation in the following ways:

Feature	In NonStop SQL/MP	In Programs Used With the NonStop ODBC Server
Can you specify the locking mode?	Yes	No, unless you use pass-through mode.
Default locking mode	STABLE ACCESS	REPEATABLE ACCESS

PRINT

Use PRINT to return a user-defined message to the user’s application.

The PRINT statement has the following syntax:

```
PRINT  { "ascii-string"    }
       { @local-variable   }
       { @@global-variable }
```

"ascii-string"

is a character string, up to 255 characters, enclosed in double quotation marks (").

local-variable

is the name of a local variable. The variable must be of type CHAR or VARCHAR and must be declared within the batch in which it is used.

global-variable

is the name of a global variable. The variable must be of type CHAR or VARCHAR or automatically convertible to these types (such as @@version).

Examples

The following statement prints an ASCII string:

```
print "This is everyone in the EMPLOYEE table"

select * from employee
```

NonStop ODBC Server Compared With SQL Server

When used through the NonStop ODBC Server, PRINT does not differ from the SQL Server implementation.

NonStop ODBC Server Compared With NonStop SQL/MP

NonStop SQL/MP does not have a statement that corresponds to PRINT.

ROLLBACK TRANSACTION

Use ROLLBACK TRANSACTION to roll back all changes to the database that occurred since the last BEGIN TRANSACTION statement was executed.

The ROLLBACK TRANSACTION statement has the following syntax:

```
ROLLBACK TRAN[SACTION]  [transaction-name ]
                        [savepoint-name   ]
```

transaction-name

is the name associated with the transaction.

In SQL Server, transaction names are needed when transactions are nested; however, because transactions cannot be nested in programs used with the NonStop ODBC Server, the *transaction-name* is not meaningful in these programs.

When used through the NonStop ODBC Server, if the *transaction-name* does not match the *transaction-name* on the previous BEGIN TRANSACTION statement, an error message is issued and the transaction is not rolled back.

If *transaction-name* is not specified, the transaction in progress is rolled back.

savepoint-name

is not supported. The NonStop ODBC Server does not distinguish between a *transaction-name* and a *savepoint-name*.

Example

The following transaction creates a table and inserts a row but rolls back before the transaction is completed:

```
use test_disk01_persnl

begin transaction

create table employee
   (empnum        smallint,
    first_name    char(15),
    last_name     char(20),
    deptnum       smallint  null,
    jobcode       smallint  null,
    salary        float     null)

insert into test_disk01_persnl..employee
   (empnum, first_name, last_name, deptnum, jobcode, salary)
   values (1234, "Georgia", "Brown", 1001, 3004, 52300)

insert into test_disk01_persnl..employee
   (empnum, first_name, last_name, deptnum, jobcode, salary)
   values (1234, "George", "Blue", 1002, 3002, 33500)

rollback transaction
```

NonStop ODBC Server Compared With SQL Server

When used through the NonStop ODBC Server, ROLLBACK TRANSACTION differs from the SQL Server implementation in the following ways:

Feature	**In SQL Server**	**In Programs Used With the NonStop ODBC Server**
Are nested transactions allowed?	Yes	No
Transaction savepoints allowed?	Yes	Yes, but they are ignored. If you want to roll back a transaction, you must roll back the entire transaction.
Are DDL operations on a table allowed within a user-defined transaction?	No	Yes, if the table is audited. See DDL Statements Allowed in User-Defined Transactions on page 4-3.

NonStop ODBC Server Compared With NonStop SQL/MP

The corresponding NonStop SQL/MP statement is ROLLBACK WORK.

The primary difference between NonStop ODBC Server transactions and NonStop SQL/MP transactions is that, in programs used with the NonStop ODBC Server, you can specify a transaction name. The transaction name is provided primarily for compatibility with SQL Server and is not meaningful in NonStop SQL/MP (transactions cannot be nested in NonStop SQL/MP).

SAVE TRANSACTION

Use SAVE TRANSACTION to set a transaction savepoint. If used in the NonStop ODBC Server, the SAVE TRANSACTION statement is ignored. No error or warning messages are generated.

The SAVE TRANSACTION statement has the following syntax:

```
SAVE TRAN[SACTION] [ savepoint-name ]
```

savepoint-name

is not supported. This parameter is ignored.

Examples

The following transaction contains several savepoints:

```
use test_disk01_persnl

begin transaction
```

```
create table employee
   (empnum        smallint,
    first_name    char(15),
    last_name     char(20),
    deptnum       smallint  null,
    jobcode       smallint  null,
    salary        float     null)

save transaction create_tab

insert into test_disk01_persnl..employee
   (empnum, first_name, last_name, deptnum, jobcode, salary)
   values (1234, "Georgia", "Brown", 1001, 3004, 52300)

save transaction insert_emp1

insert into test_disk01_persnl..employee
   (empnum, first_name, last_name, deptnum, jobcode, salary)
   values (1234, "George", "Blue", 1002, 3002, 33500)

rollback transaction
```

When the ROLLBACK TRANSACTION statement is encountered, the entire transaction is rolled back.

SELECT

Use SELECT to retrieve rows from database tables.

The SELECT statement has the following syntax:

```
SELECT  { ALL      }  select-list
        { DISTINCT }

  [ INTO  [ [database.]owner. ]  table-name) ]

FROM table-reference [ correlation-name ] [ HOLDLOCK ]

[, table-reference [ correlation-name ] [ HOLDLOCK ] ] ...

[ WHERE search-condition ]

[ GROUP BY [ ALL ] agg-free-expr [ , agg-free-expr ] ...  ]

[ HAVING search-condition ]

[ ORDER BY sort-spec [ , sort-spec ] ...  ]

[ COMPUTE comp-list BY column-name [ , column-name ] ...  ]

[ FOR BROWSE ]

select-list is:

select-item [ , select-item ] ...

select-item is:

  { [table-reference.]*                              }
  { [table-reference.]column-name                    }
  { column-heading = [table-reference.]column-name }
  { [table-reference.]column-name column-heading   }
  { expression                                       }
  { aggregate-function                               }

agg-free-expr is:

  { column-name }
  { expression  }

sort-spec is:

  {  [table-reference.]column-name  }  [ ASC  ]
  {  select-list-number             }  [ DESC ]
  {  expression                      }
```

```
comp-list is:

  aggregate(column-name) [ , aggregate(column-name) ] ...

table-reference is:

  [ [database.]owner. ]  { table-name }
                         { view-name  }
```

Examples

The following statement retrieves all rows and all columns from the EMPLOYEE table:

```
select * from employee
```

The following statement retrieves all rows from the EMPLOYEE table, and specifies alternate column headings to be displayed:

```
select
   enum = empnum,
   first_name,
   last_name,
   dept = deptnum,
   salary
   from employee
```

The following statement retrieves all employees in department 1030:

```
select * from employee
   where deptnum = 1030
```

The preceding syntax diagram shows the entire syntax of the SELECT statement. The following syntax diagrams show the syntax of each primary clause.

ALL/DISTINCT Clause

The ALL/DISTINCT clause of the SELECT statement specifies the columns to be included in the result of the query. The clause has the following syntax:

```
{ ALL      }  select-list
{ DISTINCT }

select-list is:

  select-item [ , select-item ] ...

  select-item is:

  { [table-reference.]*                             }
  { [table-reference.]column-name                   }
  { column-heading = [ table-reference.]column-name }
  { [table-reference.]column-name column-heading    }
  { expression                                      }
  { aggregate-function                              }

table-reference is:

[ [database.]owner.]  { table-name }
                       { view-name  }
```

`ALL or DISTINCT`

specifies whether all rows or only distinct rows are to be included in the result of the query:

ALL	Include all rows of the result table
DISTINCT	Include only unique rows of the result table. Duplicate rows are removed from the result table.

ALL is the default.

`[ `*`table-reference`*`.]*`

retrieves all columns of a table or view. An asterisk (*) specifies all columns, in CREATE TABLE order.

`[ `*`table-reference`*`.]`*`column-name`*

retrieves the specified columns in the order specified.

`column-heading`` = [ `*`table-reference`*`.]`*`column-name`*

retrieves the specified column and replaces the heading of the retrieved column with the specified *`column-heading`*. The *`column-heading`* is an SQL Server identifier. In SQL Server, these column headings are also called column aliases.

`[ table-reference.]column-name column-heading`

retrieves the specified column and replaces the heading of the retrieved column with the specified *column-heading*.

expression

is an expression. Expressions are described under Language Elements on page 4-5.

aggregate-function

is one of the following aggregate functions:

AVG
COUNT
MAX
MIN
SUM

Aggregates are described under Language Elements on page 4-5.

INTO Clause

The INTO clause of the SELECT statement is not supported in the NonStop ODBC Server and, if included, causes an error message. The clause has the following syntax:

```
[ INTO  [ [database.]owner. ]  table-name ]
```

table-name

is not supported.

FROM Clause

The FROM clause of the SELECT statement specifies the tables and views used in the SELECT statement and has the following syntax:

```
FROM table-reference [ correlation-name ] [ HOLDLOCK ]

[, table-reference [ correlation-name ] [ HOLDLOCK ] ] ...


table-reference is:

[ [database.]owner.]  { table-name }
                      { view-name  }
```

table-reference

is a table or view used in the SELECT statement. The *table-name* or *view-name* can be qualified with the database name and owner name.

`correlation-name`

is a name that identifies the table or view for the SELECT statement. Each *`correlation-name`* in the FROM clause must be unique.

In SQL Server, a correlation name is called an alias.

`HOLDLOCK`

in SQL Server, this makes a shared lock on a specified table or view more restrictive by holding it until the completion of a transaction.

In the NonStop ODBC Server, the HOLDLOCK keyword determines the locking mode for the SELECT statement as follows:

HOLDLOCK present?	**Locking Mode Used**
Yes	The NonStop SQL/MP REPEATABLE ACCESS mode, which retains locks on all rows that are read, keeping the data locked once it is accessed. Locks are released when the transaction is committed or aborted.
No	The NonStop SQL/MP STABLE ACCESS, which locks data for a short duration; data is locked and remains locked only while it is being processed.

WHERE Clause

The WHERE clause of the SELECT statement specifies a search condition to be applied to each row of the FROM clause result table and has the following syntax:

```
WHERE search-condition
```

Search conditions are described under Language Elements on page 4-5.

GROUP BY Clause

The GROUP BY clause of the SELECT statement specifies the groups into which the table will be divided and has the following syntax:

```
GROUP BY [ ALL ] agg-free-expr [ , agg-free-expr ] ...


agg-free-expr is:

  { column-name }
  { expression  }
```

ALL

is not supported. In the NonStop ODBC Server, ALL is ignored.

column-name

is the name of a column from one of the tables in the FROM clause.

expression

is not supported. In the NonStop ODBC Server, expressions in the *aggregate-free-expression* cause an error message.

HAVING Clause

The HAVING clause of the SELECT statement sets conditions for the GROUP BY clause and has the following syntax:

```
HAVING search-condition
```

Search conditions are described under Language Elements on page 4-5.

ORDER BY Clause

The ORDER BY clause of the SELECT statement specifies the order in which to sort the rows of the result table and has the following syntax:

```
ORDER BY sort-spec [ , sort-spec ] ...


sort-spec is:

  { [table-reference.]column-name }  [ ASC  ]
  {                               }  [ DESC ]
  { select-list-number            }
  {                                }
  { expression                    }

table-reference is:

[ [database.]owner. ]  { table-name }
                       { view-name  }
```

table-reference.column-name

is the name of a column from one of the tables or views in the FROM clause. The column name can be qualified with the database name and owner name.

select-list-number

is a number representing the position of the item in the select list. The *select-list* can be an asterisk (*).

expression

is not supported. An error message is generated.

`ASC or DESC`

specifies whether to sort in ascending or descending order; the default is ascending.

COMPUTE BY Clause

The COMPUTE BY clause of the SELECT statement is not supported in the NonStop ODBC Server. It has the following syntax:

```
COMPUTE comp-list BY column-name [ , column-name ] ...


comp-list is:

aggregate(column-name) [, aggregate(column-name) ] ...
```

Inclusion of the COMPUTE BY clause causes an error message to be generated.

FOR BROWSE Clause

The FOR BROWSE clause of the SELECT statement is not supported in the NonStop ODBC Server. It has the following syntax:

```
FOR BROWSE
```

The FOR BROWSE clause is ignored, and the SELECT statement is executed without browse access. No messages or warnings are generated.

NonStop ODBC Server Compared With SQL Server

When used through the NonStop ODBC Server, SELECT differs from the SQL Server implementation in the following ways:

Feature	In SQL Server	In Programs Used With the NonStop ODBC Server
Is the INTO clause available?	Yes	No, but an alternative is available. See INTO Clause on page 4-90.
Is SQL Server BROWSE mode available?	Yes	No
Is the COMPUTE BY clause available?	Yes	No
Locking mode used	Shared, unless you specify the HOLDLOCK keyword—then the shared lock is more restrictive.	NonStop SQL/MP STABLE ACCESS, unless you specify the HOLDLOCK keyword—then REPEATABLE ACCESS is used.

Feature	In SQL Server	In Programs Used With the NonStop ODBC Server
Are vector aggregates available?	Yes	No
Can nonaggregate and aggregate expressions be mixed in the *select-list* ?	Yes	No
Can the *select-list* be an asterisk (*)?	No	Yes

INTO Clause

In programs used with the NonStop ODBC Server, you cannot specify the SQL Server INTO clause with the SELECT statement. You can, however, sometimes achieve the same results by using an INSERT statement with a subquery.

For example, the following statements are used with the NonStop ODBC Server. They insert a new table, MY_DIVISION, and obtain data for the new table from an existing table, EMPLOYEE:

```
create table my_division
   (empnum         smallint
    first_name     char(15),
    last_name      char(20),
    jobcode        smallint,
    deptnum        smallint,
    salary         float        null)

insert into my_division
   (empnum, first_name, last_name, jobcode, deptnum, salary)
    select empnum, first_name, last_name, jobcode,
    deptnum, salary
      from employee
      where jobcode > 500 and deptnum <= 3000
```

In a program used with SQL Server, you could achieve the results of the previous statement by using an INTO clause in a SELECT statement:

```
select jobcode, deptnum, first_name, last_name, salary
   into my_division
   from persnl.employee
   where jobcode > 500 and deptnum <= 3000
```

NonStop ODBC Server Compared With NonStop SQL/MP

The corresponding NonStop SQL/MP statement is SELECT.

When used through the NonStop ODBC Server, SELECT differs from the NonStop SQL/MP implementation in the following ways:

Feature	In NonStop SQL/MP	In Programs Used With the NonStop ODBC Server
How to specify the locking mode	Use the BROWSE/REPEATABLE/ STABLE ACCESS clause.	Use the HOLDLOCK keyword; you cannot use the BROWSE/REPEATABLE/STABLE ACCESS clause.
Is BROWSE ACCESS locking mode available?	Yes	No, unless you use pass-through mode.
Is the INTO clause available?	Yes	No, unless you use pass-through mode.
Is the IN EXCLUSIVE MODE clause available?	Yes	No, unless you use pass-through mode. See IN EXCLUSIVE MODE Clause.
Is the UNION clause available?	Yes	No, unless you use pass-through mode.

IN EXCLUSIVE MODE Clause

In programs used with the NonStop ODBC Server, you cannot specify the NonStop SQL/MP IN EXCLUSIVE MODE clause; you can, however, obtain exclusive mode by locking the table using pass-through mode as shown in the following example:

```
begin transaction

use test_disk01_persnl

select "tdm: sql lock table \test.$disk01.persnl.employee
   in exclusive mode"

select jobcode, deptnum, first_name, last_name
   from employee
   where jobcode > 500 and deptnum <= 3000"

select "tdm: sql unlock table \test.$disk01.persnl.employee"

commit transaction
```

Remember to unlock the table when you no longer need the lock on it, as illustrated in the preceding example.

In a program used with NonStop SQL/MP, you would obtain exclusive mode by specifying the IN EXCLUSIVE MODE clause:

```
select jobcode, deptnum, first_name, last_name
   from persnl.employee
   where jobcode > 500 and deptnum <= 3000
   in exclusive mode;
```

SET

Use SET to set query-processing options for the duration of a work session.

The SET statement has the following syntax:

```
SET  { { ALTNAMES                      } { OFF } }
     { { ARITHABORT                    } { ON  } }
     { { ARITHIGNORE                   }         }
     { { BACKGROUND                    }         }
     { { CONTROL                       }         }
     { { NOCOUNT                       }         }
     { { NOEXEC                        }         }
     { { OFFSETS { { keyword-list } }          }
     { { PARSEONLY                     }         }
     { { PROCID                        }         }
     { { SHOWPLAN                      }         }
     { { STATISTICS IO                 }         }
     { { STATISTICS TIME               }         }
     { ROWCOUNT number                           }
     { TEXTSIZE number,                           }
```

ALTNAMES

is not supported. If included in a program used with the NonStop ODBC Server, an error message is generated.

ARITHABORT

is not supported. In the NonStop ODBC Server, ARITHABORT is always ON and cannot be reset. If included in a program used with the NonStop ODBC Server, an error message is generated.

ARITHIGNORE

is not supported. If included in a program used with the NonStop ODBC Server, an error message is generated.

BACKGROUND

is not supported. If included in a program used with the NonStop ODBC Server, an error message is generated.

CONTROL

is not supported. If included in a program used with the NonStop ODBC Server, an error message is generated.

NOCOUNT

is not supported. If included in a program used with the NonStop ODBC Server, an error message is generated.

`NOEXEC`

is not supported. If included in a program used with the NonStop ODBC Server, an error message is generated.

`OFFSETS {` *`keyword-list`* `}`

is not supported. If included in a program used with the NonStop ODBC Server, an error message is generated.

`PARSEONLY`

parses subsequent statements but does not execute them (except for other SET statements):

ON	Statements are parsed but not executed.
OFF	Statements are executed as they are received.

PARSEONLY affects all statements, including all pass-through statements.

The default is PARSEONLY OFF.

`PROCID`

is not supported. If included in a program used with the NonStop ODBC Server, an error message is generated.

`SHOWPLAN`

is not supported. If included in a program used with the NonStop ODBC Server, an error message is generated.

`STATISTICS IO`

is not supported. If included in a program used with the NonStop ODBC Server, an error message is generated.

`STATISTICS TIME`

is not supported. If included in a program used with the NonStop ODBC Server, an error message is generated.

`ROWCOUNT` *`number`*

is not supported. If included in a program used with the NonStop ODBC Server, an error message is generated.

`TEXTSIZE` *`number`*

is the size, in bytes, of TEXT type data to be returned with a SELECT statement. The *`number`* is a number between 0 and 4050.

If you specify 0, the text size is unchanged. If you do not explicitly set TEXTSIZE, the default is 512 bytes.

Examples

The following batch sets PARSEONLY to ON and executes several INSERT statements. The INSERT statements are not executed.

```
set parseonly on

insert into employee
   (empnum, first_name, last_name, deptnum,
    jobcode, salary)
    values (2001, "Dana", "Hall", 1010, 3001, 40000)

insert into employee
   (empnum, first_name, last_name, deptnum,
    jobcode, salary)
    values (2003, "Jeff", "Brown", 1030, 3001, 35000)
```

The following statements set TEXTSIZE to 100, then create a table with a TEXT column. Because TEXTSIZE is set to 100, the text column will be 100 bytes wide:

```
set textsize 100

create table abstracts
   (book_id int,
    book_title char (25),
    description text)
```

NonStop ODBC Server Compared With SQL Server

When used through the NonStop ODBC Server, SET differs from the SQL Server implementation in the following ways:

Feature	**In SQL Server**	**In Programs Used With the NonStop ODBC Server**
Parameters available	All	PARSEONLY and TEXTSIZE
Default TEXTSIZE	4096 bytes	512 bytes
Effect of setting TEXTSIZE to 0	TEXTSIZE is set to the default (4096 bytes).	TEXTSIZE is unchanged.

NonStop ODBC Server Compared With NonStop SQL/MP

NonStop SQL/MP does not have a statement that corresponds to SET.

TRUNCATE TABLE

Use TRUNCATE TABLE to remove all rows in a table.

The TRUNCATE TABLE statement has the following syntax:

```
TRUNCATE TABLE  [ [database.]owner. ]  table-name
```

table-name

identifies the table to be truncated. The table name can be qualified with the database name and owner name.

Examples

The following statement removes all rows from the EMPLOYEE table:

```
use test_disk01_persnl

truncate table employee
```

NonStop ODBC Server Compared With SQL Server

When used through the NonStop ODBC Server, TRUNCATE TABLE differs from the SQL Server implementation in the following way:

Feature	In SQL Server	In Programs Used With the NonStop ODBC Server
Are changes written to the audit log?	No	Yes. NonStop SQL/MP does not have a TRUNCATE TABLE statement, so a DELETE statement is executed instead.

NonStop ODBC Server Compared With NonStop SQL/MP

The corresponding NonStop SQL/MP statement is DELETE.

Executing the TRUNCATE TABLE statement is the same as executing the following DELETE statement:

```
DELETE table-name
```

For example, both of the following statements used with the NonStop ODBC Server remove all rows from the EMPLOYEE table:

```
truncate table employee

delete employee
```

UPDATE

Use UPDATE to change data in existing rows, either by adding new data or by modifying existing data.

The UPDATE statement has the following syntax:

```
UPDATE table-reference

  SET [ table-reference.]column-name =  { expression }
                                        { NULL       }

  [ column-name =  { expression } ]  ...
  [                { NULL       } ]

[ FROM table-reference [ , table-reference ] ...  ]

[ WHERE search-condition ]

table-reference is:

[ [database.]owner.]  { table-name }
                      { view-name  }
```

table-reference

identifies the table or view to be updated. The *table-name* or *view-name* can be qualified with the database name and owner name.

column-name

specifies a column to be updated. The first *column-name* can be qualified with the database name, owner name, and table or view name.

expression

is a valid expression. Expressions are described under Language Elements on page 4-5.

NULL

sets the value to NULL.

FROM *table-reference*

is not supported. In the NonStop ODBC Server, this clause generates an error message.

WHERE *search-condition*

specifies criteria for selecting the rows to be updated. The *search-condition* cannot have a subquery that references the table or view in which the rows are being updated.

Examples

The following example updates the salary of employee number 1002:

```
update test_disk01_persnl..employee
   set salary = salary * 1.1
   where empnum = 1002
```

NonStop ODBC Server Compared With SQL Server

When used through the NonStop ODBC Server, UPDATE differs from the SQL Server implementation in the following ways:

Feature	**In SQL Server**	**In Programs Used With the NonStop ODBC Server**
FROM clause available?	Yes	No; if you use it, the NonStop ODBC Server generates an error message. See FROM Clause on page 4-64.
Locking mode used	Exclusive	The NonStop SQL/MP REPEATABLE ACCESS mode, which is functionally similar to the SQL Server default of exclusive mode.
Are aggregate functions allowed in the *expression?*	Yes	No

FROM Clause

In programs used with the NonStop ODBC Server, you cannot specify the SQL Server FROM clause with the UPDATE statement. You can, however, sometimes achieve the same results by using a subquery.

The following UPDATE statement is used with the NonStop ODBC Server. It updates the salaries of all employees working for the manager whose ID number is 1001. The statement uses a subquery to select all departments for manager is number 1001:

```
update employee
set salary = salary * 1.1
where deptnum in
   (select deptnum from dept where manager = 1001)
```

In SQL Server, you could execute the previous statement by using a FROM clause:

```
update persnl.employee
set salary = salary * 1.1
from dept
where employee.deptnum = dept.deptnum
   and dept.manager = 1001
```

NonStop ODBC Server Compared With NonStop SQL/MP

The corresponding NonStop SQL/MP statement is UPDATE.

When used through the NonStop ODBC Server, UPDATE differs from the NonStop SQL/MP implementation in the following ways:

Feature	**In NonStop SQL/MP**	**In Programs Used With the NonStop ODBC Server**
Can you specify the locking mode?	Yes	No, unless you use pass-through mode.
Default locking mode	STABLE ACCESS	REPEATABLE ACCESS

UPDATE STATISTICS

Use UPDATE STATISTICS to update all statistics associated with all columns of a specified table.

The UPDATE STATISTICS statement has the following syntax:

```
UPDATE STATISTICS

 [ [database.]owner. ]  table-name [ index-name ]
```

table-name

identifies the table for which statistics are to be updated; *table-name* can be qualified with the database name and owner name.

index-name

is not supported. If included in a program used with the NonStop ODBC Server, the *index-name* is ignored.

Examples

The following statement updates statistics on the EMPLOYEE table:

```
use test_disk01_persnl

update statistics
   employee
```

NonStop ODBC Server Compared With SQL Server

When used through the NonStop ODBC Server, UPDATE STATISTICS differs from the SQL Server implementation in the following way:

Feature	In SQL Server	In Programs Used With the NonStop ODBC Server
Can you specify the index for which to update statistics?	Yes	No

NonStop ODBC Server Compared With NonStop SQL/MP

The corresponding NonStop SQL/MP statement is UPDATE STATISTICS.

USE

Use the USE statement to set the default NonStop SQL/MP catalog. The new catalog takes effect as soon as the statement is executed.

When you start a NonStop ODBC Server session, the default database in a user profile is used as the default database. MASTER is used if a default database does not exist. You can define the default database in a user profile; see the description of the ADD USER statement under ADD USER on page 7-115.

The USE statement has the following syntax:

```
USE database-name
```

database-name

identifies the customized NonStop SQL/MP catalog to use.

If you specify MASTER, as shown in the following example, the NonStop ODBC Server points to the system catalog, which is a NonStop SQL/MP catalog that corresponds to the SQL Server MASTER database:

```
use master
```

If you specify a database name, as shown in the following example, the NonStop ODBC Server points to the NonStop SQL/MP catalog \TEST.$DISK01.PERSNL:

```
use test_disk01_persnl
```

Examples

The following statement defines the catalog to be used as the catalog PERSNL on the disk volume DISK01 on the node called TEST:

```
use test_disk01_persnl
```

NonStop ODBC Server Compared With SQL Server

When used through the NonStop ODBC Server, USE differs from the SQL Server implementation in the following way:

Feature	In SQL Server	In Programs Used With the NonStop ODBC Server
When does the new database take effect?	When the current batch finishes.	Immediately (even before the batch finishes).

NonStop ODBC Server Compared With NonStop SQL/MP

The corresponding NonStop SQL/MP statements are CATALOG and DEFINE.

Specifying a catalog in the NonStop ODBC Server differs from specifying a catalog in NonStop SQL/MP in the following ways:

Feature	In NonStop SQL/MP	In Programs Used With the NonStop ODBC Server
Format for a catalog name	\node.$volume.subvolume	node_volume_subvolume
How the catalog name is expanded if you omit the node and volume	Uses the node and volume where the user is currently logged on.	Expanded using the current volume and subvolume

5 Stored Procedures

This section describes the HP NonStop ODBC Server support for execution of stored procedures by an ODBC application or a DB-LIBRARY application connected to a NonStop ODBC server on a HP server. The term “NonStop ODBC Server” is used to distinguish the server process itself from the overall NonStop ODBC Server product.

This section covers the following topics:

- An introduction to what stored procedures are and how they are used
- An overview of the NonStop ODBC Server implementation of stored procedures
- Executing stored procedures in either CORE SQL or Transact-SQL
- Development of stored procedures, including the use of the NonStop ODBC Server library of stored procedure execution functions (SPELIB)
- Installation of stored procedures using the NonStop ODBC Server catalog utilities and mapping tables
- Transaction management, security, sharing of stored procedures, and other such considerations

Introduction to Stored Procedures

A stored procedure is a mechanism to define and store a user program in a database system and invoke it at a later time. A stored procedure is a database object that can contain many elements of 3GL programming language procedures (parameters, flow control, local variables, and so on) in addition to SQL statements.

The ability to write your own stored procedures greatly enhances the power, efficiency, and flexibility of the SQL database language. Many vendors support stored procedures, but all have variations in the definition syntax and the execution semantics.

The following is an example of a stored procedure body and how it can be used:

1. A stored procedure body for a procedure called DEBIT_ACCOUNT:

```
int DEBIT_ACCOUNT ( char *name    /* Account name     */
                  , int   amount  /* Amount to adjust */
                  )
{
 if (!name) /* Name parameter is required */
   encode_print_message ("You must supply account name");
 else
   {
   exec sql update =ACCOUNT
     set BALANCE = BALANCE + setscale (:amount , 2)
     where ACCNT_NAME = :name;
   }
}  /* DEBIT_ACCOUNT */
```

2. Invocation of the stored procedure DEBIT_ACCOUNT:

   ```
   execute DEBIT_ACCOUNT ("John Smith", 50.00)
   ```

Overview of Stored Procedures

The NonStop ODBC Server allows the user to develop stored procedures in a familiar 3GL language, such as C or COBOL. Using a 3GL language provides the user with full access to all of the capabilities of the 3GL language, along with the performance and debugging features that come with this approach. In addition, the NonStop ODBC Server uses Pathway to manage the stored procedures, which provides for efficient load balancing and server class management.

Stored procedures can be used in two ways:

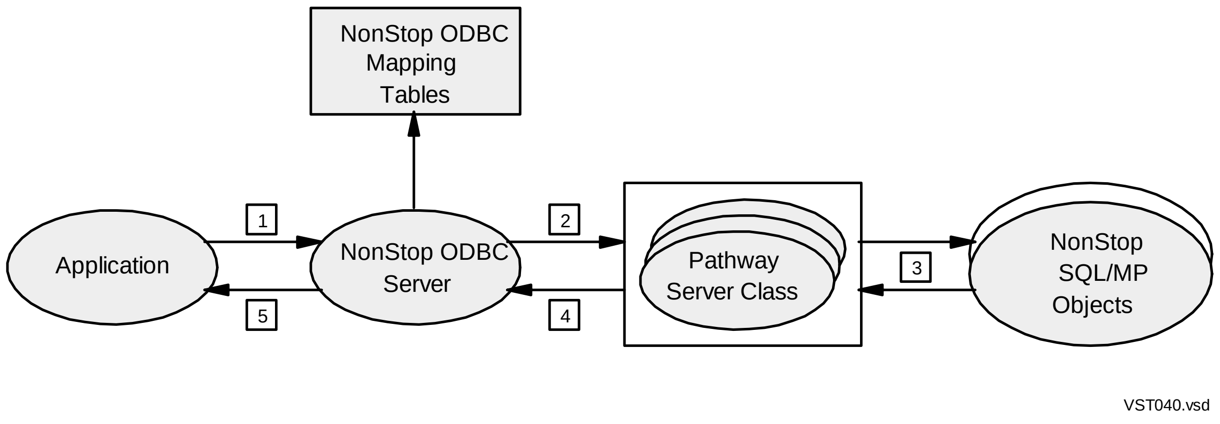
- One way involves “light” use: the procedures are created and used by the same user, they are modified often, performance is not critical, and they may have a short life cycle.
- The other way is “heavy” use: the procedures are created and changed infrequently, they are shared by several users, and their performance is important. NonStop ODBC Server support of stored procedure execution is most appropriate for this second use.

Stored procedures can be written in either the C programming language or COBOL85.

How Stored Procedure Execution Works

Figure 5-1 illustrates how the NonStop ODBC server accepts a stored procedure invocation statement, submits it to a Pathway server class program, and returns results to the client application.

Figure 5-1. Executing Stored Procedures



1. An application issues an ODBC or DBLIB call that sends a stored procedure invocation statement to the NonStop ODBC server.

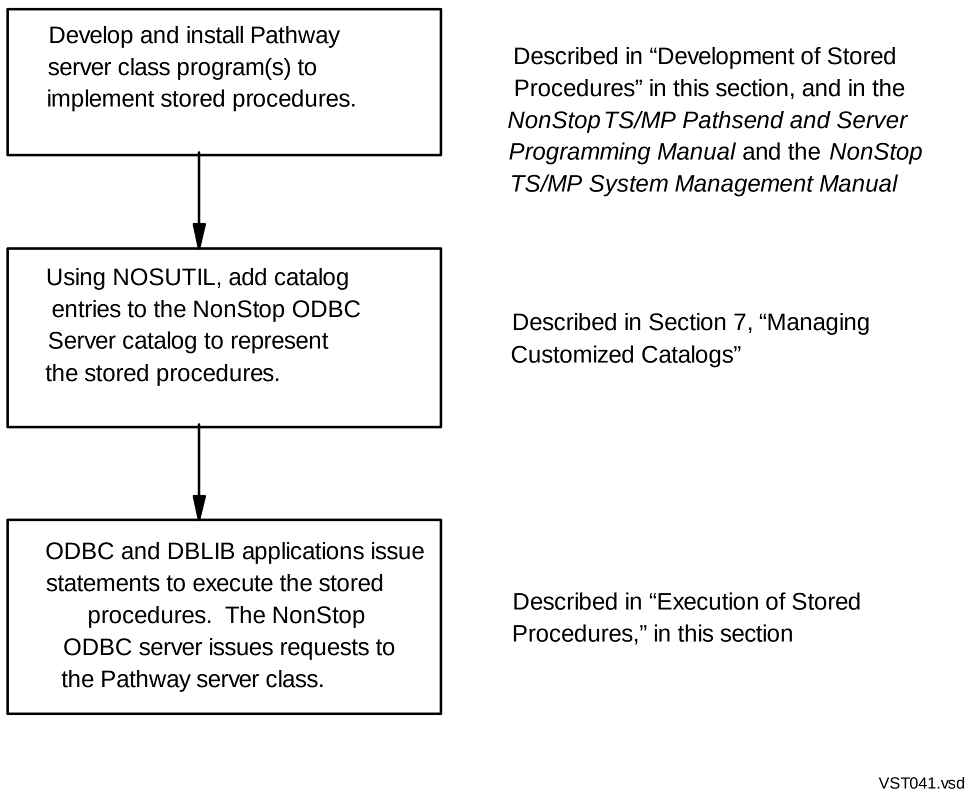
2. The NonStop ODBC server translates the stored procedure invocation to a Pathway server class request, using mapping tables in the NonStop ODBC Server catalogs to look up the Pathway server class program, and sends the request to the Pathway server class.
3. The code in the server class process carries out the stored procedure steps.
4. The server class program returns HP NonStop SQL/MP data (or a diagnostic message) to the NonStop ODBC server.
5. The NonStop ODBC server converts the NonStop SQL/MP output to ODBC or SQL Server format and sends the result to the application program.

The overall NonStop ODBC Server architecture, as well as the architecture’s support for stored procedures, is described in detail in Section 2, Architecture Overview.

Installing and Using Stored Procedures

Figure 5-2 Illustrates the steps involved in developing, installing, and executing stored procedures, and the location of additional information about those steps.

Figure 5-2. Sequence of Activities in Stored Procedure Usage


Develop and install Pathway server class program(s) to implement stored procedures.
Described in “Development of Stored Procedures” in this section, and in the *NonStop TS/MP Pathsend and Server Programming Manual* and the *NonStop TS/MP System Management Manual*
Using NOSUTIL, add catalog entries to the NonStop ODBC Server catalog to represent the stored procedures.
Described in Section 7, “Managing Customized Catalogs”
ODBC and DBLIB applications issue statements to execute the stored procedures. The NonStop ODBC server issues requests to the Pathway server class.
Described in “Execution of Stored Procedures,” in this section
VST041.vsd

Execution of Stored Procedures

Stored procedures can be invoked through the NonStop ODBC Server using either the CORE SQL syntax or the Transact-SQL syntax. The following conditions must be met for successful execution of a stored procedure:

- The stored procedure must have been created as a Pathway server (see Development of Stored Procedures on page 5-6).
- Information regarding the stored procedure must exist in the NonStop ODBC Server catalog tables

The stored procedure can have input and output parameters and a return status code. It can also return “result sets,” which are the output of SQL SELECT statements.

The following subsections describe stored procedure invocation from both CORE SQL and Transact-SQL.

Invoking Stored Procedures in CORE SQL

Stored procedure execution is an extension of ODBC that is supported by the NonStop ODBC Server. CORE SQL uses the CALL statement to invoke stored procedures. The following describes the syntax of the statement. The CALL statement, and the ODBC “escape clause” format that enables the use of extensions to CORE SQL, are described in greater detail in Section 3, CORE SQL Language.

```
--*( VENDOR(Microsoft),PRODUCT(ODBC)
   [ ?= ] CALL procedure-name [ ( parameter [, ... ] ) ] )*--
        or
{ [?=] CALL procedure-name [ ( parameter [, ... ] ) ] }
```

Note. In the escape clause format, the --*(...)*-- or { ... } enclosure is a required element.

Execution of the stored procedure results in the following:

- If the stored procedure contains SELECT statements, the caller receives a result set for every SELECT statement. A result set consists of a description of the columns in the result rows together with zero or more rows of values. UPDATE, INSERT, and DELETE statements return a count of affected rows.

  In the following example, the procedure GET_EMPNAME is called to obtain the names of employees whose IDs are between 1000 and 1010. The result set consists of columns LAST_NAME and FIRST_NAME, and three rows of data.

  ```
  { ? = CALL GET_EMPNAME (1000,1010) }
  ```

The result appears as follows:

```
FIRST_NAME    LAST_NAME
----------    ---------
JOE           SMITH
DON           HUE
DEBBIE        SMITH
```

- If the optional return status (?=) is specified and the stored procedure has been coded and configured to return a status code, the caller receives the execution status in addition to the results.

Stored procedure execution can return multiple result sets or counts. Errors regarding the stored procedure are obtained in the same way as for other SQL statements.

Invoking Stored Procedures in Transact-SQL

Transact-SQL uses the EXECUTE statement to invoke stored procedures. The following describes the syntax of the EXECUTE statement. The statement is described in greater detail in Section 4, Transact-SQL Language.

```
[ EXEC[UTE] ] [ @return-status = ]

   [ server.]  [ [database.]owner. ]  procedure-name[;number
]
               [ database..        ]

   [ [@parameter-name =]  { value                  }  [, ...]
]
                          { @variable [ OUT[PUT] ] }

   [ WITH RECOMPILE ]
```

Execution of the stored procedure results in the following:

- If the stored procedure contains SELECT statements, the caller receives a result set for every SELECT statement. A result set consists of a description of the columns in the result rows together with zero or more rows of values. UPDATE, INSERT, and DELETE statements return a count of affected rows.

  In the following example, the procedure GET_EMPNAME is called to obtain the names of employees whose IDs are between 1000 and 1010. The result set consists of columns LAST_NAME and FIRST_NAME, and three rows of data.

  ```
  @RETURN_STATUS = EXECUTE GET_EMPNAME
                   @LOW_EMP_ID =  1000,
                   @HIGH_EMP_ID = 1010
  ```

The result appears as follows:

```
FIRST_NAME    LAST_NAME
----------    ---------
JOE           SMITH
```

```
DON            HUE
DEBBIE         SMITH
```

- If the procedure returns a status code, the caller also receives the execution status. In the previous example, the execution status is returned in the variable @RETURN_STATUS.
- If the procedure is defined to have output parameters and the procedure execution requests output values, the caller receives those output values. In the following example, the stored procedure DEBIT_ACOUNT also returns the account balance in the variable BALANCE.

```
@RETURN_STATUS = DEBIT_ACCOUNT
                 @NAME="SMITH",
                 @AMOUNT=100,
                 @BALANCE OUT
```

For DBLIB users, the return status and output parameters are consumed within the current batch and are not automatically returned to the application.

The result of a stored procedure execution can be obtained as a batch of statements is processed. Errors regarding the stored procedure are obtained in the same way as for other SQL statements.

Configuration Options

The configuration options handled by the NonStop ODBC server for execution of SQL statements are as follows:

```
SQL_ACCESS_MODE
SQL_TXN_ISOLATION
SQL_CURSOR_MODE
SQL_MAX_ROWS
```

These configuration options do not apply to stored procedure execution, however.

Development of Stored Procedures

The following subsection provides guidelines for creating stored procedures, using as an example a set of procedures and files that are included with the NonStop ODBC Server product for instructional purposes.

Design Considerations

The NonStop ODBC Server support of stored procedures is implemented using Pathway server programs. A set of stored procedures can be implemented using one or more Pathway server classes.

The NonStop ODBC Server defines an interprocess communication (IPC) format and protocol (SPEIPC) between the NonStop ODBC server and the Pathway server. The NonStop ODBC Server provides a set of library functions (SPELIB) for the Pathway server program to decode the request and to encode parameters, result sets, and error messages. The SPELIB functions do not perform any actual I/O ($RECEIVE, OPEN, WRITE and REPLY handling) between the NonStop ODBC Server and the Pathway server class program. However, a sample Pathway server program has been provided with the NonStop ODBC Server product that includes a set of routines for handling $RECEIVE and a set of SPELIB shell routines for handling call-back processing. In addition, the source code for three sample stored procedures that access the NonStop SQL/MP sample database are released with this product.

When developing a stored procedure, you have two options (option 1 is the recommended approach):

1. Develop the stored procedure using the sample Pathway server program and the set of shell routines provided with the sample server program. These routines mirror the SPELIB functions, but are integrated into the sample Pathway server program and remove the need for you to do call-back processing and to perform I/O.
2. Develop the stored procedure using the SPELIB library functions. The stored procedure can then be embedded in the sample Pathway server program, or you can provide your own Pathway server code.

The differences between these two options are as follows:

- Call-back handling—the major difference is that the shell routines in the sample Pathway server program insulate you from having to deal with call-back processing. The routines automatically initiate a call-back if the IPC buffer becomes full. When using the SPELIB functions directly, you are responsible for handling this. However, by using the SPELIB functions directly, you have control over when to initiate call-back.
- Use of the sample server—when using the shell routines, you must use the sample server. When using the SPELIB functions directly, you can use your own server code. The sample server is intended as a starting point; you can then change it to suit your own purposes. (SPELIB functions and shell routines are listed at the end of this subsection)

Server Logic Sequence

The Pathway server program executing the stored procedures should perform the following functions (see the “Sample Stored Procedures” subsection for more details):

- Read $RECEIVE to receive the IPC message from the NO
- Decode the message by calling the sample shell routines or the SPELIB functions.
- Allocate memory for input parameters and output results.
- Call a stored procedure to perform work.
- Encode the reply message by calling the sample shell routines or the SPELIB functions.

- Send the reply or write the output buffer back by utilizing the call-back feature as needed.

The following diagram illustrates the calling sequences involved. The SP_SRV_*xxx* functions are explained in detail under SPELIB Interface on page 5-27. In addition to the SPELIB functions, a set of shell routines has been provided with the sample Pathway server program.

The following sequences apply to both the shell routines and the equivalent SPELIB functions.

```
NonStop ODBC Server                    Pathway Server

PATHSEND( buffer, messagelen );       1.  Do READUPDATE from $RECEIVE
      .                                   process_event_queue      [shell routines]
      .                                       .

/* NSODBC server now waits for */     2.  Decode message.
/* PATHSEND to terminate.      */         process_request          [shell routines]
      .                                       or
      .                                   SP_SRV_DECODE_INIT           [SPELIB]
      .                                       .

      .                               3.  Get stored procedure information.
      .                                   get_spe_info             [shell routines]
      .                                       or
      .                                   SP_SRV_GET_SERVICE_NAME      [SPELIB]
      .                                       .

      .                               4.  Get number of input parameters.
      .                               5.  Allocate space for SQLDA.
      .                               6.  Get input parameters into SQLDA.
      .                               7.  Allocate space for parameter values.
      .                                   get_input_params         [shell routines]
      .                                       or
      .                                   SP_SRV_GET_NUM_INPUT_PARAMS  [SPELIB]
      .                                   SP_SRV_GET_INPUT_PARAM_INFO
      .                                   SP_SRV_GET_INPUT_PARAMS
      .                                       .

      .                               8.  Execute SQL to satisfy SP request.
      .                                   /*  -- WAIT --  */
      .                                       .

      .                               9.  Encode REPLY.
      .                                   encode_init              [shell routines]
      .                                       or
      .                                   SP_SRV_ENCODE_INIT@@@        [SPELIB]
      .                                       .

      .                               10. Set up column definitions.
      .                                   encode_row_descriptor    [shell routines]
      .                                       or
      .                                   SP_SRV_ENCODE_ROW_DESCR      [SPELIB]
      .                                       .

      .                               11. Set up result set.
      .                                   encode_row_data          [shell routines]
      .                                       or
      .                                   SP_SRV_ENCODE_ROW_DATA       [SPELIB]
      .                                       .
      .                                   Repeat Step 11 as needed.
      .                                       .
```

```
NonStop ODBC Server                  Pathway Server
        .                            12. If any error/warning occurs:
        .                                encode_sql_diagnostic     [shell routines]
        .                                       or
        .                                SP_SRV_ENCODE_SQL_DIAGNOSTIC [SPELIB]
        .                                       .
        .                            13. End every SQL statement with:
        .                                encode_end_statement      [shell routines]
        .                                       or
        .                                SP_SRV_ENCODE_END_STATEMENT  [SPELIB]
        .                                       .
        .                            14. End procedure.
        .                                encode_end_proc           [shell routines]
        .                                       or
        .                                SP_SRV_ENCODE_END_PROC       [SPELIB]
        .                                       .
        .                            15. Reply.
        .                                    send_reply            [shell routines]
/* PATHSEND terminates. */                      .
                                     .free (senv);
```

Using the Sample Server

To assist in the development of stored procedures, the following files for a sample stored procedure server are included with the NonStop ODBC Server product. A set of three sample stored procedures that have been implemented using the sample server are also provided.

ADDSP	An OBEY file that contains sample NOSCOM ADD PROCEDURE commands for registering all of the sample stored procedures. Sample commands are included for all of the sample stored procedures.
BINDFILE	A sample file for binding the sample server and performing the SQL compilation step.
CODECOB	The working storage declarations referenced from the sample COBOL85 stored procedure for handling NonStop SQL/MP errors. CODECOB is distributed as part of the NonStop SQL/MP sample database.
CSAMPLEC	The main routine for the C-only version of the sample server.
CSAMPLE	The executable program for running the C-only sample server. This program includes the stored procedures SELECT_DEPT3000, SELECT_INPUT_DEPT, and UPDATE_SALARY.
CSHELLC	The $RECEIVE handling and SPELIB shell routines used by the C-only sample server program. Its object file, CSHELLO, is bound in CSAMPLE.
CSHELLO	The object file for CSHELLC.

DBCREATE	An OBEY file for creating the sample database (use this only if the sample database has not already been created).
DBLOAD	An OBEY file for populating the sample database.
DROPSP	An OBEY file that contains sample NOSCOM REMOVE PROCEDURE commands to drop the sample stored procedures.
PATHIN	An example of a Pathway configuration file. In addition to the Pathway server class EMP-DATA-DML for the C-only version of the sample server (CSAMPLE), it includes the server class EMP-DATA-MIX for use with the mixed-code sample server (SAMPLE).
SAMPLEC	The source code for the main routine of the mixed-code sample server.
SAMPLE	The executable program for the mixed-code sample server. This program includes the stored procedures DEPT9000 and SELECT_EMP. The DEPT9000 stored procedure is written in COBOL85 and SELECT_EMP is written in C. The sample server shows how stored procedures written in COBOL85 and C can be mixed.
SAMPLEO	The object file for the main routine of the mixed-code sample server.
SHELLC	The source code for the shell routines that can be used for developing stored procedures written in COBOL85 or C.
SHELLO	The object file for the shell routines.
SP20C	The C source code for the stored procedure SELECT_DEPT300, which selects employee data from the EMPLOYEE table for department 3000 and returns the employee number, first name, last name, and department number of every employee in department 3000 to the client. This procedure has no input parameters, and returns one result set.
SP21C	The C source code for the stored procedure SELECT_INPUT_DEPT, which selects employee data from the EMPLOYEE table for a specific department. The department number is a required input parameter. The procedure returns one result set that includes the employee number, first name, last name, and department number of qualified employees to the client.
SP22C	The C source code for the stored procedure UPDATE_SALARY, which updates a given employee's salary with a new salary amount. The employee number and new salary are required input parameters. The procedure returns the employee number, first name, last name, department, and new salary amount to the client. SP22C is the source file that contains this procedure.

SP23COB	The COBOL85 source code for the sample stored procedure DEPT9000. This stored procedure returns employee data for department 9000.
SP24C	The C source code for the sample stored procedure SELECT_EMP. This procedure returns employee data for the specified employee number. The employee number is a required input parameter.
SPMACRO	A file that contains TACL macros for the sample stored procedure server.

The Sample Pathway Server Program

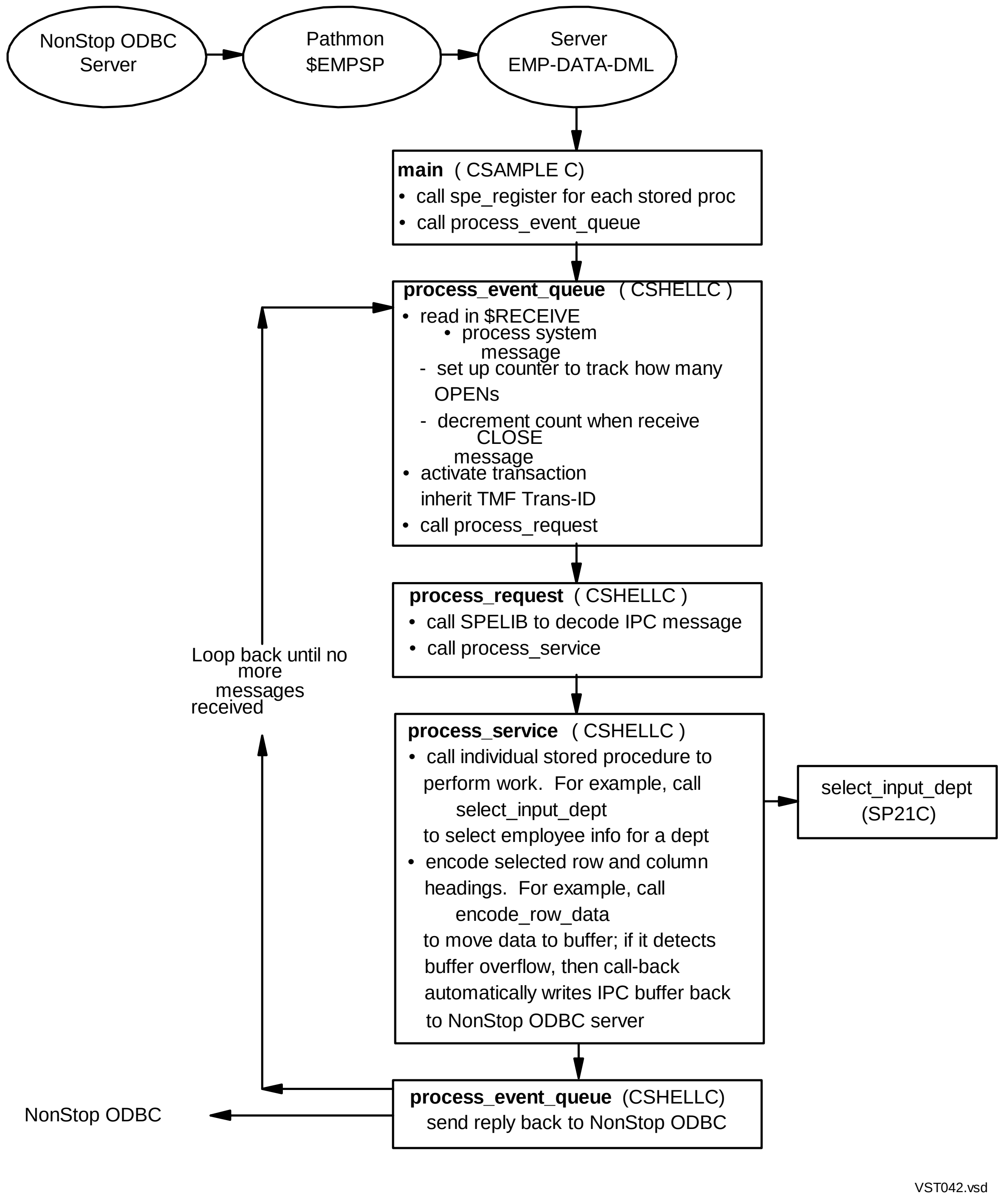
The following describes the logic processes of the sample Pathway server program. In this example, the following assumptions are made:

- The stored procedure name is SELECT_INPUT_DEPT.
- The Pathmon process name is $EMPSP.
- The Pathway server class name is EMP-DATA-DML.

Flow of the Sample Program

Figure 5-3 illustrates the control flow and sequence of events in the sample program.

Figure 5-3. Sample Program Flow


NonStop ODBC Server
Pathmon $EMPSP
Server EMP-DATA-DML
main (CSAMPLE C)
• call spe_register for each stored proc
• call process_event_queue
process_event_queue (CSHELLC)
• read in $RECEIVE
• process system message
- set up counter to track how many OPENs
- decrement count when receive CLOSE message
• activate transaction inherit TMF Trans-ID
• call process_request
process_request (CSHELLC)
• call SPELIB to decode IPC message
• call process_service
Loop back until no more messages received
process_service (CSHELLC)
• call individual stored procedure to perform work. For example, call select_input_dept to select employee info for a dept
• encode selected row and column headings. For example, call encode_row_data to move data to buffer; if it detects buffer overflow, then call-back automatically writes IPC buffer back to NonStop ODBC server
select_input_dept (SP21C)
process_event_queue (CSHELLC) send reply back to NonStop ODBC
NonStop ODBC
VST042.vsd

Each of the major functions in Figure 5-3 are described as follows:

`main`

The main() routine is very simple and uses a function named REGISTER_SPE for registering all of the stored procedures included within the server. The REGISTER_SPE function is called once for each stored procedure included within the server. The function PROCESS_EVENT_QUEUE is then called for handling any event received from $RECEIVE.

`register_spe`

This function is called to register a stored procedure within the server program. The registration process allows each stored procedure to provide the following:

- the address of a function to be called when the server program is first started
- the address of a function to be called when the stored procedure is to be executed
- the address of a function to be called when the server program terminates

`process_event_queue`

This is the main function of the stored procedure server. It is responsible for initializing the server by calling the initialization functions provided for each of the stored procedures. After initialization is completed, this function handles all system messages received on $RECEIVE, then calls the function PROCESS_REQUEST to handle any stored procedure request received from an NonStop ODBC Server. When the number of openers reaches zero, this function calls the termination functions provided for each of the stored procedures; it then stops the server program.

`process_request, process_service`

These functions are responsible for processing a stored procedure request from an NonStop ODBC Server and for invoking the relevant stored procedure function.

Developing a Stored Procedure in C

The basic programming paradigm used for stored procedures is the same as that used for developing NonStop SQL/MP applications. The system uses SQLDA structures for handling input parameters and for returning results to the client application. The sample stored procedures provided give examples of how the SQLDA structures are used. The remainder of this subsection describes the steps involved in the development and installation of a stored procedure using the sample files provided. The stored procedure SELECT_INPUT_DEPT and its source files SP21C and SP21H are used as examples.

The basic steps in developing a stored procedure are described as follows. A more detailed description of how to create the code for the stored procedure appears later in this subsection.

1. Create the stored procedure (SELECT_INPUT_DEPT):
   a. Provide a routine, if needed, for any initialization that should be performed for this stored procedure when the Pathway server is first started. For example, INITIALIZE_SPE_21.
   b. Write the code for the body of the stored procedure, such as SELECT_INPUT_DEPT.
   c. Provide a routine, if needed, for any clean-up that should be performed for this stored procedure when the Pathway server is stopped. For example, DISPOSE_SPE_21.
   d. Generate an “H” file for this stored procedure that includes declarations for the routines created in the previous steps. See SP21H for an example.
   e. Compile the stored procedure and associated functions created in the preceding steps.
2. Add the stored procedure to the sample program (CSAMPLEC):
   a. Include the header file generated for the stored procedure being added.
   b. Register the new stored procedure in the server program by calling the function SPE_REGISTER.
3. Build the new sample program (CSAMPLEC):
   a. Compile the sample program (to generate the object CSAMPLEO).
   b. Rebind the sample program, CSAMPLE, adding in the object file for the new stored procedure. See the file BINDFILE for a sample bind file.
   c. SQL compile the new sample program. Use the SPCOMP macro from the SPMACRO file for the compilation.
4. Set up a Pathway configuration for the new server. See the file PATHIN for a sample configuration. In this example, the server class is named EMP-DATA-DML.
5. Register the stored procedure with the NonStop ODBC server using the ADD PROCEDURE command. For example:

```
ADD PROCEDURE tess_data1_db.sql_odbc.select_input_dept
   PATHMON_NAME $empsp SERVERCLASS EMP-DATA-DML
   NUM_RESULT_SETS 1
   MAX_BUFFER_LEN 2000

ADD PROCEDURE_COLUMNS
   tess_data1_db.sql_odbc.select_input_dept
   p1 SMALLINT
```

The ADD PROCEDURE command registers the stored procedure SELECT_INPUT_DEPT for user SQL.ODBC in the NonStop ODBC Server catalog \TESS.$DATA1.DB, with the Pathmon process name of $EMPSP, server name of EMP-DATA-DML, one result set, and a maximum reply buffer length of 2000 bytes. The ADD PROCEDURE_COLUMNS command specifies one input parameter for the stored procedure.

6. Set up the sample database (if not already installed):

   Create the NonStop SQL/MP sample database by obeying the file DBCREATE and load the data by obeying the file DBLOAD. The catalog for the sample database is created in $*xxx*.PERSNL. The tables EMPLOYEE, DEPT, and so on, are created and registered in the same location.

7. Execute the Stored Procedure:

   a. Start the Pathway server; for example:

      ```
      Pathmon /name $empsp/
      Pathcom /in PATHIN/ $empsp
      ```

      To add a new stored procedure after the Pathmon process has been started, you need to freeze the server, stop the server, and start the server again.

   b. Send in a request from the client to execute the stored procedure.

Coding a Stored Procedure in C

This subsection covers Steps 1 and 2 from Developing a Stored Procedure in C on page 5-13 in greater detail, using the sample code for the stored procedure SELECT_INPUT_DEPT.

1. Create the initialization routine.

   This routine is called when the stored procedure is first started. This step is where the SQLDA structures used within the stored procedure are set up, along with any other data structures that need to be initialized.

   In the sample file SP21C, the routine INITIALIZE_SPE_21 is used for initializing the stored procedure SELECT_INPUT_DEPT. The following explains that initialization code.

   - First, there are some global pointers set up for the SQLDA data structures that have been declared for the input parameter, result set, and column headings:

     ```
     /* -- number of columns -- */
     #define SPE_21_max_entries 4
     /* -- number of inputs -- */
     #define SPE_21_max_inputs 1

     EXEC SQL BEGIN DECLARE section;

     /* ---------------------------------------------------
        Set up SQLDA for the input param & return results
     ```

```
   (data)
  ------------------------------------------------------ */
  typedef struct SQLDA_TYPE *sdaptr;
  sdaptr sdai21;
  sdaptr sdao21;

/* -------------------------------------------------------
   Set up SQLDA for the column headings
  ------------------------------------------------------ */
  sdaptr sdan21;
  typedef char (*arrayptr) [1000];
  arrayptr colnames21;

  char temp_stmt21[max_stmt_len + 1];
  /*  area for null_ind values  */
  short  int          NULL_ind[max_columns];
  int input_value;

/* -------------------------------------------------------
   Invoke employee table
  ------------------------------------------------------ */
  exec sql invoke =employee as employee_type;
  struct employee_type  emp_rec;

  EXEC SQL DECLARE emp21_cursor cursor FOR
                   select empnum,
                          first_name,
                          last_name,
                          deptnum
                   from =employee
                   where  deptnum = :input_value
                   browse access;

EXEC SQL END DECLARE section;
```

- The routine INITIALIZE_SPE_21 itself is used for setting up the SQLDA structure to be used for the input parameter:

```
/* -------------------------------------------------------
   SQLDA for storing the column definition for the input
   parameter and the input value. The space needed is
   dependent on how many input parameters are used. This
   area is used by GET_INPUT_PARAMS
  ------------------------------------------------------ */
  sdai21 = allocate_SQLDA (SPE_21_max_inputs, FALSE);
  strncpy (sdai21->eye_catcher, SQLDA_EYE_CATCHER, 2);
  sdai21->num_entries = SPE_21_max_inputs;
```

- Finally, there is code in INITIALIZE_SPE_21 for setting up the SQLDAs for the column descriptions and row data:

```
/* ------------------------------------------------------
   SQLDA for storing the column definition and column
   headings needs to be allocated. The space needed is
   dependent on how many columns are to be returned. This
   area is used by ENCODE_ROW_DESCRIPTOR.
```

```
    -------------------------------------------------------- */
    sdan21 = allocate_SQLDA ( SPE_21_max_entries, FALSE);
    colnames21 = (arrayptr) malloc (500);
    strncpy (sdan21->eye_catcher, SQLDA_EYE_CATCHER, 2);
    sdan21->num_entries = SPE_21_max_entries;

  /* ---------------------------------------------------------
     Separate SQLDA for column definitions needs to be
     allocated. This area is used by ENCODE_ROW_DATA.
    --------------------------------------------------------- */
    sdao21 = allocate_SQLDA ( SPE_21_max_entries, FALSE);
    strncpy (sdao21->eye_catcher, SQLDA_EYE_CATCHER, 2);
    sdao21->num_entries = SPE_21_max_entries;
```

2. Create the stored procedure routine

   This is the routine for the actual stored procedure. There are four basic operations that must be performed:

   a. Get the input parameters, if any.

   b. Call the encode initialization routine

   c. Execute the main logic and for each result set to be returned encode a row descriptor, then encode the row data. After executing each SQL statement, encode an “end statement” message.

   d. Encode an “end procedure” message to indicate that the procedure execution is completed.

   The following example uses the code for the routine SELECT_INPUT_DEPT to illustrate the preceding steps:

   a. Get the input parameters.

```
  /* ---------------------------------------------------------
                 Get the input param values
    --------------------------------------------------------- */
    rc = get_input_params( env
                         , reply_err
                         , max_len
                         , output_buffer
                         , sdai21
                         );
  /* --retrieve input value and copy into host variable--*/
    len=get_data_len(&(sdai21->sqlvar[0]));
    ptr = (char *)&(input_value);
    memcpy (ptr,((long int *)
   (sdai21->sqlvar[0].var_ptr)),len);
```

b. Encode initialization.

```
/* -------------------------------------------------------
   Encode_init must be called to set up each reply buffer
   before writing any data into the buffer for return to
   the NSODBC server.
  ------------------------------------------------------ */
  rc = encode_init( env
                  , max_len
                  , output_buffer
                  , reply_err
                  );
  if (rc != SPELIB_OK)
    {
     strncpy(errtext, "Error attempting encode_init", rc);
     goto Liberr;
    }
```

c. Main logic. The purpose of this stored procedure is to select employee data for a specified department number. The steps involved are to open the cursor for selecting the data from the table and encode a row descriptor for the rows to be returned and then encode the row data for each row fetched from the table. The first step is as follows:

```
  EXEC SQL OPEN emp21_cursor;
  if (sqlcode != SQL_OK)
    {
     strncpy(errtext, "Unable to OPEN cursor", 30);
     goto SQLERR;
    }

/* -------------------------------------------------------
   Call ENCODE_ROW_DESCRIPTOR to set up IPC message with
   column headings
  ------------------------------------------------------ */
  rc = encode_row_descriptor( env
                            , max_len
                            , sdan21
                            , output_buffer
                            , reply_err
                            );
```

Fetch the data and encode the resulting rows:

```
/* ------------------------------------------------------
                      SQL FETCH loop
  ---------------------------------------------------- */
  do
    {
      EXEC SQL FETCH emp21_cursor
                     into :emp_rec.empnum,
                          :emp_rec.first_name,
                          :emp_rec.last_name,
                          :emp_rec.deptnum;
      if ((sqlcode >= SQL_OK) && (sqlcode !=
SQL_NOT_FOUND))
      {
        num_rows++;

/* ------------------------------------------------------
   Call ENCODE_ROW_DATA to set up the IPC message with
each
   row of data being returned.
  ---------------------------------------------------- */
      rc = encode_row_data( env
                          , max_len
                          , sdao21
                          , output_buffer
                          , reply_err
                          );
      if (rc != SPELIB_OK)
        {
          EXEC SQL CLOSE emp21_cursor;
          strncpy(errtext, "Unable to encode Row Data",
          goto LIBERR;
        }
      }
    }
  while ((sqlcode >= SQL_OK) && (sqlcode !=
SQL_NOT_FOUND));
/*------------- end SQL fetch --------------------------*/
```

d. “End statement” and “end proc.” After all rows have been returned, the stored procedure must encode an “end statement” notification to indicate that the SQL statement has been completed. As this is also the end of the stored procedure, an “end proc” notification must also be encoded.

```
/* ------------------------------------------------------
   Every SQL statement should end with an END_STATEMENT
msg.
  ----------------------------------------------------- */
  rc = encode_end_statement(  env, &sqlca);
  if (rc != SPELIB_OK)
     return (rc);
```

```
    /* ---------------------------------------------------
       Every stored procedure should end with an END_PROC
       message
      ---------------------------------------------------- */
      rc = encode_end_proc(  env);
      if (rc != SPELIB_OK)
          *reply_err = rc;
```

3. Create Clean-Up Routine

   This is the routine that will be called when the stored procedure server process is stopping. Typically, this routine is used for freeing up resources allocated for this stored procedure, as follows:

```
boolean dispose_SPE_21(void)
{
  boolean rc = SUCCESS
     dispose(sdai21);
     dispose(sdao21);
     dispose(sdan21);
     dispose(colnames21);
}
```

4. Include the header file generated for the routines INITIALIZE_SPE_21, SELECT_INPUT_DEPT, and DISPOSE_SPE_21 in the source file for the main routine (CSAMPLEC).

```
#pragma page "Example Stored Procedure, CSAMPLEC"
#pragma sql
#include <stdio.h>   nolist
#include "cshell.h"  nolist
/* source in header file for shell routines  */
#include "sp21.h"  nolist
/* source in header for for stored procedure */
```

5. Register the new procedure by adding a call to the routine SPE_REGISTER.

```
main()
{
   ...
register_spe( "select_input_dept" /*stored procedure name*/
             , 17                 /*procedure name length*/
             , select_input_dept  /*procedure function*/
             , initialize_SPE_21  /*initialization for proc*/
             , dispose_SPE_21     /*cleanup for procedure*/
            );
```

Developing a Stored Procedure in COBOL

The basic development model for COBOL is similar to that for C, as described in the preceding subsection. There are differences for COBOL development, however.

The basic development steps are described as follows. A more detailed description of how to create the code for the stored procedure appears later in this subsection.

1. Create the stored procedure (DEPT9000).

   a. For the main sample server code to be able to call a stored procedure, it must be given the name of a function to call for that procedure. This function name is set up when the stored procedure is configured into the sample server. For COBOL procedures, the function name is the PROGRAM-ID of the module for the procedure. An entry must be set up in the CONFIGURATION SECTION to reference the shell routines (SHELLO), or the SPELIB routines (SPELIBO) if the shell routines are not being used; alternatively, you can use a CONSULT directive. You must also set up the WORKING-STORAGE for any SQLDA structures, host variables, or other data items.
   b. Set up the LINKAGE SECTION and PROCEDURE DIVISION to declare the parameters passed from the main server program when this stored procedure is invoked.
   c. Provide an initialization routine for any setup to be performed when the stored procedure is first invoked, such as performing a PREPARE/DESCRIBE to set up the row descriptor information. See the 1000-INITIALIZE routine for some sample code.
   d. Write the main code, such as 2000-PROCESS-REQUEST and 3000-END-PROCESS.
   e. Compile the module.
2. Add the stored procedure to the sample program (SAMPLEC).
   a. Include the external declaration for the COBOL module; for example:

      ```
      _cobol void DEPT9000(.....).
      ```

   b. Register the new stored procedure in the server program by calling the function REGISTER_SPE.
3. Build the new sample program.
   a. Compile and rebind the sample program. See the file BINDFILE for some sample BIND commands.
   b. SQL compile the new sample program.
4. Set up the Pathway configuration for the new server. See EMP-DATA-MIX in the file PATHIN for a sample configuration.
5. Register the stored procedure with the NonStop ODBC Server by using the NOSUTIL command ADD PROCEDURE. See the command for adding the stored procedure DEPT9000 in the file ADDSP.
6. Set up the sample database, if it is not already installed.
7. Execute the stored procedure.
   a. Start the Pathway server; for example:

      ```
      Pathmon /name $empsp/
      Pathcom /in pathin/ $empsp
      ```

b. Send in a request from the client to execute the stored procedure.

Coding a Stored Procedure in COBOL

This subsection covers Steps 1 and 2 from “Developing a Stored Procedure in COBOL” in greater detail, using the sample code in the file SP23COB for the stored procedure DEPT9000 and the file SAMPLEC for the main routine of the sample server.

1. Set up the stored procedure name and point to the shell routines.

   The PROGRAM-ID is set to the name to be used when configuring the stored procedure into the sample server. The CONFIGURATION SECTION must include a reference to the shell functions (SHELLO) or to SPELIBO if you are calling the SPELIB functions directly.

```
IDENTIFICATION DIVISION.
*
*---------------------------------------------------------
* Program is the stored procedure name.
*---------------------------------------------------------
PROGRAM-ID. DEPT9000.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.
SPECIAL-NAMES.
*---------------------------------------------------------
* The shell routine is included via the object file
*---------------------------------------------------------
FILE "SHELLO"  IS C-PROC.
```

   Set up the WORKING-STORAGE section, including any SQLDA structures for returning row data or for handling parameters. In this example, a SQLDA is defined for returning five columns from the employee table. The WORKING-STORAGE section should also include any host variables and other working variables. In this example, there is a SELECT statement for returning the employee data for department 9000.

```
WORKING-STORAGE SECTION.
*********************************************************
* INCLUDE SQLCA AND SQLSA                               *
*********************************************************
EXEC SQL INCLUDE SQLCA END-EXEC.

EXEC SQL INCLUDE SQLDA (SDAO23, 5, SDAN23, 30) END-EXEC.

*********************************************************
* DECLARE HOST VARIABLES                                *
*********************************************************

01 SP-STMT  PIC X(120) VALUE SPACES.

01 SELECT-STMT.
05 FILLER PIC X(40) VALUE
```

```
       "SELECT EMPNUM,FIRST_NAME,LAST_NAME,".
05 FILLER PIC X(40) VALUE
       "DEPTNUM,SALARY FROM =EMPLOYEE      ".
05 FILLER PIC X(40) VALUE
       "WHERE DEPTNUM = 9000 BROWSE ACCESS ".

EXEC SQL SOURCE CODECOB (ERRWS) END-EXEC.

EXEC SQL  END DECLARE SECTION   END-EXEC.

/
*********************************************************
* COBOL PROGRAM WORKING-STORAGE                         *
*********************************************************

01 SP-VARIABLES.
05 SP-DATA             PIC X           VALUE SPACE.
88 NO-MORE-DATA                        VALUE   "Y".
05 SPE-23-MAX-ENTRIES  PIC S9(2)       VALUE     5.
05 WS-MAX-LEN          PIC 9(8) COMP   VALUE  3000.
05 WS-RC               PIC 9(6) COMP   VALUE 32450.
88 SPELIB-OK                           VALUE 32450.
01  I                  PIC S9.
```

One item to note is that the variable WS-RC has been initialized to the value 32450. This is the value returned by the SPELIB library and shell functions to indicate a successful completion (SPELIB_OK). This value is defined in the C “h” file, spelibh, but this file cannot be accessed from a COBOL program, so it is declared explicitly.

2. Set up the LINKAGE SECTION and PROCEDURE DIVISION for the parameters passed to this stored procedure when it is called from the main program. The buffer size is dependent on the stored procedure and must be at least as large as the max_buffer_len parameter that is set when the stored procedure is registered with the NonStop ODBC Server. In this example, the buffer size is set to 3000.

```
LINKAGE SECTION.
*------------------------------------------------------------
* Parameters passing between the shell routines when this
* stored procedure is called.
*------------------------------------------------------------

01 SP-MAX-LEN   PIC 9(9) COMP.
01 SP-BUFFER    PIC X(3000).
01 SP-ENV       PIC X(500).
01 SP-REPLY     PIC 9(4) COMP.
01 SP-RC        PIC 9(4) COMP.

/
*------------------------------------------------------------
* Standard parameters for every stored procedure.
*------------------------------------------------------------
PROCEDURE DIVISION USING SP-ENV,
SP-MAX-LEN,
SP-BUFFER,
SP-REPLY,
SP-RC.
```

3. Set up the initialization code.

   This code is called when the stored procedure is first executed. In this example, an SQLDA is set up that will be used for returning the row descriptor and data, and the SQL statement to be executed for querying the employee table is SQL prepared.

```
1000-INITIALIZE.
MOVE "00" TO SQL-STATUS-SW.
MOVE "D1" TO EYE-CATCHER OF SDAO23.
MOVE SPE-23-MAX-ENTRIES TO NUM-ENTRIES OF SDAO23.

*-----------------------------------------------------------
* To retrieve column descriptions, a dynamic SQL DESCRIBE
* statement is needed, so you need to set up for the
* PREPARE/DESCRIBE stmts.
*-----------------------------------------------------------
MOVE SELECT-STMT TO SP-STMT.
EXEC SQL PREPARE SP23 FROM :SP-STMT END-EXEC.

*-----------------------------------------------------------
* DESCRIBE moves column definitions to sdaO23 and column
* names to colnames. This is used by ENCODE_ROW_DESCRIPTOR
*-----------------------------------------------------------

EXEC SQL
DESCRIBE SP23 INTO :SDAO23 NAMES INTO :SDAN23
END-EXEC.

EXEC SQL
DECLARE CUR23 CURSOR FOR SP23
END-EXEC.
```

4. Write the main code.

   Three basic operations must be performed:

   - Call the initialization code, if needed.
   - Execute the main logic. The first item is to call the encode initialization routine. For each result set returned, encode a row descriptor, then encode the row data.
   - After returning all of the rows, encode an “end statement” message. Then encode an “end procedure” message to indicate that the procedure execution is completed, and set up the return code.

   The following code illustrates these steps. The main loop is set to call the initialization code (1000-INITIALIZE), execute the main logic (2000-PROCESS-REQUEST and 3000-END-PROCESS), and then set up the return code.

- Call initialization, if needed, then execute the main logic.

```
*********************************************************
* MAIN PROGRAM
*********************************************************
0000-MAIN.
IF SP-FIRST-TIME = "Y"
   PERFORM 1000-INITIALIZE.
PERFORM 2000-PROCESS-REQUEST.
PERFORM 3000-END-PROCESS.
MOVE WS-RC TO SP-RC.
MOVE WS-MAX-LEN TO SP-MAX-LEN.
EXIT PROGRAM.
```

- Encode initialization. Call the shell function ENCODE_INIT.

```
*--------------------------------------------------------
* The reply buffer needs to be set up by calling
* ENCODE_INIT.
*--------------------------------------------------------

ENTER C "ENCODE_INIT" IN C-PROC USING
                                SP-ENV,
                                WS-MAX-LEN,
                                SP-BUFFER,
                                SP-REPLY
                       GIVING   WS-RC.

IF NOT SPELIB-OK
PERFORM 9000-LIB-ERROR
EXIT PROGRAM.
```

The purpose of the remaining code is to select employee data for department 9000. The steps involved are to encode a row descriptor for the rows to be returned, open the cursor for selecting the data from the table, and then encode the row data for each row to be returned.

The first step is to encode the row descriptor:

```
*--------------------------------------------------------
* Call ENCODE_ROW_DESCRIPTOR to set up IPC message with
* column colnames
*--------------------------------------------------------

ENTER C "ENCODE_ROW_DESCRIPTOR" IN C-PROC USING
                                          SP-ENV,
                                          WS-MAX-LEN,
                                          SDAO23,
                                          SP-BUFFER,
                                          SP-REPLY
                                 GIVING   WS-RC.

IF NOT SPELIB-OK
PERFORM 9000-LIB-ERROR
EXIT PROGRAM.
```

Before looking at the code to open the cursor and return the row data, it is helpful to examine the code to set up the varptrs in the SQLDA structure for the column data to be returned. This code is explained more fully in the NonStop SQL/MP Programming Reference Manual for your programming language:

```
*---------------------------------------------------------
*  Initialize before SELECT
*  - set up var_ptr to point to the selected columns
*    In this example, allocate var_ptr for 5 columns
*---------------------------------------------------------

MOVE 1 TO I.
ENTER TAL SQLADDR
USING EMPNUM   OF EMP-REC
GIVING VAR-PTR OF SQLVAR OF SDAO23 ( I )
COMPUTE I = I + 1.
```

This code is repeated for each column.

The code to open the cursor and then return the row data, using the shell function ENCODE_ROW_DATA, follows:

```
EXEC SQL OPEN CUR23 END-EXEC.
MOVE SPACE TO SP-DATA.
PERFORM 2020-FETCH-DATA
UNTIL NO-MORE-DATA.

2020-FETCH-DATA.
EXEC SQL
FETCH CUR23 USING DESCRIPTOR :SDAO23
END-EXEC.

*---------------------------------------------------------
*  Call ENCODE_ROW_DATA to set up the IPC message with
*  the row of data being returned.
*---------------------------------------------------------
IF SQLCODE = 0
ENTER  C "ENCODE_ROW_DATA" IN C-PROC USING
                                     SP-ENV,
                                     WS-MAX-LEN,
                                     SDAO23,
                                     SP-BUFFER,
                                     SP-REPLY
                            GIVING   WS-RC
ELSE
IF SQLCODE = 100
MOVE "Y" TO SP-DATA.
IF NOT SPELIB-OK
PERFORM 9000-LIB-ERROR
EXIT PROGRAM.
```

- Encode “end statement” and “end procedure” messages. After all the rows have been returned, the stored procedure must encode an “end statement” message to indicate the SQL statement has been completed. As this is the end of the stored procedure, an “end procedure” message must also be encoded.

```
*---------------------------------------------------------
*  Every SQL statement should end with an
*  END_STATEMENT message
*---------------------------------------------------------
ENTER C "ENCODE_END_STATEMENT" IN C-PROC USING
                                         SP-ENV,
                                         WS-MAX-LEN,
                                         SQLCA,
                                         SP-BUFFER,
                                         SP-REPLY
                                GIVING   WS-RC.

*---------------------------------------------------------
*  Every stored procedure should end with an
*  END_PROC message
*---------------------------------------------------------
ENTER C "ENCODE_END_PROC" IN C-PROC USING
                                    SP-ENV,
                                    WS-MAX-LEN,
                                    SP-BUFFER,
                                    SP-REPLY
                           GIVING   WS-RC.
```

5. The following code is taken from the file SAMPLEC; it is written in C. The external declaration is set up for the COBOL85 stored procedure:

```
_cobol void DEPT9000(char*, char*, char*, char*, char*);
```

6. Register the stored procedure, DEPT9000, by calling the shell function REGISTER_SPE. In this case, the stored procedure name is DEPT9000; the procedure function is the PROGRAM-ID, which is DEPT9000. There are no initialization or clean-up procedures for COBOL85 stored procedures.

```
REGISTER_SPE(  "DEPT9000"
             , 8
             , DEPT9000
             , NULL
             , NULL
            );
```

SPELIB Interface

The SPELIB function library allows Pathway server programs to execute and reply to stored procedure execution requests.

All functions in the SPELIB require at least one parameter, the SPE environment object. Server programmers must allocate space for this structure. The information contained within it is used by the SPELIB during encoding and decoding. Programmers should not attempt to directly manipulate the data stored in the environment structure.

Programmers allocate the environment structure by simply calling “malloc” in C (or a similar function in a different language), using a DEFINE that specifies the size of the structure. For example:

```
malloc (SPEIPC_OBJECT_SIZE);
```

The DEFINE, SPEIPC_OBJECT_SIZE, is provided in the C header file (SPELIBH) containing SPELIB function prototypes. The environment object must be allocated and initialized before other SPELIB functions can be used.

As described earlier, in addition to the SPELIB functions a set of shell routines have been provided with the sample Pathway server program. For each SPELIB function that generates data to be returned to the NonStop ODBC Server, there is an equivalent shell routine. The list of shell routines and their equivalent SPELIB functions follows:

`encode_init`	`SP_SRV_ENCODE_INIT`
`encode_end_proc`	`SP_SRV_ENCODE_END_PROC`
`encode_end_statement`	`SP_SRV_ENCODE_END_STATEMENT`
`encode_sql_diagnostic`	`SP_SRV_ENCODE_SQL_DIAGNOSTIC`
`encode_return_status`	`SP_SRV_ENCODE_RETURN_STATUS`
`encode_output_params`	`SP_SRV_ENCODE_OUTPUT_PARAMS`
`encode_row_descriptor`	`SP_SRV_ENCODE_ROW_DESCR`
`encode_row_data`	`SP_SRV_ENCODE_ROW_DATA`
`encode_printmsg`	`SP_SRV_ENCODE_PRINTMSG`
`encode_raiserror`	`SP_SRV_ENCODE_RAISERROR`
`encode_spelib_error`	`SP_SRV_ENCODE_SPELIB_ERROR`

`get_SPE_info`	`SP_SRV_GET_SERVICE_NAME`
	`SP_SRV_GET_SP_NAME`
	`SP_SRV_GET_REQ_PROCESS_NAME`
	`SP_SRV_GET_USER_NAME`

`get_input_params`	`SP_SRV_GET_INPUT_PARAM_INFO` `SP_SRV_GET_INPUT_PARAMS`

SPELIB Functions for Decoding Requests

The following functions allow server programmers to decode stored procedure execution requests (extract the procedure, get the NonStop ODBC server name, and so on).

As stated previously, these functions cannot be used until the stored procedure execution environment object has been allocated.

SP_SRV_DECODE_INIT

This function must be called before any attempt is made to decode the stored procedure request. This function sets up the environment object that allows decoding of stored procedure requests in “server mode.”

Note. If you are using the shell routines supplied with the sample Pathway server program, this function is called on your behalf by the routine PROCESS_REQUEST; *it must not be called again*.

```
short int SP_SRV_DECODE_INIT
             ( void              * env
             , char              * buffer
             , unsigned long int buffer-len
             )
```

env
input

is a pointer to the environment object STRUCT.

buffer
input

is a pointer to the data received from a READUPDATE call to $RECEIVE.

buffer-len
input

defines the length, in bytes, of the buffer.

The input of this function is a buffer read from $RECEIVE.

This function checks the SPELIB version. If the function returns an SPELIB_VERSION_NOT_SUPPORTED error, the caller should return this error, as well as the current SPELIB version (by calling the SPELIB function SP_SRV_SET_CURRENT_VERSION), to the requester (the NonStop ODBC Server). The requester should reissue the PATHSEND call with the SPELIB version supported by the server. If this attempt fails, the NonStop ODBC Server issues an error message to the client. The following example illustrates this action.

```
NonStop ODBC Server           Pathway Server

PATHSEND(...)                 Read from $RECEIVE
                                 ...
                              rc = SP_SRV_DECODE_INIT (senv,...  )
                              If (rc==SPELIB_VERSION_NOT_SUPPORTED)
                              {
                               /* Allocate reply buffer with */
                               /* length SPELIB_MIN_BUFFER_SIZE */
                               SP_SRV_ENCODE_INIT
                              (senv,replybuf,...)
                               /* Set current SPELIB version */
                               SP_SRV_SET_CURRENT_VERSION (senv);
                               /* Set SPELIB error */
                               SP_SRV_ENCODE_SPELIB_ERROR
                              (senv,rc);
                               /* Reply to PATHSEND.  */
                               REPLY (replybuf,len);
PATHSEND terminates.          }

Decode the reply buffer.

Try the process again using
the returned SPELIB version.
```

SP_SRV_SET_CURRENT_VERSION

This function sets up the current SPELIB version number. The function is used to negotiate SPELIB versions between requester and server, and should be called when the function SP_SRV_DECODE_INIT returns the error SPELIB_VERSION_NOT_SUPPORTED (For more information, see description of SP_SRV_DECODE_INIT).

```
short int SP_SRV_SET_CURRENT_VERSION ( void * env )
```

env
`input`

is a pointer to the environment object STRUCT.

SP_SRV_GET_SERVICE_NAME

This function returns the Pathway service name associated with the stored procedure name.

```
short int SP_SRV_GET_SERVICE_NAME
             ( void      * env
             , char      * service-name
             , short int * service-name-len
             )
```

env
input

is a pointer to the environment object STRUCT.

service-name
output

is a pointer to the Pathway service name.

service-name-len
output

is a pointer to the length of the Pathway service name. The maximum length is 60 characters.

SP_SRV_GET_SP_NAME

This function returns the name of the stored procedure as specified by the client application.

```
short int SP_SRV_GET_SP_NAME
             ( void      * env
             , char      * sp-name
             , short int * sp-name-len
             )
```

env
input

is a pointer to the environment object STRUCT.

sp-name
output

is a pointer to the stored procedure name.

sp-name-len
output

is a pointer to the length of the stored procedure name. The maximum length is 60 characters.

SP_SRV_GET_REQ_PROCESS_NAME

This function returns the name of the requester process that sent the stored procedure execution request. The name is used if the server must open the requester to invoke the call-back capability.

```
short int SP_SRV_GET_REQ_PROCESS_NAME
             ( void      * env
             , char      * req-process-name
             , short int * req-process-name-len
             )
```

env
input

is a pointer to the environment object STRUCT.

req-process-name
output

is a pointer to the requester process name, in Guardian external form.

req-process-name-len
output

is a pointer to the length of the requester process name.

SP_SRV_GET_USER_NAMES

This function provides the logical username (if used by a client) and the Guardian username of the of the client requesting the stored procedure execution request.

```
short int SP_SRV_GET_USER_NAMES
             ( void      * env
             , char      * logical-user-name
             , short int * logical-user-name-len
             , short int * guardian-user-id
             )
```

env
input/output

is a pointer to the environment object STRUCT.

logical-user-name
output

is a pointer to the logical username of the current user.

logical-user-name-len
`output`

is a pointer to the length of the logical username; the maximum length is 60 characters.

guardian-user-id
`output`

is a pointer to the two-character Guardian user ID in internal format.

SP_SRV_GET_NUM_INPUT_PARAMS

This function returns the number of input parameters that accompany the execution request. The caller should use the *num-params* value returned to allocate the SQLDA structure and space for the input parameter values.

```
short int SP_SRV_GET_NUM_INPUT_PARAMS
             ( void      * env
             , short int * num-params
             )
```

env
`input`

is a pointer to the environment object STRUCT.

num-params
`output`

is a pointer to the number of input parameters.

SP_SRV_GET_INPUT_PARAM_INFO

This function decodes a description of the input parameters. Note that if *truncated* is true, it is the responsibility of the Pathway server class program to determine whether to proceed or to flag an error.

```
short int SP_SRV_GET_INPUT_PARAM_INFO
             ( void      * env
             , void      * sqlda
             , short int * truncated
             )
```

env
`input`

is a pointer to the environment object STRUCT.

sqlda
`output`

is a pointer to a SQL descriptor area. to be used for the input parameters. The Pathway programmer must have allocated sufficient space for the SQLDA.

truncated
`output`

indicates whether the input parameter has been truncated.

SP_SRV_GET_INPUT_PARAMS

This function is used to extract the input parameters associated with a stored procedure request.

```
short int SP_SRV_GET_INPUT_PARAMS
             ( void * env
             , void * sqlda
             )
```

env
`input`

is a pointer to the environment object STRUCT.

sqlda
`output`

is a pointer to a SQL descriptor area. The SQLDA contains VAR_PTR and IND_PTR values pointing to valid memory locations. This function also sets the NULL_INFO and IND_PTR values

SPELIB Functions for Encoding Replies

The functions described in this subsection enable a server program to encode the result of a stored procedure execution for transmission back to the NonStop ODBC server.

If an SP_SRV_ENCODE_*xxx* function returns an error code, the server class programmer can use SP_SRV_ENCODE_SPELIB_ERROR to encode the SPELIB error and send the buffer to the NonStop ODBC server using the value returned from the SP_SRV_GET_BUFFERLEN function as the REPLY size or WRITE length.

SP_SRV_ENCODE_INIT

This function must be called before attempting to format each buffer that will be returned to the NonStop ODBC server. It prepares the base buffer format. The environment object must have been previously used to decode a stored procedure request from a NonStop ODBC Server process.

```
short int SP_SRV_ENCODE_INIT
             ( void              * env
             , char              * buffer
             , unsigned long int max-buffer-len
             )
```

env
`input`

is a pointer to the environment object STRUCT.

buffer
`output`

is a pointer to the buffer to be used for encoding a response.

max-buffer-len
`input`

is the length, in bytes, of the response buffer.

SP_SRV_ENCODE_ROW_DESCR

This function encodes an SQL descriptor area (SQLDA) describing a “result set” generated by a stored procedure. A result set is a set of rows returned from an SQL SELECT statement. Each new result set must be started with a call to this function.

```
short int SP_SRV_ENCODE_ROW_DESCR
             ( void * env
             , void * sqlda
             )
```

env
`input`

is a pointer to the environment object STRUCT.

sqlda
`input`

is a pointer to a SQL descriptor area. The caller should set up the SQLDA as if it were the output from a DYNAMIC DESCRIBE statement for all of the columns to be returned.

SP_SRV_ENCODE_ROW_DATA

This function encodes a row of data of the result set generated by the execution of the stored procedure. This function cannot be used until a call to SP_SRV_ENCODE_ROW_DESCR has been successfully completed for that result set.

```
short int SP_SRV_ENCODE_ROW_DATA
             ( void * env
             , void * sqlda
             )
```

env
input

is a pointer to the environment object STRUCT.

sqlda
output

is a pointer to a SQL descriptor area containing the row data to be returned. The caller should set up the SQLDA as if it were the output from a DYNAMIC FETCH statement.

SP_SRV_ENCODE_SQL_DIAGNOSTIC

This function encodes the diagnostic structure of the SQL communications area (SQLCA). This enables the server to report SQL errors, warnings, and any information associated with the execution attempt.

```
short int SP_SRV_ENCODE_SQL_DIAGNOSTIC
             ( void      * env
             , void      * sqlca
             , short int  num-levels
             )
```

env
input

is a pointer to the environment object STRUCT.

sqlca
input

is a pointer to a SQL communications area.

num-levels
input

specifies the number of error levels/entries from the SQLCA to be packed into the IPC. Normally, two entries will be sufficient.

The NonStop ODBC Server does not abort the transaction or terminate processing the remainder of the messages coming back from server class. It is recommended, however, that this message be the last message before END_STMT or END_PROC.

SP_SRV_ENCODE_OUTPUT_PARAMS

This function enables the server to encode any output parameters using one function call.

```
short int SP_SRV_ENCODE_OUTPUT_PARAMS
             ( void * env
             , void * sqlda
             )
```

env
`input`

is a pointer to the environment object STRUCT.

sqlda
`input`

is a pointer to a SQL descriptor area. The caller sets up the SQLDA which includes a descriptor for each output parameter and the value of the output parameter.

SP_SRV_ENCODE_RETURN_STATUS

This function enables the server to encode a return status.

```
short int SP_SRV_ENCODE_RETURN_STATUS
             ( void      * env
             , short int  status
             )
```

env
`input`

is a pointer to the environment object STRUCT.

status
`input`

specifies the return status to be encoded.

SP_SRV_ENCODE_END_STATEMENT

This function enables the server to encode an “end-statement” message. It signals that a stored procedure operation has finished, but that the stored procedure execution is still in progress. This function should be called after each SQL statement has been executed in the stored procedure.

```
short int SP_SRV_ENCODE_END_STATEMENT
             ( void     * env
             , void     * sqlca
             )
```

env
`input`

is a pointer to the environment object STRUCT.

sqlca
`input`

is a pointer to a SQL communications area. The caller sets up the SQLCA as if it were the output from a DYNAMIC FETCH statement.

SP_SRV_ENCODE_END_PROC

This function enables the server to encode an “end-proc” message. It signals that the stored procedure execution is finished. This function must be the last function called and should be followed by a call to REPLY to return the final buffer to the NonStop ODBC Server.

```
short int SP_SRV_ENCODE_END_PROC ( void * env )
```

env
`input`

is a pointer to the environment object STRUCT.

SP_SRV_ENCODE_PRINTMSG

This function enables the server to encode a “print” message. The NonStop ODBC server sends this message to the client.

```
short int SP_SRV_ENCODE_PRINTMSG
             ( void     * env
             , char     * message
             , short int  message-len
             )
```

env
input

is a pointer to the environment object STRUCT.

message
input

is a pointer to the message to be returned to the client.

message-len
input

specifies the length, in bytes, of the message. The maximum is 256 bytes.

SP_SRV_ENCODE_RAISERROR

This function enables the server to encode a “raise error” message. The NonStop ODBC server sends this message to the client. If the NonStop ODBC Server is run in TSQL mode, the global variable @@error also is set to the *error-num* returned from this function

The NonStop ODBC Server does not abort the transaction or terminate processing the remainder of the messages coming back from server class. It is recommended, however, that this message be the last one before SP_SRV_ENCODE_END_STMT or SP_SRV_ENCODE_END_PROC.

```
short int SP_SRV_ENCODE_RAISERROR
             ( void      * env
             , long int    error-num
             , char      * message
             , short int   message-len
             )
```

env
input

is a pointer to the environment object STRUCT.

error-number
input

specifies the error number to be returned to the client.

message
input

is a pointer to the message text to be returned to the client.

message-len
input

specifies the length, in bytes, of the message. The maximum length is 255 bytes.

SP_SRV_ENCODE_SPELIB_ERROR

This function enables the server to encode an SPELIB error.

```
short int SP_SRV_ENCODE_SPELIB_ERROR
             ( void     * env
             , long int  error-num
             )
```

env
`input`

is a pointer to the environment object STRUCT.

error-num
`input`

specifies the error number to be returned to the client.

SP_SRV_GET_BUFFERLEN

This function enables the server programmer to determine how much of the IPC buffer is currently in use. This information can then be used with the call-back to facility to send the current IPC buffer to the NonStop ODBC Server. The value returned can be used in a call to WRITE or REPLY.

```
short int SP_SRV_GET_BUFFERLEN
             ( void               * env
             , unsigned long int * buffer-len
             )
```

env
`input`

is a pointer to the environment object STRUCT.

buffer-len
`output`

is a pointer to the length of the output buffer.

Note. If you are using the SPELIB shell routines provided with the sample server, you should not need to use this function; all call-back processing is handled automatically by the shell routines.

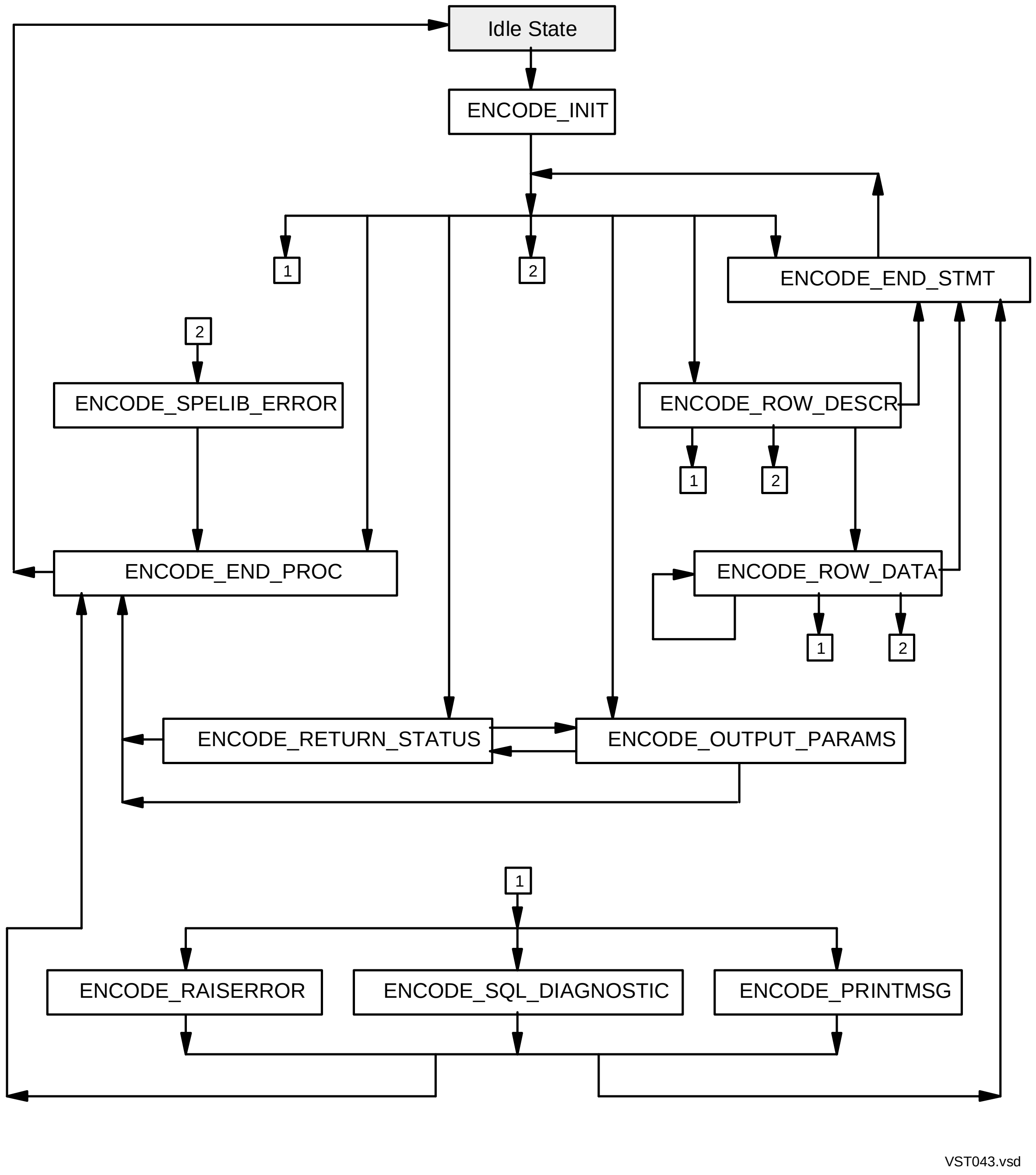
SPELIB Sequence Rules

This subsection specifies the sequence considerations for the Pathway server and for the SPELIB functions (and the equivalent SPELIB shell routines provided with the sample Pathway server).

SP_SRV_ENCODE_INIT must be the first function called in the encoding process; SP_SRV_ENCODE_END_PROC is the last. In between, a prescribed logical sequence must be followed (for example, SP_SRV_ENCODE_ROW_DATA cannot precede SP_SRV_ENCODE_ROW_DESCR).

Figure 5-4 illustrates the sequence required for issuing SPELIB encoding functions. (The SP_SRV_ prefix has been removed from all of the function names in Figure 5-4 because of space limitations.)

Figure 5-4. SPELIB Sequence for Server Encoding



1. represents the path taken if a non-SQL error message, an SQL diagnostic message, or an advisory message is to be returned.
2. represents the path taken if an SPE library error occurs.

SPELIB Errors

For error-handling purpose, a Pathway server class program that emulates a stored procedure is considered to have two sets of errors: SPELIB errors and other errors. The latter might be NonStop SQL/MP execution errors, Pathway program errors, and so on. SPELIB provides SP_SRV_ENCODE_SQL_DIAGNOSTIC for encoding SQL errors or warnings, SP_SRV_ENCODE_RAISERROR for encoding non-SQL errors and SP_SRV_ENCODE_PRINTMSG for reporting informational messages. When the NonStop ODBC Server receives SQL diagnostic or “raise error” messages, it does not terminate processing the SPE messages coming back from server class program. However, it is recommended that SP_SRV_ENCODE_SQL_DIAGNOSTIC (when an SQLCA error—not a warning—occurs) or SP_SRV_ENCODE_RAISERROR should be the last message encoded before issuing SP_SRV_ENCODE_END_STMT or SP_SRV_ENCODE_END_PROC.

SPELIB also provides SP_SRV_ENCODE_SPELIB_ERROR for SPELIB errors. Each SPELIB function returns a status code that signals the success or failure of the called function. It is recommended that if an SPELIB function returns an error on which the server class program cannot take action (or if it is difficult to recover), the server class program should call SP_SRV_ENCODE_SPELIB_ERROR to return the error, then call the system function REPLY to indicate the end of the server processing. It is the responsibility of the server class program to return the reply buffer, with or without a reply error code. A reply error code is normally used for file errors and I/O errors (for example, cannot open the requester for call-back activity). For an SPELIB error, it is recommended that the server class program reply with the reply buffer and with the reply error code set to zero.

For other fatal program errors (for example, “cannot allocate buffer for encode/decode”), a server class program can return a reply buffer with its length set to zero and the reply error code also set to zero.

When the NonStop ODBC Server receives a reply (or write) buffer from the server class program, it checks whether there is any I/O error or if the buffer is empty (an empty reply buffer is considered a fatal error of the server class program). In both cases, the NonStop ODBC Server does not decode the buffer; however, it does send an error message to the client, and also aborts the transaction.

If there is no I/O error, and the buffer is not empty, the NonStop ODBC Server decodes the buffer and sends a result back to the client. If the NonStop ODBC Server encounters any SPELIB error, the NonStop ODBC Server ends further decoding processes, sends an error message back to the client, and aborts the transaction.

Note that SPELIB enforces certain validation to keep the SPE environment intact; however, if the NonStop ODBC Server detects any SPE environment error (bad sequence, bad pointers, and the like), the NonStop ODBC Server cancels the PATHSEND operation.

SPELIB Return Codes

The following describes the error codes returned by the SPELIB interface; they can be used for decision-making in subsequent SPELIB function calls.

- SPELIB_OK

  The operation was completed successfully.

- SPELIB_BUFFER_OVERFLOW

  The reply buffer became full during SPEIPC encoding. The server program should use the call-back mechanism to send the result back to the requester.

  For example:

```
SP_SRV_ENCODE_INIT (senv, replybuf, maxlen);
      ...
SP_SRV_ENCODE_ROW_DESCR (senv, sqlda);
Fetch_loop:
/* SQL Fetch a row data. */
      ...
rc = SP_SRV_ENCODE_ROW_DATA (senv, sqlda);
if (rc == SPELIB_BUFFER_OVERFLOW)
/* Do call-back, OPEN requester process, */
/* if it is not opened already.          */


OPEN (<requester process>);
/* Send this buffer to the requester. */


WRITE (<requester process>, replybuf, len);
/* Allocate buffer or reuse same buffer. Always call  */
/* ENCODE_INIT for any "new" buffer to be used.       */


SP_SRV_ENCODE_INIT (senv, replybuf, maxlen); ...
/* Need to encode the one that previously failed. */
SP_SRV_ENCODE_ROW_DATA (senv, sqlda);
      ...
GOTO Fetch_loop;
/* End Fetch Loop */
      ...
/* Finally, reply to PATHSEND. */
REPLY(repbuf, len, ,0);
```

- SPELIB_INSUFFICIENT_BUFFER

  The reply buffer is too small. The reply buffer size must be greater than SPEIPC_MIN_BUFFER_SIZE and less than SPEIPC_MAX_BUFFER_SIZE (32K bytes). The server class program can readjust the buffer size or encode the SPELIB error and issue a REPLY.

- SPELIB_EXCEEDS_MAX_BUFFER

  The reply buffer size exceeds the 32K bytes limit. The server class program can readjust the buffer size or encode the SPELIB error and issue a REPLY.

- SPELIB_VERSION_NOT_SUPPORTED

  The SPELIB version does not match between the requester and the server. The server class program should issue a REPLY with the currently supported SPELIB version. When the NonStop ODBC Server receives the reply, it attempts to communicate with a server having the SPELIB version supported by the server. If this attempt fails, an “SPELIB version mismatch” error message is sent to the client.

- SPELIB_UNKNOWN_SQLDA_VER

  The SQLDA version is not recognized.

  This is considered as an internal error. The server class program should encode the SPELIB error and issue a REPLY.

- SPELIB_UNKNOWN_SQLCA_VER

  The SQLCA version is not recognized.

  This is considered as an internal error. The server class program should encode the SPELIB error and issue a REPLY.

- SPELIB_INVALID_MSG_TYPE

  An SPELIB message type is not recognized.

  This is considered as an internal error. The server class program should encode the SPELIB error and issue a REPLY.

- SPELIB_INVALID_SEQUENCE

  The SPELIB encoding sequence has been violated.

  This is a caller error. The server class program can either change the calling sequence or encode the SPELIB error and issue a REPLY.

- SPELIB_INVALID_PARM_ADDRESS

  A parameter pointer address is invalid.

  This is considered a caller error. The server class program should encode the SPELIB error and issue a REPLY.

- SPELIB_INVALID_PARM_VALUE

  A parameter value is out of range.

  This is considered a caller error. The server class program should encode the SPELIB error and issue a REPLY.

- SPELIB_INVALID_ENV_OBJECT

An SPELIB environment object is not recognized.

The server class program should encode the SPELIB error and issue a REPLY.

- SPELIB_INVALID_IPC_KEY

  An SPELIB key value is not recognized.

  This is an internal error. The server class program should encode the SPELIB error and issue a REPLY.

- SPELIB_INVALID_SEQ_NUM

  The SPELIB sequence number is out of range.

  This could be an internal error or a user error. The server class program should encode the SPELIB error and issue a REPLY.

- SPELIB_FATAL_ERROR

  An unspecified fatal SPELIB internal error has occurred.

  The server class program should encode the SPELIB error and issue a REPLY.

SPELIB Tracing

The NonStop ODBC Server supports SP_READ and SP_WRITE trace flags that record I/O flows between the NonStop ODBC Server and the Pathway server class program. Tracing is for the NonStop ODBC Server only, not for the Pathway server class.

SP_READ	Records a PATHSEND request to a server class program.
SP_WRITE	Records the result set coming back from a server class program.

Shell Routines Interface

The following describes the entire suite of exported shell routines provided with the NonStop ODBC Server product. Comments in each function explain its purpose. Additional information on the ENCODE_*xxx* and GET_*xxx* functions can be found in the descriptions of the related SPELIB functions, earlier in this subsection.

allocate_SQLDA

```
extern
/*____________________________________________________________________
  This function allocates an SQLDA struct. If requested, it will
  include space for the Names Buffer.

  num_entries:        IN:  number of entries in the new SQLDA struct
  need_names_buffer:  IN:  column names buffer indicator

  Returns:            if successful -- pointer to SQLDA
                      else          -- NULL
  ___________________________________________________________________*/
```

```
void *allocate_SQLDA(  short int num_entries
                     , boolean   need_names_buffer
                    );
```

assign_names_buffer

```
extern
/*_____________________________________________________________________
  This function assigns the names_buffer pointer to the address
  immediately following the SQLDA.

  SQLDA:    IN:  pointer to the SQLDA struct area

  Returns:  if successful -- pointer to Names Buffer area
            else          -- NULL
  ___________________________________________________________________*/

void *assign_names_buffer(void *SQLDA);
```

encode_end_proc

```
extern
/*_____________________________________________________________________
  This function encodes an end proc token into the output buffer. If
  it detects an SPELIB_BUFFER_OVERFLOW condition, it will invoke the
  CALL-BACK facility.

  env:        IN:      pointer to SPELIB environment object
  max_len:    IN:      the maximum output buffer size in bytes
  buffer:     IN/OUT:  pointer to the output buffer
  reply_err:  IN/OUT:  pointer to the reply error

  Returns:    if successful -- SPELIB_OK
              else          -- an SPELIB error
  ___________________________________________________________________*/

short int encode_end_proc(           void      *env
                          , unsigned long  int  max_len
                          ,          char      *buffer
                          ,          short int *reply_err
                         );
```

encode_end_statement

```
extern
/*_____________________________________________________________________
  This function encodes an end statement token into the output buffer.
  If it detects an SPELIB_BUFFER_OVERFLOW condition, it will invoke the
  CALL-BACK facility.

  env:        IN:      pointer to SPELIB environment object
  max_len:    IN:      the maximum output buffer size in bytes
  SQLCA:      IN:      pointer to SQLCA struct holding sql diagnostics
  buffer:     IN/OUT:  pointer to the output buffer
  reply_err:  IN/OUT:  pointer to the reply error

  Returns:    if successful -- SPELIB_OK
              else          -- an SPELIB error
  ___________________________________________________________________*/

short int encode_end_statement(           void      *env
                               , unsigned long  int  max_len
                               ,          void      *SQLCA
```

```
                              ,           char      *buffer
                              ,           short int *reply_err
                             );
```

encode_init

```
extern
/*______________________________________________________________________
  This function initializes the output buffer for server encode.

  env:        IN:      pointer to SPELIB environment object
  max_len:    IN:      the maximum output buffer size in bytes
  buffer:     IN/OUT:  pointer to the output buffer
  reply_err:  IN/OUT:  pointer to the reply error

  Returns:    if successful -- SPELIB_OK
              else          -- an SPELIB error
  ____________________________________________________________________*/

short int encode_init(           void      *env
                     , unsigned long  int  max_len
                     ,           char      *buffer
                     ,           short int *reply_err
                    );
```

encode_output_params

```
extern
/*______________________________________________________________________
  This function encodes an output parameter SQLDA struct into the
  output buffer. If it detects an SPELIB_BUFFER_OVERFLOW condition,
  it will invoke the CALL-BACK facility.

  env:        IN:      pointer to SPELIB environment object
  max_len:    IN:      the maximum output buffer size in bytes
  SQLDA:      IN:      pointer to SQLDA struct containing the params
  buffer:     IN/OUT:  pointer to the output buffer
  reply_err:  IN/OUT:  pointer to the reply error

  Returns:    if successful -- SPELIB_OK
              else          -- an SPELIB error
  ____________________________________________________________________*/

short int encode_output_params(           void      *env
                              , unsigned long  int  max_len
                              ,           void      *SQLDA
                              ,           char      *buffer
                              ,           short int *reply_err
                             );
```

encode_printmsg

```
extern
/*______________________________________________________________________
  This function encodes a printmsg message into the output buffer. If
  it detects an SPELIB_BUFFER_OVERFLOW condition, it will invoke the
  CALL-BACK facility.

  env:        IN:      pointer to SPELIB environment object
  max_len:    IN:      the maximum output buffer size in bytes
  msg_len:    IN:      the printmsg text length in bytes
  msg_text:   IN:      pointer to the printmsg text buffer
  buffer:     IN/OUT:  pointer to the output buffer
  reply_err:  IN/OUT:  pointer to the reply error

  Returns:    if successful -- SPELIB_OK
              else          -- an SPELIB error
  ___________________________________________________________________*/

short int encode_printmsg(           void      *env
                        , unsigned long  int  max_len
                        ,          short int  msg_len
                        ,          char      *msg_text
                        ,          char      *buffer
                        ,          short int *reply_err
                       );
```

encode_raiserror

```
extern
/*____________________________________________________________________
  This function encodes a raiserror message into the output buffer. If
  it detects an SPELIB_BUFFER_OVERFLOW condition, it will invoke the
  CALL-BACK facility.

  env:        IN:      pointer to SPELIB environment object
  max_len:    IN:      the maximum output buffer size in bytes
  err_num:    IN:      error number associated with the raiserror text
  msg_len:    IN:      the raiserror text length in bytes
  msg_text:   IN:      pointer to the raiserror text buffer
  buffer:     IN/OUT:  pointer to the output buffer
  reply_err:  IN/OUT:  pointer to the reply error

  Returns:    if successful -- SPELIB_OK
              else          -- an SPELIB error
  ___________________________________________________________________*/

short int encode_raiserror(           void      *env
                         , unsigned long  int  max_len
                         ,          short int  err_num
                         ,          short int  msg_len
                         ,          char      *msg_text
                         ,          char      *buffer
                         ,          short int *reply_err
                        );
```

encode_return_status.

```
extern
/*____________________________________________________________________
  This function encodes a return status message into the output buffer.
  If it detects an SPELIB_BUFFER_OVERFLOW condition, it will invoke the
  CALL-BACK facility.

  env:        IN:      pointer to SPELIB environment object
  max_len:    IN:      the maximum output buffer size in bytes
  status:     IN:      the return status value
  buffer:     IN/OUT:  pointer to the output buffer
  reply_err:  IN/OUT:  pointer to the reply error

  Returns:    if successful -- SPELIB_OK
              else          -- an SPELIB error
  ___________________________________________________________________*/

short int encode_return_status(           void      *env
                             , unsigned long  int  max_len
                             ,          short int  status
                             ,          char      *buffer
```

```
                        ,           short int *reply_err
                       );
```

encode_row_data

```
extern
/*____________________________________________________________________
  This function encodes a row of data into the output buffer. If it
  detects an SPELIB_BUFFER_OVERFLOW condition, it will invoke the
  CALL-BACK facility.

  env:        IN:      pointer to SPELIB environment object
  max_len:    IN:      the maximum output buffer size in bytes
  SQLDA:      IN:      pointer to SQLDA struct containing the row data
  buffer:     IN/OUT:  pointer to the output buffer
  reply_err:  IN/OUT:  pointer to the reply error

  Returns:    if successful -- SPELIB_OK
              else          -- an SPELIB error
  ___________________________________________________________________*/

short int encode_row_data(          void      *env
                         , unsigned long  int  max_len
                         ,          void      *SQLDA
                         ,          char      *buffer
                         ,          short int *reply_err
                        );
```

encode_row_descriptor

```
extern
/*____________________________________________________________________
  This function encodes an SQLDA descriptor in the output buffer. If it
  detects an SPELIB_BUFFER_OVERFLOW condition, it will invoke the
  CALL-BACK facility.

  env:        IN:      pointer to SPELIB environment object
  max_len:    IN:      the maximum output buffer size in bytes
  SQLDA:      IN:      pointer to SQLDA struct containing the row data
  buffer:     IN/OUT:  pointer to the output buffer
  reply_err:  IN/OUT:  pointer to the reply error

  Returns:    if successful -- SPELIB_OK
              else          -- an SPELIB error
  ___________________________________________________________________*/

short int encode_row_descriptor(          void      *env
                               , unsigned long  int  max_len
                               ,          void      *SQLDA
                               ,          char      *buffer
                               ,          short int *reply_err
                              );
```

encode_spelib_error

```
extern
/*____________________________________________________________________
  This function encodes an spelib_error message into the output buffer.
  If it detects an SPELIB_BUFFER_OVERFLOW condition, it will invoke the
  CALL-BACK facility.

  env:        IN:      pointer to SPELIB environment object
  max_len:    IN:      the maximum output buffer size in bytes
  err_num:    IN:      error number associated with spelib_error text
  buffer:     IN/OUT:  pointer to the output buffer
  reply_err:  IN/OUT:  pointer to the reply error

  Returns:    if successful -- SPELIB_OK
              else          -- an SPELIB error
  ___________________________________________________________________*/

short int encode_spelib_error(           void      *env
                             , unsigned long  int  max_len
                             ,          short int  err_num
                             ,          char      *buffer
                             ,          short int *reply_err
                            );
```

encode_sql_diagnostic

```
extern
/*____________________________________________________________________
  This function encodes an SQL diagnostic SQLCA struct into the output
  buffer. If it detects an SPELIB_BUFFER_OVERFLOW condition, it will
  invoke the CALL-BACK facility.

  env:        IN:      pointer to SPELIB environment object
  max_len:    IN:      the maximum output buffer size in bytes
  SQLCA:      IN:      pointer to SQLCA struct holding sql diagnostics
  buffer:     IN/OUT:  pointer to the output buffer
  reply_err:  IN/OUT:  pointer to the reply error

  Returns:    if successful -- SPELIB_OK
              else          -- an SPELIB error
  ___________________________________________________________________*/

short int encode_sql_diagnostic(           void      *env
                               , unsigned long  int  max_len
                               ,          void      *SQLCA
                               ,          char      *buffer
                               ,          short int *reply_err
                              );
```

get_data_len

```
extern
/*____________________________________________________________________
  This function determines the length of the data in the SQLDA.

  sqlvar:  IN:  pointer to the current sqlvar entry

  Returns:      if successful -- len
                else          -- 0
  ___________________________________________________________________*/

short int get_data_len(struct SQLVAR_TYPE *sqlvar);
```

get_input_params

```
extern
/*____________________________________________________________________
  This function decodes the input parameters from the output buffer.

  env:            IN:      pointer to SPELIB environment object
  reply_err:      IN/OUT:  pointer to the reply error
  max_len:        IN:      the maximum output buffer size in bytes
  output_buffer:  IN/OUT:  pointer to the output buffer
  input_sqlda:    IN/OUT:  pointer to the input parameter sqlda

  Returns:        if successful -- SPELIB_OK
                  else          -- an SPELIB error
  __________________________________________________________________*/

short int get_input_params(           void              *env
                          ,           short  int        *reply_err
                          , unsigned long   int          max_len
                          ,           char              *output_buffer
                          ,           struct SQLDA_TYPE *input_sqlda
                         );
```

get_scale_qualifier

```
extern
/*__________________________________________________________________
  This function determines the scale or the qualifier of the data in
  the SQLDA.

  sqlvar:  IN:  pointer to the current sqlvar entry

  Returns:      if successful -- len
                else          -- 0
  ____________________________________________________________________*/

short int get_scale_qualifier(struct SQLVAR_TYPE *sqlvar);
```

get_SPE_info

```
extern
/*__________________________________________________________________
  This function decodes the service name, sp name, process name, and
  usernames into the spe_info struct.

  env:      IN:  pointer to the SPELIB environment object

  Returns:  if successful -- SPELIB_OK
            else          -- and SPELIB error
  __________________________________________________________________*/

static short int get_SPE_info(void *env);
```

initialize_SQLDA

```
extern
/*__________________________________________________________________
  This function initializes the SQLDA data struct, using the SQLDA
  descriptor struct. It assumes that the SQLDA descriptor struct has
  already been initialized.

  SQLDA_data:        OUT:  the SQLDA data struct
  SQLDA_descriptor:  IN:   the SQLDA descriptor struct
  __________________________________________________________________*/

void initialize_SQLDA(  void *SQLDA_data
                      , void *SQLDA_descriptor
                     );
```

process_event_queue

```
extern
/*____________________________________________________________________
  This function performs waited I/O, $RECEIVE handling.
  __________________________________________________________________*/

void process_event_queue(void);
```

register_spe

```
extern
/*____________________________________________________________________
  This function registers an spe with its Pathway Server class. It
  first executes the constructor function, if present, and then adds
  the spe information to the spe map linked list.

  svc_name  IN:  pointer to the requested service name
  svc_len:  IN:  the byte length of the requested service name
  spe_fun:  IN:  pointer to the requested stored procedure function
  con_fun:  IN:  pointer to associated constructor function, or NULL
  des_fun:  IN:  pointer to associated destructor function, or NULL
  __________________________________________________________________*/

void register_spe(  char        *svc_name
                  , short  int   svc_len
                  , exe_fun_def  spe_fun
                  , con_fun_def  con_fun
                  , des_fun_def  des_fun
                 );
```

TRACEOFF

```
extern
/*____________________________________________________________________
  This SPE function disables the trace facility.

  env:            IN:      SPELIB environment object
  max_len:        IN:      the maximum output buffer size in bytes
  output_buffer:  IN/OUT:  pointer to the output buffer
  reply_err:      IN/OUT:  pointer to the reply error

  Returns:        if successful -- SPELIB_OK
                  else          -- an SPELIB error
  __________________________________________________________________*/

short int TRACEOFF(           void      *env
                   , unsigned long  int  max_len
                   ,          char      *output_buffer
                   ,          short int *reply_err
                  );
```

TRACEON

```
extern
/*____________________________________________________________________
  This SPE function enables the trace facility.

  env:            IN:      SPELIB environment object
  max_len:        IN:      the maximum output buffer size in bytes
  output_buffer:  IN/OUT:  pointer to the output buffer
  reply_err:      IN/OUT:  pointer to the reply error
  Returns:        if successful -- SPELIB_OK
                  else          -- an SPELIB error
  ____________________________________________________________________*/

short int TRACEON(           void      *env
                  , unsigned long  int  max_len
                  ,          char      *output_buffer
                  ,          short int *reply_err
                 );
```

Installation of Stored Procedures

The following describes the procedures and mechanisms for installing stored procedures and for removing them when they are no longer needed. Installation and removal of stored procedures affect the mapping tables ZNUOBJ, ZNUPCOL, ZNUPCI1, ZNUPROC, and ZVUPCOL. See Section 8, HP NonStop ODBC Server Mapping Tables, for details on the mapping tables that support stored procedures.

Catalog Utilities

The NonStop ODBC Server provides two catalog utilities, ADD PROCEDURE and REMOVE PROCEDURE, for installing and removing stored procedures.

For a catalog created with an earlier version of the NonStop ODBC Server, you must execute a USERCAT UPGRADE statement to upgrade the catalog to the level that supports stored procedures before you can execute the ADD PROCEDURE or REMOVE PROCEDURE statements.

All of the catalog utilities are described in detail in Section 7, Managing Customized Catalogs.

ADD PROCEDURE

The ADD PROCEDURE statement adds one entry in the ZNUPROC table for procedure name mapping, and adds one entry in the ZNUPCOL table for each parameter definition.

```
ADD PROCEDURE logical-procedure-name

   PAT[HMON-NAME] pathmon-name

   SER[VERCLASS] server-class-name

   [ SERVICE service-name ]

   [ NUM_RESULT_SETS result-set-count ]

   [ ( parameter-declaration [, ... ] ) ]

   [ { NO_RETURN_STATUS | RETURN_STATUS } ]

   [ MAX_BUF[FER_LEN] ipc-message-buffer-len ]

   [ REMARKS remarks-text ]
```

REMOVE PROCEDURE

The REMOVE PROCEDURE statement deletes the entry for a procedure from the ZNUPROC catalog table, and deletes all entries associated with the procedure from the ZNUPCOL table.

The syntax of the REMOVE PROCEDURE statement is as follows:

```
REMOVE PROCEDURE logical-procedure-name
```

Catalog Access

The following describes how users can gain access to the information in the catalog tables. The methods used depend on whether the user uses Transact-SQL or CORE SQL, and whether the API is DB-LIBRARY or ODBC.

- TSQL/DBLIB Access

  The SQL Server catalog SYSOBJECTS is extended to support procedure objects. Use Transact-SQL to query the catalog for information about stored procedures.

- TSQL/ODBC Access

  The TSQL mode user using ODBC has two ways to obtain catalog information about stored procedures.

  - Use Transact-SQL to query the SQL Server SYSOBJECTS catalog.

- Use the ODBC functions SQLProcedures and SQLProcedureColumns. These are described in the *Microsoft ODBC 2.0 Programmer's Reference and SDK Guide*.

- CORE/ODBC Access

  The CORE SQL user calls the ODBC functions SQLProcedures and SQLProcedureColumns, as described in the *Microsoft ODBC 2.0 Programmer's Reference and SDK Guide*, to obtain catalog information.

Other Considerations

The following describes additional topics related to stored procedures.

Server I/O Protocol

There are two server I/O protocols the NonStop ODBC Server supports and recommends:

- The NonStop ODBC server issues a PATHSEND request; the Pathway server packs all of the data to be returned, then replies. This is the protocol on which most of the existing Pathway/SQL applications operate.
- The Pathway server uses a "call-back" mechanism to handle replies greater than the Guardian IPC size limit (32KB). A Pathway server might want to do a call-back even though the total message size is less than 32K bytes. For example, a server that returns multiple rows may want to start sending back rows of data as they are fetched; this may provide better response time and throughput.

The server programmer must exercise caution when using no-wait I/O for call-back operations. When the call-back option is used, the server program should do all the WRITE operations before calling REPLY. Making a no-wait call to WRITE followed by a call to REPLY does not guarantee that the write data is accepted by the NonStop ODBC Server first unless the call to WRITE is first completed by a call to AWAITIO before the call to REPLY is done.

In general, server programmers must not reply to the PATHSEND from the NonStop ODBC Server until all data has been transferred to the NonStop ODBC server process. This is necessary because Pathway assumes the server is available for use after the server issues a call to REPLY. If the server continues to send data to the NonStop ODBC Server by means of the call-back mechanism after a call to REPLY, future PATHSEND operations to this server may experience unnecessary $RECEIVE queuing. An early reply by the server violates the implied Pathway server protocol.

Transaction Management

The NonStop ODBC server automatically starts a transaction before sending a stored procedure request, if there is no outstanding user transaction already. The transaction identifier is propagated to the server class. Therefore, a TSQL batch can include stored procedure execution and other TSQL statements in the same transaction.

A Pathway server class program can explicitly abort a transaction or cause a transaction to be aborted. As a result, it is necessary for the NonStop ODBC Server to check transaction status after the server class program replies to a stored procedure request.

For a server class program that deals with nonaudited NonStop SQL/MP DDL, or would like to start its own transaction, the server class program should check whether it inherits a transaction from the NonStop ODBC Server. The server class program can call the system function RESUMETRANSACTION to suspend and resume the transaction, if necessary.

The following describes the basic transaction model, in which a Pathway server class program inherits a transaction from the NonStop ODBC Server, executes SQL statements, then issues a REPLY. The NonStop ODBC Server then performs an ABORTTRANSACTION statement or a COMMITTRANSACTION statement, as appropriate.

NonStop ODBC Server

```
      ...
BEGINTRANSACTION (&transid)
PATHSEND(buffer, messagelen
      .
      .
      .
      .
      .
      .
/* PATHSEND COMPLETE
*/?RESUMETRANSACTION (transid);
/* Check transaction status    */
/* abort or commit transaction */
```

Pathway Server

```
 OPEN ($RECEIVE)

READUPDATE from $RECEIVE
/* Get inherited transaction.   */
LASTRECEIVE (&msg_tag);
/* Activate transaction.        */
ACTIVATERECEIVETRANSID (msg_tag);
/* Execute SQL statements here. */
REPLY ()
```

A Pathway server class program can suspend the inherited transaction, and can start transactions of its own. Because transactions started by the server class are transparent to the NonStop ODBC server, it commits or aborts only the transactions started by itself.

If the call-back mechanism is used, it is recommended that SETMODE (requestor_file_num, 117, 1) be called (after OPEN) to suppress attaching *transid* to WRITE or WRITEREAD.

The sample server program supplied with the NonStop ODBC Server handles activation of the transaction.

Fault-Tolerant Programming

The NonStop ODBC server is not a process pair and does not retry ServerClass_Send requests. If a client loses a session with the NonStop ODBC Server, it is the client’s responsibility to determine the state left by the server class process (it is possible for a ServerClass_Send to be completed and for the NonStop ODBC Server to commit the transaction before the session is lost).

Security

The NonStop ODBC server issues the ServerClass_Send in such a way that the Guardian user ID with which the client is connected participates in the call. As described in the *NonStop TS/MP Pathsend and Server Programming Manual*, LINKMON performs authorization checks to ensure that the user ID conforms to the OWNER and SECURITY attributes configured for the server class.

The NonStop ODBC Server user ID need not exist on, nor have remote passwords for, the node where Pathmon is, or the node containing the server class process.

Break Handling

If client sends a “break” signal, the NonStop ODBC Server terminates further processing for a stored procedure invocation. If a PATHSEND is still outstanding, the NonStop ODBC Server calls CANCEL to cancel the request to the Pathway server class program. The CANCEL aborts the current transaction. The Pathway server class program can receive the CANCEL system message or a notification that the transaction has disappeared. The program should then issue a REPLY and end the program.

A stored procedure that uses the shell routines can handle a break or CANCEL message by checking `reply_err` for a value of 2. If `reply_err` is 2, the stored procedure should perform any necessary clean-up routines and then return to its caller. For example:

```
rc = encode_row_data(env,
                     max_len,
                     sdao90,
                     output_buffer,
                     reply_err);
if (rc != SPELIB_OK)
     {
     EXEC SQL CLOSE govtst90_cursor;
     goto LIBERR;
     }

/* check for cancel */
if (*reply_err == 2)
     {
     EXEC SQL CLOSE govtst90_cursor;
     freevar_SPE_90();
     return rc;
     }
```

Sharing Procedures

One mechanism is used by the NonStop ODBC Server to support stored procedure execution for both TSQL mode and CORE mode. The primary requirement is that users in each mode can execute stored procedures defined for that mode (use a valid name, obtain results, and be able to access catalog information about the procedures).

Because there is so much similarity between the TSQL-mode and the CORE-mode stored procedure models and because both kinds of procedures are implemented in server class processes, it is possible for any client to execute any procedure. For clean sharing of procedures, the following guidelines are suggested:

- Use procedure names that are valid names for both Transact-SQL and CORE SQL. That is, use procedure names that are valid CORE SQL names, do not exceed 30 characters in length, and do not have a “;*number*” suffix.
- Use parameter names that are valid names for both Transact-SQL and CORE SQL. That is, use parameter names that are valid CORE SQL names and do not exceed 30 characters in length.
- Use Parameter data types that are legal for both Transact-SQL and CORE SQL.
- Do not instruct the procedure to return output parameters; CORE SQL does not support output parameters. Return all information through result sets.

6 Using Pass-Through Mode

The HP NonStop ODBC Server provides a pass-through mode that you use to execute HP NonStop SQL/MP statements, catalog utility statements, and trace statements, and to specify various ODBC options and configurations.

This section covers the following topics:

Activities Available in Pass-Through Mode

You can perform any of the following activities through the use of the pass-through mode.

Entering NonStop SQL/MP Statements

You might need to execute a NonStop SQL/MP statement to use NonStop SQL/MP features not available in ODBC/SQL Server. For example, you must use pass-through mode to do the following NonStop SQL/MP operations:

- Create a partitioned table
- Create a nonaudited table
- Alter table attributes
- Override the default table attributes
- Create a primary key
- Create an index with a descending key
- Issue control statements (Executor, Query, and Table)
- Lock or unlock an object
- Enable parallel query processing

Running Catalog Utilities

The NonStop ODBC Server provides catalog utilities to help you maintain customized catalogs. You can execute catalog utility statements in pass-through mode.

Setting Server Options

You can enter commands in pass-through mode to specify the configuration of certain server options dealing with the SQL access mode (read/write or read only), the cursor mode (update or read only), the maximum number of rows returned in response to a query, and the transaction isolation level (browse, repeatable, or stable).

Using the Trace Feature

The NonStop ODBC Server provides trace options that you can specify to record information such as SQL statements sent to the NonStop ODBC Server, and SQL/MP statements and error messages generated by the NonStop ODBC Server.

Using pass-through mode you can initiate, start, and stop traces using the following trace options:

- TRA_NAME
- TRA_LOGTABLE_NAME
- LOG_TO_HOMETERM
- INPUT_STREAM
- OUTPUT_STREAM
- NSSQL
- TRA_ERROR
- CACHE_STATISTICS
- SP_WRITE
- SP_READ

For example, the following pass-through statement sets trace options:

```
select "tdm: set tra_name traceit tra_log_tablename
sqldb.sql_user.tracetab sql_ nssql y tra_error y
cache_statistics y"
```

For more information about trace options, see the ADD TRACE statement in Section 7, Managing Customized Catalogs.

Using Resource Accounting

You can start and stop resource accounting with a single command in a user session. For examples, see Examples of Using Pass-Through Mode on page 6-15.

Using Pass-Through Syntax

You use pass-through mode by embedding keywords within a SELECT statement. Pass-through statements use the following syntax:

```
SELECT "TDM:  [ SET server-option-specification ]  "
              [ SQL nonstop-sql-statement        ]
               [ UTIL catalog-utility-statement  ]
```

`TDM:`

is a required keyword that identifies a pass-through statement.

`SET` *`server-option-specification`*

specifies the server option specification to be applied. The syntax of the individual specifications is given under Specifying Server Options in Pass-Through Mode on page 6-4.

Note. Use a separate SET command for each option.

`SQL` *`nonstop-sql-statement`*

specifies the NonStop SQL/MP statement to execute. The syntax of the individual statements is described in the *NonStop SQL/MP Reference Manual*.

The required keyword SQL indicates that this pass-through statement is a NonStop SQL/MP statement.

Specify object names in Guardian format.

`UTIL` *`catalog-utility-statement`*

specifies the catalog utility statement to execute. The syntax of the individual statements is described in Running the Statements Using Pass-Through Mode on page 7-16.

The required keyword UTIL indicates that this pass-through statement is a catalog utility statement.

You cannot execute the catalog utility statements SYSCAT INSTALL or SYSCAT DEINSTALL as pass-through statements, as this would cause all running servers to terminate abnormally; the catalog utility generates an error message if it determines that one of these statements was executed in the pass-through mode.

The SQL PREPARE and SQL EXECUTE statements do not support pass-through statements.

Example Server Option Specification

The following pass-through statement executes the server option specification SET SQL_ACCESS_MODE:

```
select "tdm: set sql_access_mode ro"
```

Example UNLOCK Statement

The following pass-through statement executes the NonStop SQL/MP UNLOCK statement:

```
select "tdm: sql unlock table \test.$disk01.persnl.employee"
```

Example Catalog Utility Statement

The following pass-through statement executes the USERCAT INSTALL catalog utility statement:

```
select "tdm: util usercat install \test.$disk01.persnl"
```

Specifying Server Options in Pass-Through Mode

The following pass-through commands specify server options that affect the SQL transactions in the current SQL session (only).

Setting the Access Mode

The following SET command configures the ODBC option SQL_ACCESS_MODE, which controls the degree of database access given to SQL statements that follow.

```
SELECT "TDM: SET SQL_ACCESS_MODE { RO }  "
                                 { RW }
```

`RO`

specifies that SQL transactions can only read from the database. If this option is set, only SELECT statements are allowed.

`RW`

specifies that SQL transactions can both read from and write to the database. This is the default mode.

Setting the Cursor Default Mode

The following command sets the server option SQL_CURSOR_DEFAULT_MODE, which controls the degree of database access given by default to SQL cursors.

```
SELECT "TDM: SET SQL_CURSOR_DEFAULT_MODE { RO }  "
                                         { RW }
```

`RO`

specifies that cursors not otherwise defined default to the read-only mode.

`RW`

specifies that cursors not otherwise defined default to the read/write mode. This is the default for this option.

Setting the Maximum Number of Rows Returned

The following SET command configures the server option SQL_MAX_ROWS, which controls the maximum number of database rows returned in SELECT statements.

```
SELECT "TDM: SET SQL_MAX_ROWS value"
```

`value`

specifies the maximum number of database rows to be returned in response to a SELECT statement. If *`value`* is zero, the NonStop ODBC Server can return all rows. Zero is the default value for SQL_MAX_ROWS.

If a client request generates a greater number of rows than the maximum number specified, the rows in excess of that number are discarded and are not returned to the client.

Setting the Isolation Level

The following SET command configures the server option SQL_TXN_ISOLATION, which specifies the isolation level to be applied to SQL statements.

```
SELECT "TDM: SET SQL_TXN_ISOLATION level"
```

`level`

specifies the isolation level. The following table shows the valid `level` entries and their NonStop SQL/MP translations:

***level* Entry**	**NonStop SQL/MP Translation**
0	BROWSE ACCESS
1	STABLE ACCESS
2	REPEATABLE ACCESS
3	REPEATABLE ACCESS

If you specify this option, the NonStop ODBC Server applies the appropriate isolation level to each SQL statement; for example, BROWSE ACCESS is meaningful only if the SQL statement is a SELECT statement.

If you do not specify this option, the NonStop ODBC Server uses the NonStop SQL/MP default locking mode, STABLE ACCESS, for all SQL statements.

See the *NonStop SQL/MP Reference Manual* for additional information about transaction isolation.

Note. All of the preceding server options can be set up in a profile record to eliminate the need for specifying them individually. See ADD PROFILE on page 7-66.

Executing a NonStop SQL/MP Statement

When you issue a NonStop SQL/MP statement in pass-through mode, the NonStop ODBC Server executes the statement you specify without any translation taking place. NonStop SQL/MP statements executed in pass-through mode are independent of the NonStop ODBC Server. Results can be returned in the same way as for SQL statements entered without using pass-through mode.

Pass-through statements for NonStop SQL/MP use the following syntax:

```
SELECT "TDM: SQL nonstop-sql-statement"
```

`TDM:`

is a required keyword that identifies a pass-through statement.

`SQL`

is a required keyword that indicates a NonStop SQL/MP statement.

`nonstop-sql-statement`

specifies the NonStop SQL/MP statement to execute. The statement can contain up to 1024 characters.

Specify names for catalogs, tables, views, and indexes in Guardian format.

Examples

The following pass-through statement creates a key-sequenced table:

```
select "tdm: sql create table $vol3.prsntabs.customer
   (custnum   numeric (4) unsigned  no default,
    custname  character (25)        no default,
    city      character (14)        no default,
    primary key custnum)
    catalog $vol3.persnl
    organization key sequenced"
```

```
            select "tdm: util usercat refresh $vol3.persnl"
```

To make the new table available through the NonStop ODBC Server, refresh the NonStop ODBC Server mapping tables using the following catalog utility statement:

```
select "tdm: util usercat refresh $vol3.persnl"
```

For more examples, see Examples of Using Pass-Through Mode on page 6-15.

Transactions and Pass-Through Mode

When used inside a transaction, using pass-through statements differs from using standard statements in the following ways:

- Multiple DDL statements are allowed in a batch.

  When not using pass-through mode, the NonStop ODBC Server allows only one DDL statement in a batch.

- All statements are allowed in a user-defined transaction.

  When not using pass-through mode, the NonStop ODBC Server does not allow DDL statements in user-defined transactions.

- Most pass-through statements do not cause an implicit BEGIN TRANSACTION. The NonStop ODBC Server does, however, start a transaction if the pass-through statement is DELETE, EXECUTE, INSERT, SELECT (even if it contains a FOR BROWSE ACCESS clause), or UPDATE.

  When not using pass-through mode, the NonStop ODBC Server starts a transaction for every SQL statement.

- ROLLBACK TRANSACTION has no effect on DCL statements.

  Changes made with DCL statements are permanent throughout the user's session.

Available NonStop SQL/MP Statements

A subset of NonStop SQL/MP statements is available in pass-through mode. Table 6-1 through Table 6-4 summarize the NonStop SQL/MP DDL, DCL, DML, and Dynamic SQL statements that you can use.

After executing some statements, you might need to update the NonStop ODBC Server mapping tables to make an object available through the NonStop ODBC Server. For details, see Updating the Mapping Tables on page 6-12.

Table 6-1. DDL Statements Available in Pass-Through Mode (page 1 of 3)

Statement	Description	Mapping Table Action Needed	Mapping Table Action Description
ALTER CATALOG	Alters security attributes for catalogs.	None	–
ALTER INDEX	Adds or drops index partitions or alters attributes of indexes such as the index name, security attributes, or file attributes.	USERCAT REFRESH	A mapping table update is needed only if you rename a mapped index and will need to drop the index using the NonStop ODBC Server.
ALTER PROGRAM	Alters security attributes for a program or renames a program.	None	–
ALTER TABLE	Adds new columns to a table, adds and drops table partitions, alters table properties such as security attributes or file attributes, or renames a table. If you change the owner of a mapped table, the new owner name is not reflected in the NonStop ODBC Server mapping tables. You must use the original owner name when accessing the table using the NonStop ODBC Server.	USERCAT REFRESH	A mapping table update is needed only if you rename a mapped table. The file name of the table is changed, but the file name listed in the NonStop ODBC Server mapping tables is not. You cannot access the table using the NonStop ODBC Server unless you refresh the mapping tables. If you rename a mapped table and then update the mapping tables, the ODBC/SQL Server name of the table changes (unless you are changing the name of a table generated by NonStop ODBC Server).

Table 6-1. DDL Statements Available in Pass-Through Mode (page 2 of 3)

Statement	Description	Mapping Table Action Needed	Mapping Table Action Description
ALTER VIEW	Alters security attributes for a view or renames a view. If you change the owner of a mapped view, the new owner name is not reflected in the NonStop ODBC Server mapping tables.	USERCAT REFRESH	A mapping table update is needed only if you rename a mapped view. The file name of the view is changed, but the file name listed in the NonStop ODBC Server mapping tables is not changed. You cannot access the view using the NonStop ODBC Server unless you refresh the mapping tables. If you rename a mapped view and then update the mapping tables, the ODBC/SQL Server name of the view could change.
CREATE CATALOG	Creates a NonStop SQL/MP catalog.	USERCAT INSTALL	NonStop SQL/MP creates a catalog on the specified subvolume. NonStop ODBC Server mapping tables are not created, however. You cannot access the catalog using the NonStop ODBC Server unless you customize the catalog.
CREATE CONSTRAINT	Defines an integrity constraint for a table. When a constraint is in effect, all rows inserted or updated must satisfy the constraint.	None	–
CREATE INDEX	Creates an index on a base table.	USERCAT REFRESH	The index is registered in the NonStop SQL/MP catalog but not in the NonStop ODBC Server mapping tables. (This is an issue only if you want to drop the index later, or if you want an accurate list of user objects seen through the NonStop ODBC Server).

Table 6-1. DDL Statements Available in Pass-Through Mode (page 3 of 3)

Statement	Description	Mapping Table Action Needed	Mapping Table Action Description
CREATE TABLE	Defines and creates a table.	USERCAT REFRESH	The table is registered in the NonStop SQL/MP catalog but not in the NonStop ODBC Server mapping tables. You cannot access the table using the NonStop ODBC Server unless you refresh the mapping tables.
CREATE VIEW	Creates a view.	USERCAT REFRESH	The view is registered in the NonStop SQL/MP catalog but not in the NonStop ODBC Server mapping tables. You cannot access the view using the NonStop ODBC Server unless you refresh the mapping tables.
DROP	Drops a catalog, constraint, program, table, view, or index. Using pass-through mode, you can drop only catalogs that are not customized.	REFRESH	A mapping table update is needed only if you drop a mapped table, view, or index. The object is dropped from the NonStop SQL/MP catalog but is still registered in the NonStop ODBC Server mapping tables. If you then try to create an object with the name of the dropped object, you will receive an error message saying that the object already exists. Dropping a table also causes dependent views and indexes to be drop be dropped. If the table is mapped, the dependent views and indexes remain registered in the mapping tables. You can remove the mapping table entry for the dropped object and its dependent objects by running REFRESH.
UPDATE STATISTICS	Updates statistics stored in the catalog for the specified table.	None	–

Table 6-2. DCL Statements Available in Pass-Through Mode

Statement	Description	Mapping Table Action Needed	Mapping Table Action Description
CONTROL EXECUTOR	Allows or prohibits parallel evaluation of a query by multiple SQL executors.	None	–
CONTROL QUERY	Specifies whether SQL should optimize the speed of processing either for returning the first few rows found or for returning all rows found.	None	–
CONTROL TABLE	Specifies options to control locks and opens on identified tables and views.	None	–
FREE RESOURCES	Releases locks held by the current TMF transaction.	None	–
LOCK TABLE	Locks a table or underlying tables of a view and associated indexes.	None	–
UNLOCK TABLE	Releases locks held on nonaudited tables or views.	None	–

Table 6-3. DML Statements Available in Pass-Through Mode (page 1 of 2)

Statement	Description	Mapping Table Action Needed	Mapping Table Action Description
DELETE	Deletes rows from a table or protection view.	None	–

Table 6-3. DML Statements Available in Pass-Through Mode (page 2 of 2)

Statement	Description	Mapping Table Action Needed	Mapping Table Action Description
INSERT	Inserts rows into a table or protection view.	None	–
SELECT	Retrieves data from tables and views.	None	–
UPDATE	Updates values in columns of a table or protection view.	None	–

Table 6-4. Dynamic SQL Statements Available in Pass-Through Mode

Statement	Description	Mapping Table Action Needed	Mapping Table Action Description
EXECUTE	Executes a previously compiled statement.	Depends	If you execute a DDL statement, you might need to update the mapping tables. See Table 6-1 for information on DDL statements.
PREPARE	Compiles a DDL, DML, or DCL statement.	None	–
RELEASE	Deallocates memory for a prepared SQL statement.	None	–

For details on the NonStop SQL/MP statements, see the *NonStop SQL/MP Reference Manual.*

Updating the Mapping Tables

When you use pass-through mode to execute NonStop SQL/MP statements on customized catalogs, the changes made are not always reflected in the NonStop ODBC Server mapping tables. Therefore, you might need to update the mapping tables if you want the changes visible through the NonStop ODBC Server.

Deciding Whether to Update the Mapping Tables

The following situations indicate when you would need to update the mapping tables after using pass-through mode:

- You create an object in a customized catalog and need to access it using the NonStop ODBC Server.
- You drop an object from a customized catalog and need to create another object by the same name, or you want the object to be removed from the NonStop ODBC Server mapping table.
- You create a catalog and want to access it using the NonStop ODBC Server.
- You rename an object.

These situations indicate when you would *not* need to update the mapping tables after using pass-through mode:

- The pass-through statement operated on a catalog that is not customized.
- You created an object in a customized catalog but do not need to access it using the NonStop ODBC Server.

Updating With the Catalog Utility Statements

To update the mapping tables to reflect changes made in pass-through mode, use one of the following catalog utility statements:

USERCAT INSTALL	Customizes a NonStop SQL/MP catalog
USERCAT REFRESH	Adds a mapping table entry for any object that is not mapped and removes mapping table entries for nonexistent objects

For detailed information on the catalog utility statements, see Section 7, Managing Customized Catalogs.

Frequency of Using the Catalog Utility Statements

You do not need to update the mapping tables each time you use pass-through mode; you can update them only when you need to remove extraneous entries or to map objects that you need to access through the NonStop ODBC Server. Also, because the catalog utility statements affect an entire catalog, you can update the mapping table after executing several pass-through statements. For example, you can do the following:

- Execute one REFRESH statement after creating several tables, views, and indexes in pass-through mode.
- Execute one REFRESH statement after dropping several tables in pass-through mode.
- Execute one REFRESH statement after creating one table and renaming another table in pass-through mode.

Using SELECT Statements

When you execute a SELECT statement in pass-through mode, the NonStop ODBC Server receives NonStop SQL/MP data and translates it to ODBC/SQL Server format.

If the data you select is of the NonStop SQL/MP INTERVAL data type, you will receive an error message, because the NonStop ODBC Server does not support the INTERVAL data type.

Using PREPARE and EXECUTE Statements

The NonStop SQL/MP PREPARE and EXECUTE statements dynamically compile a statement and execute it multiple times in the session. Executing a statement by using PREPARE and EXECUTE improves performance because the statement does not need to compile every time it is executed.

This subsection describes how using these statements through the NonStop ODBC Server differs from using them in NonStop SQL/MP. When used through the NonStop ODBC Server, RELEASE is the same as when used in NonStop SQL/MP. For details on these statements, see the *NonStop SQL/MP Reference Manual.*

PREPARE Statement

When used through the NonStop ODBC Server, the implementation of the PREPARE statement differs from the NonStop SQL implementation in the following ways:

- The *string* portion of the PREPARE statement is not enclosed in quotes:

```
PREPARE statement-name FROM string
```

  For examples, see Using PREPARE, EXECUTE, and RELEASE Statements on page 6-20.

- Only the CORE SQL statements listed in Table 3-8 or the Transact-SQL statements listed in Table 4-14 through Table 4-16 can be prepared.
- Named parameters are not allowed.
- You must use the SQLCI version of the statement (as opposed to the programmatic SQL version).

The following considerations apply to using PREPARE in both NonStop SQL/MP and the NonStop ODBC Server:

- Parameters are allowed anywhere a numeric or string literal is allowed. You cannot, however, use an unnamed parameter as a table or column name.
- The prepared statement is recompiled when you switch to another database.
- The *statement-name* must be a NonStop SQL/MP identifier.

EXECUTE Statement

When used through the NonStop ODBC Server, the implementation of the EXECUTE statement differs from the NonStop SQL implementation in the following ways:

- Named parameters are not allowed.
- You must use the SQLCI version of the statement (as opposed to the programmatic SQL version).

The following considerations apply to using EXECUTE in both NonStop SQL/MP and the NonStop ODBC Server:

- When a transaction is in progress, you cannot execute a prepared DDL statement (except UPDATE STATISTICS) that operates on a nonaudited table or index.
- You must specify a value in the USING clause for every unnamed parameter in the prepared statement.

Examples of Using Pass-Through Mode

This subsection contains examples of executing NonStop SQL/MP statements in pass-through mode. The examples show how to create or alter catalogs, tables, views, and indexes. There are also some examples of selecting and locking data and using PREPARE, EXECUTE, and RELEASE Statements.

The examples in this subsection are all based on the following assumptions:

- You are using pass-through mode to do operations on customized catalogs.
- You want objects available through the NonStop ODBC Server after creating or altering them in pass-through mode.

If you are using pass-through mode to do operations on catalogs that are not customized, you can ignore the information about updating the mapping tables.

Creating or Altering Catalogs

The following examples show how to use pass-through mode to create or alter catalogs.

Specifying the Security When Creating a Catalog

When you create a catalog using the NonStop ODBC Server, you can use the Transact-SQL statement CREATE DATABASE. You cannot specify file-system security with this statement. You can specify the security, however, by creating the catalog in pass-through mode:

```
select "tdm: sql create catalog
   \test01.$users.addres
   secure 'nunu' "
```

To make the new catalog available through the NonStop ODBC Server, customize it using the USERCAT INSTALL statement:

```
select "tdm: util usercat install \test01.$users.addres"
```

Creating or Altering Tables

The following examples show how to use pass-through mode to create or alter tables.

Note. If you create a nonaudited table in pass-through mode, any DML operation you perform on that table must be done in pass-through mode because the NonStop ODBC Server starts a transaction for every SQL statement executed.

Creating a Nonaudited Table

All tables created with the SQL statement CREATE TABLE are audited by TMF. You can create a nonaudited table, however, by creating the table in pass-through mode:

```
select "tdm: sql create table $vol3.saletabs.customer
   (custnum   numeric   (4) unsigned  no default,
    custname  character (25)          no default,
    city      character (14)          no default)
   catalog $vol3.sales
    no audit"
```

After creating a table in pass-through mode, you can update the mapping tables using the REFRESH statement:

```
select "tdm: util usercat refresh $vol3.sales"
```

Creating a Table With a Primary Key

When you create a table with the SQL statement CREATE TABLE, you cannot specify a primary key. You can specify a primary key, however, by creating the table in pass-through mode:

```
select "tdm: sql create table $vol3.saletabs.customer
   (custnum   numeric   (4) unsigned  no default,
    custname  character (25)          no default,
    city      character (14)          default 'San Jose',
    primary key custnum)
   catalog $vol3.sales
   organization key sequenced"
```

After creating a table in pass-through mode, you can update the mapping tables using the REFRESH statement:

```
select "tdm: util usercat refresh $vol3.sales"
```

Creating Columns With Default Values

When you create a table with the CREATE TABLE statement, you cannot specify default values for columns. You can specify default values, however, by creating the table in pass-through mode:

```
select "tdm: sql create table $vol3.saletabs.customer
   (custnum   numeric   (4) unsigned  no default,
    custname  character (25)          no default,
    city      character (14)          default 'San Jose')
   catalog $vol3.sales”
```

After creating a table in pass-through mode, you can update the mapping tables using the REFRESH statement:

```
select "tdm: util usercat refresh $vol3.sales"
```

You can also create a column with default values using the NonStop SQL/MP statement ALTER TABLE in pass-through mode:

```
select "tdm: sql alter table $vol3.saletabs.customer
   add column last_order_date
   numeric (6) default 860000
   catalog $vol3.sales”
```

You do not need to update the mapping tables after adding columns.

Creating a Constraint

The NonStop SQL/MP statement CREATE CONSTRAINT is not available in CORE SQL or Transact-SQL. You can create a constraint, however, by using pass-through mode:

```
select "tdm: sql create constraint
   on $users.persnl.employee
   check empnum between 0001 and 9999"
```

You do not need to update the mapping tables after adding or dropping a constraint.

Note. The equivalent in SQL Server is the statement CREATE RULE combined with the stored procedure SP_BINDRULE; however, neither of these is supported by the NonStop ODBC Server.

Altering Table Attributes

When you create a table using the CREATE TABLE statement, the table is created with NonStop SQL/MP default attributes. You can change the attributes, however, by altering the table in pass-through mode:

```
select "tdm: sql alter table $vol3.prsntabs.employee
   secure 'cc-u' "
```

```
select "tdm: sql alter table $vol3.prsntabs.dept
   nopurgeuntil jan 01 1995
   no audit"
```

You do not need to update the mapping tables after altering table attributes.

Creating or Altering Views

The following examples show how to use pass-through mode to create or alter views.

Creating a Protection or Shorthand View

When you create a view using the SQL statement CREATE VIEW, you do not know whether the view will be a protection view or a shorthand view.

However, you can specify the view type by creating the view in pass-through mode:

```
select "tdm: sql create view $vol3.prsntabs.empview
   as select * from $vol3.prsntabs.employee
      where empnum > 1000
   for protection
   catalog $vol3.persnl"
```

After creating a view in pass-through mode, you can update the mapping tables using the REFRESH statement:

```
select "tdm: util usercat refresh $vol3.persnl"
```

Creating a View WITH CHECK OPTION

When you create a view using the SQL statement CREATE VIEW, you cannot specify WITH CHECK OPTION, which specifies that a row cannot be inserted through the view unless the row satisfies the view definition. You can specify the check option, however, by creating the view in pass-through mode:

```
select "tdm: sql create view $vol3.prsntabs.empview
   as select * from $vol3.prsntabs.employee
      where empnum > 1000
   for protection
   catalog $vol3.persnl
   secure 'n-nu'
   with check option"
```

After creating a view in pass-through mode, you can update the mapping tables using the REFRESH statement:

```
select "tdm: util usercat refresh $vol3.persnl"
```

Managing Resource Accounting

The following example statement starts resource accounting:

```
select "tdm: set acc_mode_on y"
```

The following example statement stops resource accounting:

```
select "tdm: set acc_mode_on n"
```

For details on resource accounting, see the *NonStop ODBC Server Installation and Management Manual.*

Managing Indexes

The following examples show how to use pass-through mode to manage indexes.

Creating a Descending Index

All indexes created with the NonStop ODBC Server using the SQL statement CREATE INDEX use ascending keys. You can create an index with a descending key, however, by using pass-through mode:

```
select "tdm: sql create unique index $vol3.prsntabs.xempnam
   on $vol3.prsntabs.employee
      (last_name descending,
       first_name)
   catalog $vol3.persnl"
```

After creating an index in pass-through mode, you can update the mapping tables using the REFRESH statement:

```
select "tdm: util usercat refresh $vol3.persnl"
```

Locking Data

The following examples show how to use pass-through mode to lock data.

Locking or Unlocking a Table

You can obtain an exclusive lock by using the LOCK statement in pass-through mode:

```
select "tdm: sql lock table $vol3.prsntabs.employee
   in exclusive mode"
```

```
update $vol3_pertabs..employee
   set salary = salary * 1.15
   where deptnum = 2500
```

```
select "tdm: sql unlock table $vol3.prsntabs.employee"
```

You do not need to update the mapping tables after locking or unlocking a table. The preceding UPDATE statement is not executed using pass-through mode.

Using PREPARE, EXECUTE, and RELEASE Statements

The following examples show how to use the PREPARE, EXECUTE, and RELEASE statements.

The following statement prepares an INSERT statement and saves it as EMPIN:

```
select "tdm: sql prepare empin
   from insert into $vol3.prsntabs.employee
   values (?, ?, ?, ?, ?, ?)"
```

The following statement executes the prepared statement, specifying a value for each column (EMPNUM, FIRST_NAME, LAST_NAME, DEPTNUM, JOBCODE, SALARY) of the EMPLOYEE table:

```
select "tdm: sql execute empin
   using 606, 'Amy', 'Ryan', 3100, 300, 55000"
```

The following statement releases the prepared statement:

```
select "tdm: sql release empin"
```

Managing Traces

The following example statement sets several trace options:

```
select "tdm: set tra_name trace01 tra_log_tablename
data_disk01_persnl..logtab1 log_to_hometerm y tra_error y"
```

The following example statement starts a server trace:

```
select "tdm: set tra_mode_on y"
```

The following example statement stops a server trace:

```
select "tdm: set tra_mode_on n"
```

7 Managing Customized Catalogs

A customized catalog is a HP NonStop SQL/MP catalog available through the HP NonStop ODBC Server. To make the customized catalog available, the NonStop ODBC Server creates special tables to map ODBC/SQL Server objects and data types to NonStop SQL/MP objects and data types or to Pathway objects. You will occasionally need to access these mapping tables to locate objects or to perform maintenance tasks.

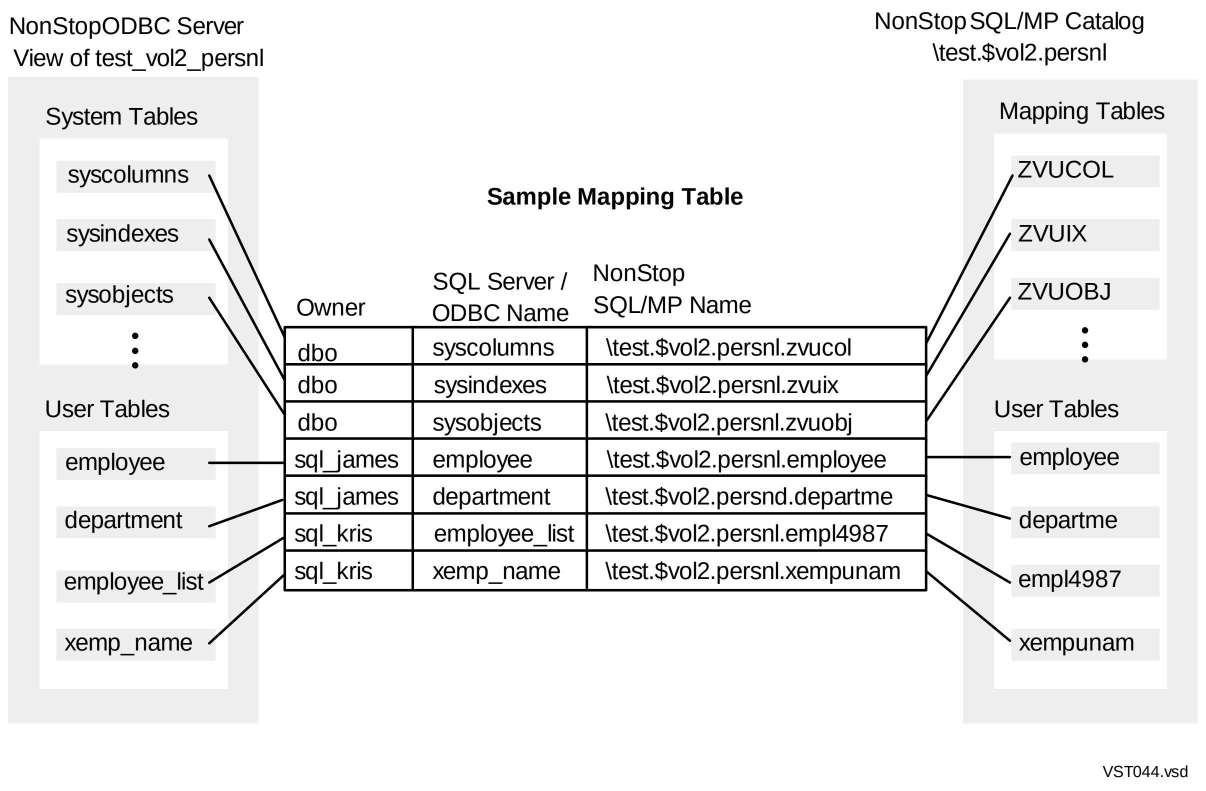
A customized catalog corresponds to an ODBC/SQL Server database.

This section explains what customization tasks you need to perform and when to perform these tasks. The section covers the following topics:

- Reasons to access the mapping tables
- Locating objects using the mapping tables
- Renaming ODBC/SQL Server objects
- Running the catalog utility statements
- Customizing the system catalog—SYSCAT INSTALL
- Customizing a user catalog—USERCAT INSTALL
- Decustomizing the system catalog—SYSCAT DEINSTALL
- Decustomizing a user catalog—USERCAT DEINSTALL
- Adding objects to the mapping tables, removing extraneous mapping entries, and rebuilding mapping tables—SYSCAT REFRESH and USERCAT REFRESH
- Reporting referential inconsistencies—SYSCAT VALIDATE and USERCAT VALIDATE
- Dropping temporary objects—CLEANUP
- Upgrading catalogs to the current version—SYSCAT UPGRADE and USERCAT UPGRADE
- Adding, changing, and deleting mapping and configuration entities with the mapping table statements listed in Table 7-4 on page 7-38

Figure 7-1 illustrates a customized NonStop SQL/MP catalog, PERSNL, and a portion of the table that maps the customized objects to ODBC/SQL Server object names.

Figure 7-1. A NonStop ODBC Server Mapping Table



Owner	SQL Server / ODBC Name	NonStop SQL/MP Name
dbo	syscolumns	\test.$vol2.persnl.zvucol
dbo	sysindexes	\test.$vol2.persnl.zvuix
dbo	sysobjects	\test.$vol2.persnl.zvuobj
sql_james	employee	\test.$vol2.persnl.employee
sql_james	department	\test.$vol2.persnd.departme
sql_kris	employee_list	\test.$vol2.persnl.empl4987
sql_kris	xemp_name	\test.$vol2.persnl.xempunam

Note that in the ODBC/SQL Server view of the database in Figure 7-1, the object names are fully qualified: for example, test_vol2_persnl.sql_kris.xemp_name.

Mapping tables also map ODBC/SQL Server system tables and data types. For a complete description of the mapping tables, see Section 8, HP NonStop ODBC Server Mapping Tables.

Summary of Tasks

To make a NonStop SQL/MP catalog and its associated objects available through the NonStop ODBC Server, you must customize the catalog. You also perform maintenance tasks that involve the catalogs and mapping tables.

Table 7-1 summarizes tasks that involve customization and mapping tables and lists where to find that information.

Table 7-1. Tasks That Involve Customization and Catalog Utilities (page 1 of 2)

Task	What You Need to Do	Where to Find Further Information
Access a NonStop SQL/MP catalog using the NonStop ODBC Server	First, customize the catalog.	SYSCAT INSTALL Statement on page 7-18 and USERCAT INSTALL Statement on page 7-20
Create a database using the NonStop ODBC Server	Log on to the NonStop ODBC Server and execute the Transact-SQL statement CREATE DATABASE.	(See Section 4, Transact-SQL Language, for information on creating databases)
	You do not need to perform any additional steps to customize the catalog, because when you execute CREATE DATABASE, the NonStop ODBC Server creates a NonStop SQL/MP catalog and all the necessary mapping tables, and completes the customizing automatically.	
	CORE SQL has no CREATE DATABASE statement; instead,	USERCAT INSTALL Statement on page 7-20
Use pass-through mode to create a NonStop SQL/MP object in a customized catalog	When you create the object, it is not visible through the NonStop ODBC Server. To make the object visible, update the mapping tables after you create the object.	SYSCAT REFRESH Statement on page 7-27 and USERCAT REFRESH Statement on page 7-29 or ADD TABLE on page 7-105, ADD INDEX on page 7-50, or ADD VIEW on page 7-119
Use NonStop SQL/MP to create an object in a customized catalog	When you add the object, it is not visible through the NonStop ODBC Server, so you must first refresh or update the mapping tables.	SYSCAT REFRESH Statement on page 7-27 and USERCAT REFRESH Statement on page 7-29 or ADD TABLE on page 7-105, ADD INDEX on page 7-50, or ADD VIEW on page 7-119
Access an object using NonStop SQL/MP when the object was created using the NonStop ODBC Server	Determine the NonStop SQL/MP file name of the object by looking in the mapping tables.	Locating Objects Using the Mapping Tables on page 7-4

Table 7-1. Tasks That Involve Customization and Catalog Utilities (page 2 of 2)

Task	What You Need to Do	Where to Find Further Information
List all NonStop SQL/MP catalogs available through the NonStop ODBC Server	Check the mapping table that corresponds to the ODBC/SQL Server system table SYSDATABASES.	Locating Objects Using the Mapping Tables on page 7-4
Recover a mapping table that you accidentally purged	Rebuild the mapping tables.	SYSCAT REFRESH Statement on page 7-27 and USERCAT REFRESH Statement on page 7-29
Use new features of the NonStop ODBC Server	First, upgrade the catalogs to the version that provides support for those features.	SYSCAT UPGRADE Statement on page 7-36 and USERCAT UPGRADE Statement on page 7-37
Remove the customization so the catalog is no longer available through the NonStop ODBC Server	Decustomize the catalog.	SYSCAT DEINSTALL Statement on page 7-22 and USERCAT DEINSTALL Statement on page 7-24
Drop a customized catalog using NonStop SQL/MP	The catalog will still be listed in the mapping tables, so you must refresh the system catalog.	SYSCAT REFRESH Statement on page 7-27 and USERCAT REFRESH Statement on page 7-29

Locating Objects Using the Mapping Tables

You will sometimes need to use the mapping tables to locate objects. Because the mapping tables map ODBC/SQL Server object names to NonStop SQL/MP file names, you might need information from the mapping table in the following cases:

- To list all customized catalogs
- To determine the file name of a table so you can use the table name with NonStop SQL/MP statements or pass-through statements
- To determine the ODBC/SQL Server name of an object you created by using NonStop SQL/MP then added to the mapping tables by refreshing them

Listing Customized Catalogs

Although you can list customized catalogs using the NonStop ODBC Server by selecting from the table SYSDATABASES, you might also want to list customized catalogs using SQLCI so you can see both the NonStop SQL/MP name and the ODBC/SQL Server name of the catalog.

You can list customized catalogs by accessing the mapping table ZNSDB, which is in the same subvolume as your NonStop SQL/MP system catalog.

For example, if the NonStop SQL/MP system catalog is in \TEST01.$SYSTEM.SQL, you can list all customized catalogs by executing the following statement in SQLCI:

```
SELECT n_catalog, t_dbname FROM \test01.$system.sql.znsdb;
```

SQLCI displays a row for each customized catalog. Each row lists the columns N_CATALOG and T_DBNAME, which contain the NonStop SQL/MP catalog name and ODBC/SQL Server database name.

For details about the mapping table ZNSDB, see Section 8, HP NonStop ODBC Server Mapping Tables.

Locating an Object Using the ODBC/SQL Server Object Name

Although you might know the ODBC/SQL Server name of an object in a customized catalog, you might also need to know the corresponding NonStop SQL/MP name. For example, you must have the NonStop SQL/MP name of a table if you want to do the following:

- Use the table name in pass-through statements
- Use the table name in NonStop SQL/MP statements

You can locate the NonStop SQL/MP name by accessing the mapping table ZNUOBJ, which is in the same subvolume as your customized catalog.

For example, suppose you know the following information about a table:

- Your database name is TEST_VOL2_PERSNL.
- The owner of the database is ADMIN_LEN.
- The owner of the table is ADMIN_LEN.
- The ODBC/SQL Server name of the table is FUTURES_TABLE.

You can determine the NonStop SQL/MP name of the table by executing the following statement in SQLCI. (Note that you must enter the owner name and table name in uppercase letters.)

```
SELECT n_objname
   FROM \test.$vol2.persnl.znuobj
   WHERE t_uname = "ADMIN_LEN"
   AND t_objname = "FUTURES_TABLE"
```

A row is displayed listing the column N_OBJNAME, which contains the NonStop SQL/MP name of the table.

For details about the mapping table ZNUOBJ, see ZNUOBJ (For Logical Object Names Mapping) on page 8-51.

Locating an Object Using the NonStop SQL/MP Object Name

Sometimes you will know the NonStop SQL/MP name of an object but not the corresponding ODBC/SQL Server name. For example, you must have the ODBC/SQL Server name of a table if the following occurs:

- You create the table in NonStop SQL/MP, refresh the mapping tables, and then need to access the table using the NonStop ODBC Server.
- You customize a NonStop SQL/MP catalog, a table name is mapped, but you cannot locate the table using the NonStop ODBC Server.

You can determine the ODBC/SQL Server name by accessing the mapping table ZNUOBJ, which is in the same subvolume as your customized catalog.

For example, suppose you know the following information about a table:

- Your database name is TEST_VOL2_PERSNL.
- The owner of the database is ADMIN_JANET.
- The NonStop SQL/MP name of the table is \TEST.$VOL13.ALLEMPS.EMPLOYEE.

By using the NonStop ODBC Server, you can determine the ODBC/SQL Server owner name and table name by executing the following statement.

```
SELECT t_uname, t_objname
   FROM test_vol2_persnl.admin_janet.znuobj
   WHERE n_objname = "\TEST.$VOL13.ALLEMPS.EMPLOYEE"
```

A row is returned listing the columns T_UNAME and T_OBJNAME, which contain the ODBC/SQL Server owner name and table name.

Note. You must enter the entire file name, including the node, volume, and subvolume, and the name must appear in uppercase letters.

For details about the mapping table ZNUOBJ, see ZNUOBJ (For Logical Object Names Mapping) on page 8-51.

Renaming ODBC or SQL Server Objects

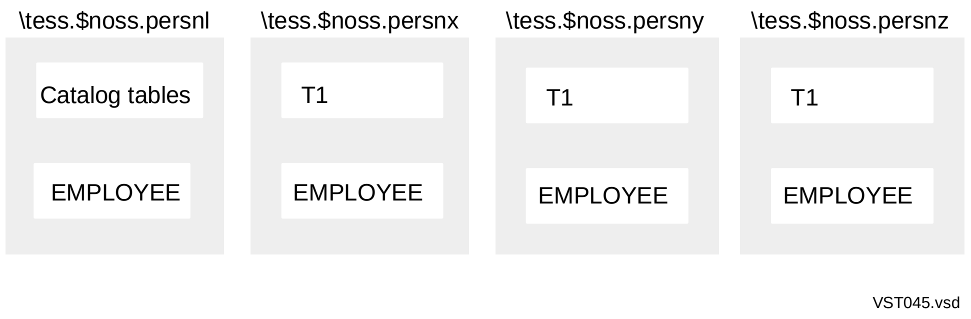
When you customize or refresh a catalog, the NonStop ODBC Server assigns ODBC/SQL Server names to the objects in the catalog not already registered in the NonStop ODBC Server catalog.

How Names Are Assigned

When assigning ODBC/SQL Server object names, the NonStop ODBC Server usually assigns the Guardian file name. For example, a table named \TESS.$NOSS.PERSNL.EMPLOYEE is given the ODBC/SQL Server name EMPLOYEE. Sometimes, however, the NonStop ODBC Server must assign a name other than the Guardian file name. This situation usually occurs when a catalog contains multiple objects that have the same file name but are in different subvolumes.

For example, four tables named EMPLOYEE (in different subvolumes) and three tables named T1 could be registered in the catalog \TESS.$NOSS.PERSNL, as shown in Figure 7-2.

Figure 7-2. Tables With Duplicate Names Registered in One Catalog



USERCAT INSTALL would assign the tables different names and generate the messages listed in Example 7-1.

Example 7-1. Name Mapping Warnings From USERCAT INSTALL

```
select "tdm: util usercat install \tess.$noss.persnl"

ODBC Server message 32652:
Object table \TESS.$NOSS.PERSNX.EMPLOYEE mapping to T_EMPLOYEE567490 added.
ODBC Server message 32652:
Object table \TESS.$NOSS.PERSNY.EMPLOYEE mapping to T_EMPLOYEE724838 added.
ODBC Server message 32652:
Object table \TESS.$NOSS.PERSNY.T1 mapping to T_T1809120 added.
ODBC Server message 32652:
Object table \TESS.$NOSS.PERSNZ.EMPLOYEE mapping to T_EMPLOYEE891245 added.
ODBC Server message 32652:
Object table \TESS.$NOSS.PERSNZ.T1 mapping to T_T1961851 added.
```

The NonStop ODBC Server assigns simple names as shown in Example 7-1 only if the tables are not partitioned. If a table is partitioned, new names are not assigned. Instead, the NonStop ODBC Server chooses one partition per object to map in ZNUOBJ. Even if primary and secondary partitions of the same object are registered in different catalogs, ZNUOBJ has only one entry for the object. NonStop ODBC Server determines which partition to map as follows:

- If the primary partition is local**,** the primary partition is the mapped object.
- If the primary partition is not local, the first partitioned object in the list of objects in the catalog (TABLES, INDEXES) is the mapped object.
- If the object has no local partitions, the primary partition is the mapped object.

When you need to access the object by using the NonStop ODBC Server, use the ODBC or SQL Server name given during USERCAT INSTALL or USERCAT REFRESH. You can change the names by executing mapping table statements that modify the NonStop ODBC Server mapping table ZNUOBJ. The following subsections describe how to change the names.

Listing the Mapped Object Names

You can list the mapped names by selecting from the NonStop ODBC Server object mapping table ZNUOBJ, where the mapped names are registered.

Table 7-2 lists and describes the columns in ZNUOBJ.

Table 7-2. Description of ZNUOBJ (page 1 of 2)

Column Name	**Data Type**	**Description**
T_UID	UNSIGNED INT (2)	The numeric user ID of the owner.
T_OBJNAME	VARCHAR (60)	The ODBC/SQL Server object name in uppercase letters.
T_UNAME	VARCHAR (60)	The owner name in uppercase letters.
T_OBJTYPE	CHAR (2)	A code indicating the object type: N NonStop SQL/MP catalog table or NonStop ODBC Server mapping table S System table view T Temporary table U User table V View
T_OBJID	INT(4)	An object ID used internally by the NonStop ODBC Server.
N_OBJNAME	CHAR (34)	The fully qualified NonStop SQL/MP object name in uppercase letters.

Table 7-2. Description of ZNUOBJ (page 2 of 2)

Column Name	**Data Type**	**Description**
N_OBJTYPE	CHAR (2)	The NonStop SQL/MP object type: TA Table VI View
N_ORIGIN	CHAR (1)	A code indicating the type of statement that created the object: T Transact-SQL or CORE SQL N NonStop SQL/MP
N_SESSIONID	LARGEINT	A session ID used internally by the NonStop ODBC Server.

For details about ZNUOBJ and the other mapping tables, see Database and Object Mapping on page 8-8.

To list the ODBC/SQL Server and NonStop SQL/MP object names, select the columns T_OBJNAME and N_OBJNAME from ZNUOBJ, as shown in Example 7-2.

Example 7-2. ODBC/SQL Server Object Names Generated by USERCAT INSTALL

```
>>select t_objname, n_objname
+>from \tess.$noss.persnl.znuobj
+>order by n_objname;

T_OBJNAME                       N_OBJNAME
------------------------------  ----------------------------------

BASETABS                        \TESS.$NOSS.PERSNL.BASETABS
COLUMNS                         \TESS.$NOSS.PERSNL.COLUMNS
COMMENTS                        \TESS.$NOSS.PERSNL.COMMENTS
CONSTRNT                        \TESS.$NOSS.PERSNL.CONSTRNT
DEPT                            \TESS.$NOSS.PERSNL.DEPT
DEPTBAK                         \TESS.$NOSS.PERSNL.DEPTBAK
EMPLBAK                         \TESS.$NOSS.PERSNL.EMPLBAK
EMPLOYEE                        \TESS.$NOSS.PERSNL.EMPLOYEE
FILES                           \TESS.$NOSS.PERSNL.FILES
INDEXES                         \TESS.$NOSS.PERSNL.INDEXES
JOB                             \TESS.$NOSS.PERSNL.JOB
JOBBAK                          \TESS.$NOSS.PERSNL.JOBBAK
KEYS                            \TESS.$NOSS.PERSNL.KEYS
PARTNS                          \TESS.$NOSS.PERSNL.PARTNS
PROGRAMS                        \TESS.$NOSS.PERSNL.PROGRAMS
TABLES                          \TESS.$NOSS.PERSNL.TABLES
TRANSIDS                        \TESS.$NOSS.PERSNL.TRANSIDS
USAGES                          \TESS.$NOSS.PERSNL.USAGES
VERSIONS                        \TESS.$NOSS.PERSNL.VERSIONS
VIEWS                           \TESS.$NOSS.PERSNL.VIEWS
ZNUIX                           \TESS.$NOSS.PERSNL.ZNUIX
ZNUOBJ                          \TESS.$NOSS.PERSNL.ZNUOBJ
SYSCOLUMNS                      \TESS.$NOSS.PERSNL.ZVUCOL
SYSTYPES                        \TESS.$NOSS.PERSNL.ZVUDT
SYSINDEXES                      \TESS.$NOSS.PERSNL.ZVUIX
SYSOBJECTS                      \TESS.$NOSS.PERSNL.ZVUOBJ
SYSPROTECTS                     \TESS.$NOSS.PERSNL.ZVUPROT
SYSUSERS                        \TESS.$NOSS.PERSNL.ZVUUS
T_EMPLOYEE567490                \TESS.$NOSS.PERSNX.EMPLOYEE
T1                              \TESS.$NOSS.PERSNX.T1
T_EMPLOYEE724838                \TESS.$NOSS.PERSNY.EMPLOYEE
T_T1809120                      \TESS.$NOSS.PERSNY.T1
T_EMPLOYEE891245                \TESS.$NOSS.PERSNZ.EMPLOYEE
T_T1961851                      \TESS.$NOSS.PERSNZ.T1

--- 34 row(s) selected.
```

Changing ODBC/SQL Server Object Names

The recommended method for changing an ODBC/SQL Server object name is to execute a REMOVE TABLE, REMOVE INDEX, or REMOVE VIEW statement for the name you want to change, then execute an ADD TABLE, ADD INDEX, or ADD VIEW statement for the new name. You can execute these mapping table statements from either the NOSCOM command interpreter or the NonStop ODBC Server Configuration Manager graphical user interface tool.

You can also change the ODBC/SQL Server object names by using INSERT and DELETE statements on ZNUOBJ. In this scenario, you cannot use UPDATE statements because the column you need to update, T_OBJNAME, is part of the primary key (T_UNAME and T_OBJNAME). Although using INSERT and DELETE is not the recommended method, it is shown in Example 7-3 for illustrative purposes.

To change an ODBC/SQL Server object name using INSERT and DELETE statements, list all columns of the record containing that name and insert a new record that is identical except for the ODBC/SQL Server name, which is column T_OBJNAME.

Example 7-3 shows NonStop SQL/MP statements that list all columns of ZNUOBJ for the tables mapped in Example 7-1.

Example 7-3. Listing All Columns of ZNUOBJ

```
>>select * from $noss.persnl.znuobj
+>where n_objname like '%EMPLOYEE%' or n_objname like '%T1%'
+>order by n_objname;

T_UID   T_OBJNAME                          T_UNAME                       T_OBJTYPE
------  ------------------------------   ----------------------------  ----------
T_OBJID      N_OBJNAME                          N_OBJTYPE  N_ORIGIN
-----------  ---------------------------------  ---------  --------
N_SESSIONID
--------------------

44812   EMPLOYEE                           SQL_LYNNR                    U
      37404  \TESS.$NOSS.PERSNL.EMPLOYEE          TA         N
                   0
44812   T_EMPLOYEE567490                   SQL_LYNNR                    U
      35906  \TESS.$NOSS.PERSNX.EMPLOYEE          TA         N
                   0
44812   T1                                 SQL_LYNNR                    U
        973  \TESS.$NOSS.PERSNX.T1                TA         N
                   0
44812   T_EMPLOYEE724838                   SQL_LYNNR                    U
      55532  \TESS.$NOSS.PERSNY.EMPLOYEE          TA         N
                   0
44812   T_T1809120                         SQL_LYNNR                    U
      46978  \TESS.$NOSS.PERSNY.T1                TA         N
                   0
44812   T_EMPLOYEE891245                   SQL_LYNNR                    U
      56069  \TESS.$NOSS.PERSNZ.EMPLOYEE          TA         N
                   0
44812   T_T1961851                         SQL_LYNNR                    U
      23895  \TESS.$NOSS.PERSNZ.T1                TA         N
                   0

--- 7 row(s) selected.
```

Example 7-4 shows NonStop SQL/MP statements that assign new names for the five tables that were mapped in Example 7-1.

Example 7-4. Changing the ODBC/SQL Server Object Names in ZNUOBJ

```
>>delete from persnl.znuobj where
+>t_objname like '%T_EMP%' or
+>t_objname like '%T_T1%';
--- 5 row(s) deleted.
>>
>>insert into persnl.znuobj (*) values
+>(44812, 'T_EMPL178234', 'SQL_LYNNR', 'U', 35906,
+>'\TESS.$NOSS.PERSNX.EMPLOYEE', 'TA', 'N', 0);
--- 1 row(s) inserted.
>>
>>insert into persnl.znuobj (*) values
+>(44812, 'T_EMPL246802', 'SQL_LYNNR', 'U', 55532,
+>'\TESS.$NOSS.PERSNY.EMPLOYEE', 'TA', 'N', 0);
--- 1 row(s) inserted.
>>
>>insert into persnl.znuobj (*) values
+>(44812, 'T_EMPL435790', 'SQL_LYNNR', 'U', 56069,
+>'\TESS.$NOSS.PERSNZ.EMPLOYEE', 'TA', 'N', 0);
--- 1 row(s) inserted.
>>
>>insert into persnl.znuobj (*) values
+>(44812, 'T1_Y', 'SQL_LYNNR', 'U', 46978,
+>'\TESS.$NOSS.PERSNY.T1', 'TA', 'N', 0);
--- 1 row(s) inserted.
>>
>>insert into persnl.znuobj (*) values
+>(44812, 'T1_Z', 'SQL_LYNNR', 'U', 23895,
+>'\TESS.$NOSS.PERSNZ.T1', 'TA', 'N', 0);
--- 1 row(s) inserted.
>>
>>select t_objname, n_objname from persnl.znuobj where
+>n_objname like '%EMPLOYEE%'
+>order by n_objname;

T_OBJNAME                        N_OBJNAME
------------------------------   ----------------------------------

EMPLOYEE                         \TESS.$NOSS.PERSNL.EMPLOYEE
T_EMPL178234                     \TESS.$NOSS.PERSNX.EMPLOYEE
T_EMPL246802                     \TESS.$NOSS.PERSNY.EMPLOYEE
T_EMPL435790                     \TESS.$NOSS.PERSNZ.EMPLOYEE

--- 4 row(s) selected.>>
>>select t_objname, n_objname from persnl.znuobj where
+>n_objname like '%T1%' order by n_objname;

T_OBJNAME                        N_OBJNAME
------------------------------   ----------------------------------

T1                               \TESS.$NOSS.PERSNX.T1
T1_Y                             \TESS.$NOSS.PERSNY.T1
T1_Z                             \TESS.$NOSS.PERSNZ.T1

--- 3 row(s) selected.
```

Enter the name in uppercase letters. When you assign the new ODBC/SQL Server object names, follow these guidelines:

- Check that the name is a valid ODBC/SQL Server identifier.
- Check that the name is unique for the database and owner.
- Check that you change only the ODBC/SQL Server object name (T_OBJNAME).

Running the Catalog Utility Statements

The NonStop ODBC Server provides utilities for managing the NonStop ODBC Server mapping tables. Access the utility statements in one of three ways:

- From a TACL prompt by running the program NOSCOM
- From the NonStop ODBC Server Configuration Manager, a graphical user interface tool
- By using pass-through mode in the NonStop ODBC Server

Most catalog utility statements can be executed by using the NonStop ODBC Server, but some statements must be executed from the NonStop ODBC Server Configuration Manager or a NOSCOM prompt.

Table 7-3. Catalog Utility Statements

Statement	Description	Execute By Using NonStop ODBC Server
SYSCAT INSTALL	Customizes the NonStop SQL/MP system catalog.	–
USERCAT INSTALL	Customizes a NonStop SQL/MP user catalog.	x
SYSCAT DEINSTALL	Decustomizes the NonStop SQL/MP system catalog.	–
USERCAT DEINSTALL	Decustomizes a NonStop SQL/MP user catalog.	x
SYSCAT REFRESH and USERCAT REFRESH	Add NonStop SQL/MP objects to the mapping tables, remove unresolved object references, re-create missing mapping tables and mapping table items.	x
SYSCAT VALIDATE and USERCAT VALIDATE	Report all referential inconsistencies with the mapping tables.	x
CLEANUP	Drops temporary objects and removes them from the mapping tables.	x
SYSCAT UPGRADE and USERCAT UPGRADE	Upgrade catalogs to the current version.	x
Mapping table statements listed in Table 7-4	Create, modify, and delete items in the mapping tables.	x
?	Displays the syntax of the utility statements.	–

x Indicates the statement can be executed using the NonStop ODBC Server
– Indicates the statement cannot be executed using the NonStop ODBC Server

The catalog utility statements are described in the following subsections, but to use them you need the information presented in this subsection.

Executing the Statements From a TACL Prompt

You execute the statements by running the NOSCOM program from the TACL prompt. The general syntax of NOSCOM is:

```
NOSCOM statement
```

`statement`

specifies a catalog utility statement listed in Table 7-3. Specify object names in Guardian format. The syntax of the individual statements is described later in this section.

Example

The following statement executes the USERCAT INSTALL statement:

```
NOSCOM USERCAT INSTALL \test.$vol2.persnl
```

Other NOSCOM Commands

NOSCOM is a command-line interpreter from which you can execute NonStop ODBC Server catalog utility statements. Other commands supported by NOSCOM are modeled after commonly used TACL commands. For more information about these commands, see the *TACL Reference Manual*.

EXIT

The EXIT command terminates your NOSCOM session.   QUIT and EXIT are equivalent.

FC

The FC command enters you into FC input mode, marked by the >>> prompt.   In FC input mode, you can re-display, modify and then re-execute a previous command. If you type only FC at a NOSCOM prompt, NOSCOM displays the last command executed, followed by the >>> prompt. If you type FC followed by an alphabetic character, NOSCOM displays the most recent command of the last ten executed that begins with the alphabetic character. If you use the HISTORY command to identify a previously executed command by number, you can type FC *`command-number`* to modify that command.

While in FC input mode, the characters “i”, “d”, and “r” have special significance. To insert text at a particular point in the command string, enter the character “i” on the input line at that point, immediately followed by the text to insert. To delete one or more characters, enter “d” on the input line beneath each character to delete. To replace text, enter “r” on the input line at that point, immediately followed by the text to substitute. Hit return to display the modified command string. Hit return again to execute the modified command. To exit FC input mode, enter only “//” on the input line.

HELP

The HELP command displays help text for NOSCOM commands. Execute HELP to view help text for a specific catalog utility statement, view a list of a particular type of catalog utility statement, or view a list of all NOSCOM commands.

To get help for a specific command, type HELP *command*. For example, HELP ADD USER displays help text for the ADD USER catalog utility statement. To list all ADD statements, type HELP ADD. To list all NOSCOM commands, type any of the following: HELP, HELP ALL, or HELP *.

HISTORY

The HISTORY command displays and identifies by number the ten most recent NOSCOM commands executed. You can refer to a command by its number when you execute the FC command in order to re-execute a command other than the most recent.

INFO

The INFO command displays attribute values associated with a particular NonStop ODBC Server entity. The INFO command syntax for configuration components is as follows:

INFO ALIAS *alias-name*

INFO CONTROL *control-name*

INFO DEFINE *define-name*
[FOR *scs-name*]

INFO GOVERNING *gov-name*

INFO INDEX *nssql-filename*
[FROM *database*]

INFO NET_SERVICE *scs-name*

INFO NSODBC_OBJECT *object-name*
[FROM *database*]

INFO PROFILE *profile-name*

INFO PROCEDURE *procedure-name*
[FROM *database*]

INFO PROCEDURE_COLUMNS
procedure-name

INFO SCFG *scfg-name*

INFO SCS *scs-processname*

INFO SERVERCLASS *server-name*

INFO SMAP *scs-name*
server-name

INFO TABLE *nssql-filename*
[FROM *database*]

INFO TRACE *trace-name*

INFO UMAP *scs-name*
alias-user-name

INFO USER *logical-username*

INFO VIEW *nssql-filename*
[FROM *database*]

The wild-card characters * (represents one or more characters) and ? (represents only one character) are valid for these commands.

The following INFO command displays all tables in a single ODBC catalog on all volumes and all subvolumes:

```
INFO TABLE $*.*.*
 [ FROM odbc-catalog]
```

If the wild card pattern is not provided, the current volume and subvolume values are used. If the ODBC catalog name is not provided, the default is the current volume and subvolume.

The following INFO commands display all server classes associated with an SCS named configuration entity:

```
INFO CONFIG scs-name
INFO MAPNAME scs-name  USER user-name
```

Wild-card characters are not valid for INFO CONFIG and INFO MAPNAME.

LOG

The LOG command captures NOSCOM command input and output to a file. To capture input and output, type LOG TO *filename*. If *filename* does not exist, NOSCOM automatically creates an EDIT file. If *filename* already exists, NOSCOM appends the logged information to the end of the file. LOG logs command input and output until either you execute LOG STOP or terminate the NOSCOM session.

OBEY

The OBEY command executes a NOSCOM command file. A command file is an EDIT file that contains a sequence of NOSCOM commands to execute. To execute a NOSCOM command file, type OBEY *filename* at a NOSCOM prompt. NOSCOM executes the commands until an error occurs or the last command executes.

To include comments in a NOSCOM command file, precede the comment text with two dashes (--); remaining characters on the line are ignored. You can execute another command file from within a command file; NOSCOM supports up to five layers of OBEY commands embedded in command files.

QUIT

The QUIT command terminates your NOSCOM session. EXIT and QUIT are equivalent.

Running the Statements Using Pass-Through Mode

You can run most of the utility statements using pass-through mode in the NonStop ODBC Server. The general syntax of a pass-through statement is:

```
SELECT "TDM: UTIL catalog-utility-statement"
```

`TDM:`

is a required keyword that indicates this is a pass-through statement.

`UTIL`

is a required keyword that indicates this is a catalog utility statement.

`catalog-utility-statement`

specifies one of the catalog utility statements listed in Table 7-3. The syntax of the individual statements is described under Customizing Catalogs.

Specify object names using the Guardian format.

For details about pass-through mode, see Section 6, Using Pass-Through Mode.

Example

The following statement executes the USERCAT REFRESH statement:

```
select "tdm: util usercat refresh \test.$vol1.persnl"
```

Privileged Users

Originally, the access rights to perform operations in the NonStop ODBC Server were based on the user’s login identity. Installation of the system catalog required the user’s mapped identity to be that of the privileged ID or a privileged user. In addition, any command that affected more than one user, such as ADD USER for a user not yourself, required the privileged ID or privileged user identity.

Subsequent releases introduced the Guardian mechanism to set the PROGID attribute of the NOSUTIL server. This led to a configuration in which the NOSUTIL server could be designated as privileged and then use Guardian file system or Safeguard access control to govern which users could use this privileged-set NOSUTIL process to perform privileged operations.

The NOSUTIL configuration statements require the user to be privileged; otherwise, an error message of the form “User does not have sufficient privileges to execute command” is returned.

While the NonStop ODBC Server provides for the definition of user identities, this is limited to mapping the login user identity to the identifier used for login authentication and object reference. The NonStop ODBC Server does not determine which login user identities are privileged by name, but relies on the Guardian and optional Safeguard authentication mechanisms. The password required by the NonStop ODBC Server in the login message must be the correct password for the mapped user identifier determined during login message processing (by lookup or algorithm). There is no way to switch user identity once the login processing is completed.

Additionally, other processes might return the same or a similar error message that the user does not have sufficient privileges to complete the work. These exceptions are reported as file-system errors 48 and 49. It is possible to set the security access list on base tables so that even a privileged user might not be able to complete a configuration statement.

Customizing Catalogs

The SYSCAT INSTALL and USERCAT INSTALL statements customize the system catalog and user catalogs, respectively

SYSCAT INSTALL Statement

The SYSCAT INSTALL statement customizes a NonStop SQL/MP system catalog at the location specified by the =_NSODBC_SYSTEM_CATALOG DEFINE. If this DEFINE is not present, SYSCAT INSTALL customizes the NonStop SQL/MP system catalog.

You can execute SYSCAT INSTALL only from a TACL prompt (not as a pass-through command), and you must be a privileged user.

SYSCAT INSTALL has the following syntax:

```
SYSCAT INSTALL [ SECURE security-string ]
               [ LOG output-filename ]
```

security-string

is a four-character alphabetic string that specifies the Guardian read, write, execute, and purge (RWEP) access for the system catalog. For more information about security for NonStop SQL/MP objects, see the "Security" entry in the *NonStop SQL/MP Reference Manual*.

output-filename

specifies the NonStop SQL/MP table name of the log table where command status messages are written. The log table is an unaudited, entry sequenced table that NOSUTIL creates and registers in the same subvolume as the NonStop SQL/MP system catalog. NOSUTIL does not create a mapping entry for the log table, because the NonStop SQL/MP system catalog is not necessarily customized by NonStop ODBC.

If the table already exists, NOSUTIL returns an error and SYSCAT INSTALL terminates. If you omit this option, NOSUTIL returns all status messages to the home terminal.

Example

The following statement customizes the NonStop SQL/MP system catalog in the $SYSYEM.SQL subvolume.

```
ADD DEFINE =_NSODBC_SYSTEM_CATALOG, CLASS CATALOG, SUBVOL
$system.sql
```

```
SYSCAT INSTALL SECURE NUNU LOG log02
```

Effects of SYSCAT INSTALL

When you run SYSCAT INSTALL, the NonStop ODBC Server creates the following objects in the subvolume of your system catalog:

ZNSALT	ZNSGOV	ZNSTRA	ZNUOBJI3	ZVUOBJ
ZNSALTI1	ZNSMSG	ZNSUMAP	ZNUPCLI1	ZVUOCOL
ZNSCON	ZNSNET	ZNSUS	ZNUPCOL	ZVUPCOL
ZNSCONI1	ZNSPROF	ZNUDT	ZNUPROC	ZVUPROT
ZNSDB	ZNSPROT	ZNUIXI1	ZVSDB	ZVUUS
ZNSDBI1	ZNSSCFG	ZNUMTRX	ZVSMSG	
ZNSDBI2	ZNSSCS	ZNUOBJ	ZVUCOL	
ZNSDEF	ZNSSER	ZNUOBJI1	ZVUDT	
ZNSDUMMY	ZNSSMAP	ZNUOBJI2	ZVUIX	

These tables, indexes, and views are called the NonStop ODBC Server mapping tables. Do not purge these objects; they are the mapping tables for your system. For a description of the objects, see Section 8, HP NonStop ODBC Server Mapping Tables.

SYSCAT INSTALL also does the following:

- Builds an SQL Server MASTER database representation with NonStop ODBC Server extensions for the NonStop SQL/MP system catalog.
- Inserts a new row into the ZNSDB mapping table to associate the logical database name “MASTER” with the customized NonStop SQL/MP system catalog.
- Runs under a separate TMF transaction. The NOSUTIL program automatically commits the transaction after customizing the system catalog.
- Generates an error if executed as a pass-through command.
- Generates an error if the system catalog is already customized for the NonStop ODBC Server.
- Generates an error if you are not a privileged user.

If SYSCAT INSTALL fails for any reason, all updates to the system catalog are automatically rolled back. If the system catalog was a valid NonStop SQL/MP system catalog, it is still valid after roll back.

USERCAT INSTALL Statement

The USERCAT INSTALL statement customizes a NonStop SQL/MP user catalog. To use the NonStop ODBC Server to access a catalog created with NonStop SQL/MP, you must first customize the catalog.

Note. If you create the catalog through the NonStop ODBC Server using the CREATE DATABASE statement, you do not need to customize the catalog. For information about the CREATE DATABASE statement, see Section 4, Transact-SQL Language.

USERCAT INSTALL has the following syntax:

```
USERCAT INSTALL [ [catalog-name ] [ SECURE security-string ]
                 [ LOG output-filename ]  [EMPTY]  ]
```

catalog-name

is the name of the subvolume containing the NonStop SQL/MP catalog to customize. The *catalog-name* is in Guardian format and can include the node and volume. If you omit *catalog-name*, the default node, volume, and subvolume for the current user is used.

If the specified subvolume already contains a customized catalog, an error message is generated and the statement does not execute.

If the specified subvolume does not contain a catalog, USERCAT INSTALL creates a new catalog and customizes it.

security-string

is a four-character alphabetic string that specifies the Guardian read, write, execute, and purge (RWEP) access for the user catalog. For more information about security for NonStop SQL/MP objects, see the “Security” entry in the *NonStop SQL/MP Reference Manual*.

If you specify *security-string*, you must also explicitly specify *catalog-name*. If you omit *security-string*, catalog security is set to the default security for the current user profile, if defined. If default security is not defined in the current profile, the default security is the system DEFAULT_SECURITY value found in the ZNSSCFG mapping table.

output-filename

specifies the NonStop SQL/MP table name of the log table where command response messages are written. The log table is an unaudited, entry-sequenced table that NOSUTIL creates in the catalog where VALIDATE, REFRESH, or INSTALL is being run. The log table also creates a mapping entry in the NonStop ODBC master catalog. If a table that has the same name as *output-filename* already exists, NOSUTIL appends the command response messages to the log table.

`EMPTY`

installs the user catalog and populates the mapping tables without installing the entries for the user-created NonStop SQL/MP objects in the current catalog.

Example

The following statement customizes the NonStop SQL/MP catalog named \TEST.$VOL2.PERSNL:

```
USERCAT INSTALL \test.$vol2.persnl
```

Effects of USERCAT INSTALL

USERCAT INSTALL does the following:

- Creates the following objects in your subvolume:

ZNUDT	ZNUOBJI1	ZNUPCOL	ZVUIX	ZVUPROT
ZNUIX	ZNUOBJI2	ZNUPROC	ZVUOBJ	ZVUUS
ZNUIXI1	ZNUOBJI3	ZVUCOL	ZVUOCOL	
ZNUOBJ	ZNUPCLI1	ZVUDT	ZVUPCOL	

  These objects are the NonStop ODBC Server mapping tables. Do not purge these objects; they contain the mapping information for your catalog. For a description of the objects, see Section 8, HP NonStop ODBC Server Mapping Tables.

- If the catalog is a NonStop SQL/MP catalog not customized for the NonStop ODBC Server, populates the mapping tables with one entry for each NonStop SQL/MP object in the current catalog.
- If the `EMPTY` option is specified while installing the catalog, does not populate the `ZNUOBJ` or `ZNUIX` system tables for user-created NonStop SQL/MP objects.
- To indicate that it is customized, adds your catalog to SYSDATABASES, and adds a row to ZNSDB for the catalog and database.
- If the owner of *`catalog-name`* has no corresponding entry in ZNSUS, creates an entry. The logical username in this entry is the default value for the current group and user. No values for profile, server class, or trace are generated.
- If you are not authorized to create an SQL catalog in the specified subvolume, or if you are not authorized to write to the underlying NonStop SQL/MP catalog, generates an error.
- If you are not authorized to write to ZNSDB, generates an error.
- If the catalog is already customized for an earlier version of the NonStop ODBC Server, generates an error. You must then either deinstall the catalog and reinstall, or upgrade it to the latest version.

If an error occurs during USERCAT INSTALL, then one of the following occurs:

- If the original catalog was a user catalog customized for the NonStop ODBC Server, then after USERCAT REFRESH fails, the original NonStop ODBC Server customized user catalog is restored.
- If the original catalog was a NonStop SQL/MP user catalog, then after USERCAT INSTALL fails, the NonStop SQL/MP user catalog is restored.
- If there was no original NonStop SQL/MP user catalog *catalog-name*, then no NonStop SQL/MP user catalog remains after USERCAT INSTALL fails.

How Object Names Are Assigned

When you customize or refresh a system catalog or user catalog, the NonStop ODBC Server assigns ODBC/SQL Server names to the objects in the catalog. The NonStop ODBC Server usually assigns the Guardian file name. For example, a NonStop SQL/MP table named \TESS.$NOSS.PERSNL.EMPLOYEE is given the ODBC/SQL Server name EMPLOYEE. Sometimes, however, the NonStop ODBC Server must assign a name other than the Guardian file name.

For information about viewing and changing the mapped names, see Renaming ODBC or SQL Server Objects on page 7-7.

How Owner Names Are Assigned

The owner name for the objects is the same as the owner of the corresponding Guardian file. The owner name of the mapping tables is the user who customizes the catalog.

Decustomizing Catalogs

The following describes SYSCAT DEINSTALL and USERCAT DEINSTALL, which decustomize the system catalog and user catalogs, respectively.

SYSCAT DEINSTALL Statement

Use the SYSCAT DEINSTALL statement to decustomize the system catalog. SYSCAT DEINSTALL drops all of the mapping tables from the system catalog and, optionally, drops all objects created using the NonStop ODBC Server. You must be a privileged user to run SYSCAT DEINSTALL.

When you no longer need the NonStop ODBC Server installed on your system, you decustomize the system catalog. When you decustomize the system catalog, all NonStop ODBC Server mapping tables are purged from the subvolume containing the NonStop SQL/MP system catalog.

Before decustomizing the system catalog, you must decustomize every customized catalog by using USERCAT DEINSTALL.

You can run SYSCAT DEINSTALL only from a NOSCOM prompt; it cannot be run as a NonStop ODBC Server pass-through command.

SYSCAT DEINSTALL has the following syntax:

```
SYSCAT DEINSTALL [ RESTRICT | CASCADE ]
```

`RESTRICT`

empties the catalog mapping tables before the tables are dropped. This option is the default. SYSCAT DEINSTALL with the RESTRICT option succeeds only if the NonStop ODBC Server mapping table ZNSDB contains the entry for MASTER, and no other information; and if entries in ZNUOBJ represent only system objects.

`CASCADE`

drops all NonStop ODBC Server objects referenced in the system catalog mapping tables before dropping the mapping tables. A NonStop ODBC Server object is an object that was created through the NonStop ODBC Server using CORE SQL or Transact-SQL statements.

If you omit CASCADE, all objects are retained.

Example

The following statement decustomizes the NonStop SQL/MP system catalog:

```
SYSCAT DEINSTALL
```

Effects of SYSCAT DEINSTALL

SYSCAT DEINSTALL does the following:

- Drops all of the mapping tables in the subvolume of the system catalog.
- Drops objects created using the NonStop ODBC Server, if you specify CASCADE.
- Runs under a separate TMF transaction. The NOSUTIL program automatically commits the transaction after decustomizing the system catalog.
- Preserves the actual NonStop SQL/MP system catalog. You must use the NonStop SQL/MP Conversational Interface to drop the system catalog.
- Generates an error if you are not a privileged user.
- Generates an error if the system catalog is not customized for the NonStop ODBC Server. However, you can deinstall a system catalog customized by an earlier version of the NonStop ODBC Server.
- Because of the TMF response time, it might take a few minutes before the catalog tables are released from the locks. Therefore, if you perform an INSTALL immediately after a DEINSTALL, a failure might occur because of these locks. If this happens, wait a few minutes and then retry the INSTALL.

If SYSCAT DEINSTALL succeeds, you must run SYSCAT INSTALL before the system catalog can be run again against the NonStop ODBC Server. If SYSCAT DEINSTALL fails for any reason, the NonStop ODBC Server customized system catalog is unchanged.

If Catalogs Are Still Customized

If your system contains a customized user catalog, you will receive an error message, and the SYSCAT DEINSTALL statement will not be executed. You must decustomize all user catalogs before decustomizing the system catalog.

USERCAT DEINSTALL Statement

Use the USERCAT DEINSTALL statement to decustomize a NonStop SQL/MP catalog.

When you no longer need to access a catalog using the NonStop ODBC Server, you can decustomize it. When you decustomize a catalog, the NonStop ODBC Server mapping tables and views are erased, and the catalog is no longer available through the NonStop ODBC Server. When you decustomize a catalog, you can also drop objects created using the NonStop ODBC Server, drop the NonStop SQL/MP catalog, or both.

USERCAT DEINSTALL has the following syntax:

```
USERCAT DEINSTALL [catalog-name
                  [ RESTRICT | CASCADE | CASCADE_NSSQL ]]
```

catalog-name

specifies the name of the subvolume containing the NonStop SQL/MP catalog to be decustomized. The *catalog-name* appears in Guardian format and can include the node and volume.

If the specified subvolume does not contain a customized catalog, USERCAT DEINSTALL generates warning messages that the mapping tables could not be found.

If you omit *catalog-name*, the current subvolume is used.

RESTRICT

causes the catalog mapping tables to be dropped if, and only if, there are no user objects created by the NonStop ODBC Server in the catalog. The underlying NonStop SQL/MP table is unaffected. If you specify RESTRICT, you must also specify *catalog-name*.

RESTRICT is the default.

`CASCADE`

drops all NonStop ODBC Server objects registered in *`catalog-name`* before dropping the mapping tables. A NonStop ODBC Server object is an object created through the NonStop ODBC Server using CORE SQL or Transact-SQL statements. The underlying NonStop SQL/MP table is unaffected.

If you specify CASCADE, you must also specify *`catalog-name`*.

`CASCADE_NSSQL`

drops both the NonStop ODBC Server objects and the NonStop SQL/MP catalog, if there are no NonStop SQL/MP objects in the catalog after removing the NonStop ODBC Server objects. If you specify CASCADE_NSSQL, you must specify *`catalog-name`*.

The catalog must be empty of all tables in order to drop it.

Examples

The following examples show uses of USERCAT DEINSTALL.

Do Not Drop Any Objects

The following statement decustomizes the NonStop SQL/MP catalog \TEST.$VOL2.PERSNL:

```
usercat deinstall \test.$vol2.persnl restrict
```

If any NonStop ODBC Server objects exist, the catalog is not deinstalled; if there are no NonStop ODBC Server objects, the mapping tables are dropped.

Drop Objects Created By Using the NonStop ODBC Server

The following statement decustomizes the NonStop SQL/MP catalog \TEST.$VOL2.SALES:

```
usercat deinstall \test.$vol2.sales
```

Use this statement, in which the CASCADE keyword is assumed by default, to drop the NonStop ODBC Server mapping tables and all objects that were created by using the NonStop ODBC Server. Objects created by using NonStop SQL/MP remain in the catalog.

Drop the NonStop SQL/MP Catalog and Objects Created With the NonStop ODBC Server

The following decustomizes the NonStop SQL/MP catalog \TEST.$VOL2.INVENT:

```
usercat deinstall \test.$vol2.invent cascade_nssql
```

Use this statement to drop the NonStop ODBC Server mapping tables, all objects that were created by using the NonStop ODBC Server, and the NonStop SQL/MP catalog.

If the catalog contains objects created by using NonStop SQL/MP, the mapping tables and NonStop ODBC Server objects are dropped, but the catalog is not dropped. A warning is displayed indicating that the catalog is not empty and was not dropped.

Effects of USERCAT DEINSTALL

USERCAT DEINSTALL does the following:

- Drops all the NonStop ODBC Server mapping tables from your catalog subvolume.
- Removes your catalog entry from the customized system catalog (which corresponds to SYSDATABASES).
- Drops all objects created using the NonStop ODBC Server (if you specify CASCADE or CASCADE_NSSQL).
- Removes all entries in ZNUOBJ that refer to remaining NonStop ODBC Server objects. If you later execute a USERCAT INSTALL command, object names longer than 8 characters will be truncated.
- Drops the NonStop SQL/MP catalog (if you specify CASCADE_NSSQL and the catalog is empty).
- Generates an error if the user catalog is not customized for the NonStop ODBC Server.The action described is conditional, succeeding if the user has the privilege to add a user. It will fail to complete if the user is not privileged and one or more objects have owners not currently registered in the system catalog.
- The action described is conditional, succeeding if the user has the privileges to add a user. It will fail to complete if the user is not privileged and one or more objects have owners not currently registered in the system catalog.

If USERCAT DEINSTALL fails with CASCADE specified, either explicitly or by default, the user catalog is unchanged.

Dropping Tables With Dependent Views

If you specify the CASCADE or CASCADE NSSQL keywords, USERCAT DEINSTALL drops objects in the order in which they are listed in the mapping tables. If a view is dependent on a table and the table is listed in the mapping table before the view, USERCAT DEINSTALL drops the table before dropping the dependent view. When dropping the table, NonStop SQL/MP also drops the dependent view; however, the view remains listed in the mapping tables.

When USERCAT DEINSTALL encounters the view in the mapping table, it attempts to drop it, and the following messages are generated:

```
NOSUTIL warning: DROP VIEW view-name failed. Sqlcode = -4059
NOSUTIL error: File system error 11
```

You can ignore these messages because the view has already been dropped.

Maintaining Catalogs

The following statements are used to maintain catalogs:

- SYSCAT REFRESH—ensures that objects in the system catalog and corresponding entries in the mapping tables balance.
- USERCAT REFRESH—ensures that objects in user catalogs and corresponding entries in the mapping tables balance.
- SYSCAT VALIDATE—reports inconsistencies in references between the system catalog and the mapping tables.
- USERCAT VALIDATE—reports inconsistencies in references between the specified user catalog and the mapping tables.
- CLEANUP—drops temporary objects from catalogs.
- SYSCAT UPGRADE—upgrades the system catalog to the current version.
- USERCAT UPGRADE—upgrades a user catalog to the current version.

SYSCAT REFRESH Statement

Use the SYSCAT REFRESH statement to add objects registered in the system catalog to the configuration tables, system mapping tables, and user mapping tables. When you run SYSCAT REFRESH, all objects registered in the system catalog that are not mapped become mapped and therefore are available through the NonStop ODBC Server.

You must run SYSCAT REFRESH if objects registered in the catalog are not listed in the mapping tables. An object does not appear in the mapping tables if any of the following are true:

- You create the object in pass-through mode.
- You create the object using NonStop SQL/MP when the catalog is already customized.
- You delete a row from the NonStop ODBC Server mapping table ZNUOBJ.

You must be a privileged user to use SYSCAT REFRESH.

SYSCAT REFRESH has the following syntax:

```
SYSCAT REFRESH [ LOG output-filename ]
```

output-filename

specifies the NonStop SQL/MP table name of the log table where command response messages are written. The log table is an unaudited, entry sequenced table that NOSUTIL creates in the NonStop ODBC master catalog subvolume. NOSUTIL also registers the log table in the NonStop ODBC master catalog and creates a mapping entry. If a table with the same name as *`output-filename`* already exists, NOSUTIL returns an error and the SYSCAT REFRESH statement terminates.

Example

The following statement refreshes the NonStop SQL/MP system catalog:

```
syscat refresh
```

Use this statement to verify that all objects registered in the system catalog are listed in the mapping tables. Objects not listed are added to the mapping tables. For example, if the ZNUOBJ table is missing, SYSCAT REFRESH re-creates it and repopulates it.

Effects of SYSCAT REFRESH

SYSCAT REFRESH checks whether each table, view, and index in the system catalog has an entry in the mapping tables. If the object is not entered in the mapping tables, SYSCAT REFRESH enters it.

If a mapping entry references a nonexistent NonStop SQL/MP object, SYSCAT REFRESH deletes the entry.

If a mapping table is not found, SYSCAT REFRESH re-creates it and resupplies it with values. SYSCAT REFRESH can re-create all system tables except ZNSDB and ZNSUS. If ZNSDB is dropped, you must either restore it from a backup image of the NonStop SQL/MP system catalog or else purge all Zzzzzs tables and views and then execute SYSCAT INSTALL. If ZNSUS is dropped, you must also either drop ZNSALT or remove the DBO entry from ZNSALT and then execute SYSCAT REFRESH again.

SYSCAT REFRESH also does the following:

- Deletes references in ZNSDB to any database without a NonStop SQL/MP system catalog.
- Deletes and reloads all values in the static mapping table ZNUDT.
- Re-creates mapping entries in ZNUOBJ for objects found in the NonStop SQL/MP INDEXES, TABLES, and VIEWS catalog tables that are missing from the NonStop ODBC Server mapping tables.
- Adds a username mapping entry in ZNSUS for each owner of an object in the system catalog that is not already a mapped username.
- Deletes mapping entries in ZNOBJ for objects listed in either ZNOBJ or ZNUIX, but not found in NonStop SQL/MP catalog tables.
- Generates an error if you are not a privileged user.
- Generates an error if the system catalog is not customized for the NonStop ODBC Server.

If An Object Is Partitioned

For each object with multiple partitions, SYSCAT REFRESH adds only one entry to the ZNUOBJ mapping table. Even if primary and secondary partitions of the same object are registered in different catalogs, ZNUOBJ has only one entry for the object. NonStop ODBC Server determines which partition to map as follows:

- If the primary partition is located in the same subvolume as the specified catalog**,** the primary partition is the mapped object.
- If the primary partition is not located in the same subvolume as the specified catalog, the first partitioned object in the list of objects in the catalog (TABLES, INDEXES) is the mapped object.
- If the object has no local partitions, the primary partition is the mapped object.

How ODBC/SQL Server Object Names Are Assigned

When you customize or refresh the system catalog, the NonStop ODBC Server assigns ODBC/SQL Server names to the objects in the catalog. The NonStop ODBC Server usually assigns the Guardian file name. For example, a NonStop SQL/MP table named \TESS.$NOSS.PERSNL.EMPLOYEE is given the ODBC/SQL Server name EMPLOYEE. Sometimes, however, the NonStop ODBC Server must assign a name other than the Guardian file name.

For information about viewing and changing the mapped names, see Renaming ODBC or SQL Server Objects on page 7-7.

Note. SYSCAT REFRESH truncates object names that are more than 8 characters long. To preserve a longer object name, manually add the object to ZNUOBJ by executing ADD TABLE, INDEX, or VIEW.

How SYSCAT REFRESH Affects Stored Procedures

Stored procedures are not necessarily SQL objects. If a stored procedure becomes invalid or is deleted from ZNUPROC and ZNUPCOL, SYSCAT REFRESH cannot re-create object references for the procedure. Instead, SYSCAT REFRESH deletes from ZNUPCOL any row that corresponds to a parent procedure no longer found in ZNUPROC.

USERCAT REFRESH Statement

Use the USERCAT REFRESH statement to add objects to the mapping tables. When you run USERCAT REFRESH, all objects that are not mapped become mapped and therefore are available through the NonStop ODBC Server.

You must run USERCAT REFRESH if objects registered in the catalog are not listed in the mapping tables. An object does not appear in the mapping tables if:

- You create the object in pass-through mode.

- You create the object using NonStop SQL/MP when the catalog is already customized.
- You delete a row from the NonStop ODBC Server mapping table ZNUOBJ.

You can refresh the system catalog if you delete a row from the ZNSDB mapping table. USERCAT REFRESH re-creates the mapping entry for the customized catalog and refreshes the object mapping table for the system catalog.

USERCAT REFRESH has the following syntax:

```
USERCAT REFRESH [ catalog-name [ LOG output-filename ] ]
```

catalog-name

specifies the name of the subvolume containing the NonStop SQL/MP catalog to be refreshed. The *catalog-name* must be in Guardian format and can include the node and volume. There must be an entry in ZNSDB for *catalog-name* and the catalog must be the most current version. If the entry for *catalog-name* is missing in ZNSDB, you must first execute USERCAT INSTALL to create the entry. If the catalog is not the most current version, you must first execute USERCAT UPGRADE to upgrade the catalog version.

If the specified subvolume contains the system catalog, the mapping tables in the system catalog are refreshed.

If the specified subvolume does not contain a customized catalog, an error message is generated and the statement is not executed.

If you omit *catalog-name*, the current subvolume is used.

output-filename

specifies the NonStop SQL/MP table name of the log table where command response messages are written. The log table is an unaudited, entry-sequenced table that NOSUTIL creates in the catalog where VALIDATE, REFRESH, or INSTALL is being run. The log table also creates a mapping entry in the NonStop ODBC master catalog. If a table that has the same name as *output-filename* already exists, NOSUTIL appends the command response messages to the log table.

Example

The following statement refreshes the NonStop SQL/MP catalog \TEST.$VOL2.PERSNL:

```
usercat refresh \test.$vol2.persnl
```

USERCAT REFRESH verifies that all objects registered in the catalog are listed in the mapping tables. Objects not listed are added to the mapping tables.

Effects of USERCAT REFRESH

USERCAT REFRESH does the following:

- Checks whether each table, view, and index in the specified catalog has an entry in the mapping tables. If the object is not entered in the mapping tables, USERCAT REFRESH enters it.
- Re-creates any NonStop ODBC Server catalog table, index, or view not found.
- Adds TABLE, or VIEW, or INDEX, as a logical username to ZNSUS for any object not found in the mapping table.
- Corrects any owner or type discrepancies between a mapping entry and its Guardian file.
- Deletes any mapping table entry that references a nonexistent NonStop SQL/MP object.
- Re-creates any mapping tables that are not found and resupplies them with values.Generates an error if you are not a privileged user, or if you did not create this NonStop ODBC Server customized catalog.
- Generates an error if the catalog is not customized for the NonStop ODBC Server.

If An Object Is Partitioned

For each object with multiple partitions, USERCAT REFRESH adds only one entry to the ZNUOBJ mapping table. Even if primary and secondary partitions of the same object are registered in different catalogs, ZNUOBJ has only one entry for the object. NonStop ODBC Server determines which partition to map as follows:

- If the primary partition is located in the same subvolume as the specified catalog**,** the primary partition is the mapped object.
- If the primary partition is not located in the same subvolume as the specified catalog, the first partitioned object in the list of objects in the catalog (TABLES, INDEXES) is the mapped object.
- If the object has no local partitions, the primary partition is the mapped object.

How ODBC/SQL Server Object Names Are Assigned

When you customize or refresh a catalog, the NonStop ODBC Server assigns ODBC/SQL Server names to the objects in the catalog. The NonStop ODBC Server usually assigns the Guardian file name. For example, a NonStop SQL/MP table named \TESS.$NOSS.PERSNL.EMPLOYEE is given the ODBC/SQL Server name EMPLOYEE. Sometimes, however, the NonStop ODBC Server must assign a name other than the Guardian file name.

For information about viewing and changing the mapped names, see Renaming ODBC or SQL Server Objects on page 7-7.

Note. USERCAT REFRESH truncates object names that are more than 8 characters long. To preserve a longer object name, manually add the object to ZNUOBJ by executing ADD TABLE, INDEX, or VIEW.

USERCAT REFRESH also changes an object name in ZNUOBJ if you change the owner of the object from NonStop SQL/MP.

How USERCAT REFRESH Affects Stored Procedures

Stored procedures are not necessarily SQL objects. If a stored procedure becomes invalid or is deleted from ZNUPROC and ZNUPCOL, USERCAT REFRESH cannot re-create object references for the procedure. Instead, USERCAT REFRESH deletes from ZNUPCOL any row that corresponds to a parent procedure no longer found in ZNUPROC.

SYSCAT VALIDATE Statement

Use the SYSCAT VALIDATE statement to report all referential inconsistencies within the mapping tables.

Nonexistent objects will be reported in your mapping tables if you drop mapped tables by using NonStop SQL/MP or pass-through mode, or if you drop a table without dropping dependent views and indexes. When you drop a table that has dependent objects, the dependent objects are dropped from the NonStop SQL/MP catalog, but they remain in the mapping tables. For more information, see DROP TABLE in Section 3, CORE SQL Language,or Section 4, Transact-SQL Language.

Mapping tables will be missing if you accidentally purge them.

SYSCAT VALIDATE also reports any object owner name not found in the system users (ZNSUS) mapping table.

You can validate the system catalog only if you are a privileged user.

SYSCAT VALIDATE has the following syntax:

```
SYSCAT VALIDATE [ LOG output-filename ]
```

output-filename

specifies the NonStop SQL/MP table name of the log table where command response messages are written. The log table is an unaudited, entry sequenced table that NOSUTIL creates in the NonStop ODBC master catalog and creates a mapping entry. If a table that has the same name as *output-filename* already exists, NOSUTIL returns an error and the USERCAT INSTALL statement terminates.

Example

The following statement validates the NonStop SQL/MP system catalog:

```
syscat validate
```

Effects of SYSCAT VALIDATE

SYSCAT VALIDATE reports all object reference inconsistencies for the system catalog. SYSCAT VALIDATE verifies that:

- Mapping tables and views exist.
- The name of the database is registered in the NonStop ODBC Server system catalog.
- Tables, views, and indexes with mapping table entries actually exist as NonStop SQL/MP objects and are the correct object type.
- Entries in ZNSUS refer to valid Guardian usernames and user IDs
- Mapping table entries that refer to NOS_UID and T_UID correspond to a user registered in ZNSUS.
- Stored procedures referred to in ZNUPCOL are found in ZNUPROC.The number of input and input-output parameters in ZNUPROC equals the number of columns defined in ZNUPCOL for input and input-output parameters.
- All entries in SYSDATABASES correspond to NonStop ODBC Server customized subvolumes and NonStop SQL/MP catalogs.

SYSCAT VALIDATE also produces a list of all exceptions detected.

USERCAT VALIDATE Statement

Use the USERCAT VALIDATE statement to report all referential inconsistencies within the mapping tables in a user catalog.

Nonexistent objects are reported in your mapping tables if you do the following:

- Drop mapped tables using NonStop SQL/MP or using the NonStop ODBC Server in pass-through mode.
- Drop a table without dropping dependent views and indexes. When you drop a table that has dependent objects, the dependent objects are dropped from the NonStop SQL/MP catalog, but they remain in the mapping tables. For more information, see DROP TABLE on page 3-48, or in Section 4, Transact-SQL Language.

Mapping tables will be missing if you accidentally purge them.

SYSCAT VALIDATE also reports any object owner name not found in the system users (ZNSUS) mapping table.

USERCAT VALIDATE has the following syntax:

```
USERCAT VALIDATE [ catalog-name [ LOG output-filename ] ]
```

catalog-name

specifies the name of the subvolume containing the NonStop SQL/MP catalog to be validated. The *catalog-name* appears in Guardian format and can include the node and volume. If you omit *catalog-name*, the current subvolume is used.

If the specified subvolume contains the system catalog, the mapping table that lists all customized catalogs is validated.

Warning messages are generated if any mapping tables cannot be found.

output-filename

specifies the NonStop SQL/MP table name of the log table where command response messages are written. The log table is an unaudited, entry-sequenced table that NOSUTIL creates in the catalog where VALIDATE, REFRESH, or INSTALL is being run. The log table also creates a mapping entry in the NonStop ODBC master catalog. If a table that has the same name as *output-filename* already exists, NOSUTIL appends the command response messages to the log table.

Example

The following statement validates the NonStop SQL/MP catalog \TEST.$VOL2.PERSNL:

```
usercat validate \test.$vol2.persnl
```

Effects of USERCAT VALIDATE

USERCAT VALIDATE reports on all object reference inconsistencies for any catalog you specify. USERCAT VALIDATE verifies the following:

- Mapping tables and views exist.
- The name of the database is registered in the NonStop ODBC Server system catalog
- Tables, views, and indexes with mapping table entries actually exist as NonStop SQL/MP objects and are the correct object type.
- Mapping table entries that refer to NOS_UID and T_UID correspond to a user registered in ZNSUS.
- Stored procedures referred to in ZNUPCOL are found in ZNUPROC.
- The number of input and input-output parameters in ZNUPROC equals the number of columns defined in ZNUPCOL for input and input-output parameters.

USERCAT VALIDATE also produces a list of all exceptions detected.

CLEANUP Statement

Use the CLEANUP statement to drop temporary objects. Although the NonStop ODBC Server drops temporary objects at the end of each session, if the objects are not dropped, you must drop them manually.

Although the NonStop ODBC Server does not notify you when temporary objects are not dropped, you will know when to drop temporary objects if you try to create a temporary object that you know should not exist, and you receive an error message saying that the object already exists.

Use CLEANUP to drop temporary objects for any of the following:

- A customized database
- One user or all users
- One session for a user or all sessions for that user

Syntax

CLEANUP has the following syntax:

```
CLEANUP [ logical-database-name [ USER logical-user-name
              [ SESSION session-id ]]]
```

logical-database-name

is an alphanumeric string up to 60 characters in length that specifies the logical name of the NonStop ODBC Server customized database to clean up.

If you omit *logical-database-name*, the current database is used by default.

logical-user-name

specifies the owner of the temporary objects to be dropped. The *logical-user-name* is a group name followed by a username, as described in Section 3, CORE SQL Language, or Section 4, Transact-SQL Language. The *logical-user-name* can be any NOS_USERNAME found in the ZNSUS mapping table.

If you omit *logical-user-name*, temporary objects for all users are dropped.

session-id

is the internally generated session ID of the current session. It is used to qualify all temporary objects created during that session. Normally, a NonStop ODBC Server user session issues a CLEANUP statement for any temporary tables created during the session.

If you omit *session-id*, all temporary objects that belong to the current user are dropped.

Examples

The following examples show uses of CLEANUP.

Drop All Temporary Objects

The following statement drops all temporary objects in the PERSNL catalog:

```
cleanup $oook.persnl
```

All temporary objects are dropped regardless of who owns them.

Drop Temporary Objects for Several Catalogs and a Specific User

The following statement drops temporary objects in the PERSNL, SALES and INVENT catalogs if the objects are owned by the user PUBS_JANET:

```
cleanup $oook.persnl $oook.sales $oook.invent user pubs_janet
```

Effects of CLEANUP

CLEANUP does the following:

- Deletes entries from the appropriate mapping tables
- Drops temporary objects from the NonStop SQL/MP catalog
- Generates an error if you are not a privileged user
- Generates an error for each NonStop SQL/MP object you are not authorized to purge
- Generates an error if you are not the same user that customized the specified catalog or database
- Generates an error if *catalog-name* is not a customized NonStop ODBC Server system catalog
- Generates an error if the temporary table to clean up is open when you execute CLEANUP

SYSCAT UPGRADE Statement

Use the SYSCAT UPGRADE statement to change the mapping tables so that the system catalog conforms to the current version specification.

SYSCAT UPGRADE has the following syntax:

```
SYSCAT UPGRADE
```

Effects of SYSCAT UPGRADE

SYSCAT UPGRADE makes the following changes to the system catalog:

- Alters the definition of the following tables and supplies a default value for each new column: ZNSUS, ZNSDB, and ZNSPROF
- Renames ZNSTRC to ZNSTRA and ZNSDT to ZNUDT
- Adds the following tables:

ZNSCON	ZNSGOV	ZNSSER	ZNSSCS	ZNSUMAP
ZNSDEF	ZNSNET	ZNSSCFG	ZNSSMAP	

- Adds mapping entries for these objects
- Updates the database version value in ZNSDB
- Generates an error if the specified catalog is not a NonStop ODBC Server version 100 or 110 catalog

USERCAT UPGRADE Statement

Use the USERCAT UPGRADE statement to change the mapping tables so that the specified user catalog conforms to the current version specification.

USERCAT UPGRADE has the following syntax:

```
USERCAT UPGRADE [ catalog-name ]
```

catalog-name

specifies the catalog to be upgraded. If you omit *catalog-name*, the catalog in the current subvolume is used by default.

Example

The following statement shows the use of USERCAT UPGRADE:

```
usercat upgrade \test.$vol2.persnl
```

Effects of USERCAT UPGRADE

The USERCAT UPGRADE statement makes the following changes to the system catalog:

- Adds mapping entries for new objects in ZNUOBJ
- Adds ZNUDT table
- Updates the database version value in ZNSDB

- Generates an error if the specified catalog is not a NonStop ODBC Server version 100 or 110 catalog

Maintaining Mapping Tables

Table 7-4 lists the NonStop ODBC Server mapping table statements by type. You can execute these statements either from the TACL prompt by running NOSCOM or from the NonStop ODBC Server Configuration Manager. Use these statements to add items to, change existing items in, and delete items from the mapping tables.

Because the NonStop ODBC Server is so flexible, your installation can use defaults other than the initial ones provided. Initial defaults are listed in Appendix D, Summary of System Installation Defaults.

Table 7-4. Summary of Mapping Table Statements (page 1 of 3)

Statement Type	What This Type of Statement Affects
ACC_LOG	Accounting log tables
CREATE ACC_LOG DROP ACC_LOG	
ALIAS	Mapping of alias usernames to logical usernames
ADD ALIAS MODIFY ALIAS REMOVE ALIAS	
CONTROL	Stored NonStop SQL/MP CONTROL statements
ADD CONTROL REMOVE CONTROL	
GOVERNING	Resource governing policy entities
ADD GOVERNING MODIFY GOVERNING REMOVE GOVERNING	
INDEX	Visibility of NonStop SQL/MP index tables
ADD INDEX REMOVE INDEX	
NET_SERVICE	Named network service configurations
ADD NET_SERVICE MODIFY NET_SERVICE REMOVE NET_SERVICE	
PROFILE	Named profile configurations
ADD PROFILE MODIFY PROFILE REMOVE PROFILE	

Table 7-4. Summary of Mapping Table Statements (page 2 of 3)

Statement Type	What This Type of Statement Affects
PROCEDURE	Stored procedures
ADD PROCEDURE ADD PROCEDURE_COLUMNS MODIFY PROCEDURE REMOVE PROCEDURE	
QST_LOG	Query status log tables
CREATE QST_LOG DROP QST_LOG	
SCFG	System configuration defaults
ADD SCFG MODIFY SCFG REMOVE SCFG	
SCS	SQL Communications Subsystem processes:
ADD SCS MODIFY SCS REMOVE SCS	Process configuration
START SCS STATUS SCS STOP SCS	Process activity
SERVERCLASS	Server class definitions
ADD SERVERCLASS MODIFY SERVERCLASS REMOVE SERVERCLASS	Server class configuration
START SERVERCLASS STOP SERVERCLASS	Process activity
SMAP	Server class mapping to an SCS configuration
ADD SMAP MODIFY SMAP REMOVE SMAP	
TABLE	Visibility of NonStop SQL/MP tables
ADD TABLE REMOVE TABLE	
TRA_LOG	Trace log tables
CREATE TRA_LOG DROP TRA_LOG	
TRACE	Named trace configurations
ADD TRACE MODIFY TRACE REMOVE TRACE	

Table 7-4. Summary of Mapping Table Statements (page 3 of 3)

Statement Type	What This Type of Statement Affects
UMAP ADD UMAP MODIFY UMAP REMOVE UMAP	Mapping of user and profile to a server class
USER ADD USER MODIFY USER REMOVE USER	Mapping of logical usernames to Guardian usernames
VIEW ADD VIEW REMOVE VIEW	Visibility of NonStop SQL/MP views

Accounting Log (ACC_LOG) Statements

Use the following statements for NonStop ODBC Server accounting log tables:

- CREATE ACC_LOG
- DROP ACC_LOG

CREATE ACC_LOG

The CREATE ACC_LOG statement creates a new accounting log table named ZNUMTRX and registers the table as a NonStop ODBC Server object in a database.

```
CREATE ACC_LOG IN database [ SECURE security-string ]
```

`IN` *`database`*

specifies the name of the database in which to register the new accounting log table. *`database`* must be a customized NonStop ODBC Server database; otherwise, the statement returns an error.

`SECURE` *`security-string`*

is a four-character alphabetic string that specifies the Guardian read, write, execute, and purge (RWEP) access for the accounting log table. For more information about security for NonStop SQL/MP objects, see the “Security” entry in the *NonStop SQL/MP Reference Manual*.

Considerations – CREATE ACC_LOG

The CREATE ACC_LOG statement returns an error if the associated SQL CREATE TABLE statement fails for resource or privilege reasons.

Example

```
CREATE ACC_LOG IN persnl SECURE COOO
```

DROP ACC_LOG

The DROP ACC_LOG statement drops an existing accounting log table and removes the mapping entry for the log table in the specified database. To execute DROP ACC_LOG, you must be a privileged user.

```
DROP ACC_LOG FROM database
```

`FROM` *`database`*

specifies the name of the database from which to drop the accounting log table. *`database`* must be a customized NonStop ODBC Server database; otherwise, the statement returns an error.

Considerations – DROP ACC_LOG

The DROP ACC_LOG statement:

- Preserves the entry in ZNSPROF for the log table
- Returns an error if you are not a privileged user
- Returns an error if either the database cannot be updated for resource or privilege reasons or the associated SQL DROP TABLE statement fails

Example

```
DROP ACC_LOG FROM persnl
```

ALIAS Statements

Use the following statements to map alias usernames to logical usernames and user profiles:

- ADD ALIAS
- MODIFY ALIAS
- REMOVE ALIAS

ADD ALIAS

The ADD ALIAS statement associates an alias username with a logical username and, optionally, with a profile. An alias username cannot refer to another alias username.

To execute ADD ALIAS, you must be authorized to modify the system catalog or be the user associated with *logical-username*.

```
ADD ALIAS alias-username
          [ NOS_USERNAME logical-username ]
          [ CHANGE_PASSWORD_OPTION { 0 | 1 | 2 } ]
          [ PROFILE profile-name ]
          [ UPDATE_SYSTEM_CONFIG ]
```

alias-username

is an alphanumeric string up to 60 characters in length that specifies a unique alias username to associate with *logical-username*. *alias-username* must begin with an alphabetic character, followed by alphabetic, numeric, and underscore characters.

NOS_USERNAME *logical-username*

is an alphanumeric string up to 60 characters in length that specifies a logical username to associate with *alias-username*. *logical-username* must begin with an alphabetic character, followed by alphabetic, numeric, and underscore characters.

CHANGE_PASSWORD_OPTION { 0 | 1 | 2 }

specifies the level of notification sent to users for Safeguard passwords that are expired or are about to expire:

0	No notification is sent. A user can log in only if the password is correct and has not expired, including the grace period. This is the default.
1	Notification is sent if the password is still in the grace period. A user can then change the password and log in.
2	Notification is sent if the password is about to expire or if it has expired but is still in the grace period. If the password has not yet expired, the user can change the password or cancel the dialog box. If the password has already expired, the user must change the password immediately.

PROFILE *profile-name*

specifies the name of a profile to associate with *alias-username*. The value must be either an SQL identifier or an asterisk (*). An asterisk indicates that the profile to use is to be determined at run time from the ZNSALT record in which NOS_ALIASNAME is equated with *logical-username*.

If you omit *profile-name*, the profile name DEFAULT is used.

`UPDATE_SYSTEM_CONFIG`

resets the time to update system configuration values to the current time. This option causes running system components to reread their configuration values on the next polling cycle.

Considerations – ADD ALIAS

The ADD ALIAS statement:

- Creates a new entry in the ZNSALT table that allows the client to use the alias username to log in or to qualify objects. An alias username is translated only once.
- Causes objects created by the alias username to be stored as if they were created by the corresponding logical username
- Generates an error if *`alias-username`* already exists in the system catalog table ZNSALT
- Generates a warning if *`logical-username`* is not a valid NonStop ODBC Server logical username
- Generates a warning if *`profile-name`* is not a valid NonStop ODBC Server profile
- Generates an error if the system catalog is not customized for the NonStop ODBC Server
- Generates an error if you are not authorized to modify the system catalog or the user associated with *`logical-username`*

Example

The following ADD ALIAS statement adds the alias name Jones for the logical username SQL_Jones using the profile associated with SQL_Jones:

```
ADD ALIAS Jones NOS_USERNAME SQL_Jones PROFILE_NAME *
```

MODIFY ALIAS

The MODIFY ALIAS statement changes the mapping between an alias username and a logical username or changes the profile name associated with an existing alias. (To change the mapping between a logical username and a Guardian username, use the MODIFY USER statement.)

To execute MODIFY ALIAS, you must be a privileged user or the user associated with *`logical-username`*.

```
MODIFY ALIAS alias-username
   [ NOS_USERNAME logical-username ]
   [ CHANGE_PASSWORD_OPTION { 0 | 1 | 2 } ]
   [ PROFILE profile-name ]
   [ UPDATE_SYSTEM_CONFIG ]
```

`alias-username`

specifies the alias username you want to modify.

`NOS_USERNAME` *`logical-username`*

specifies the logical username associated with *`alias-username`*. *`logical-username`* must begin with an alphabetic character, followed by alphabetic, numeric, and underscore characters.

`CHANGE_PASSWORD_OPTION { 0 | 1 | 2 }`

specifies the level of notification sent to users for Safeguard passwords that are expired or are about to expire. See the ADD ALIAS statement for a description of the options.

`PROFILE` *`profile-name`*

specifies the name of a profile to associate with *`alias-username`*. The value must be either an SQL identifier or an asterisk (*). An asterisk indicates that the profile to use is to be determined at run time from the ZNSALT record in which NOS_ALIASNAME is equated with *`logical-username`*.

If you enter the PROFILE keyword, you must also enter *`profile-name`*. If you omit the entire parameter, the profile name DEFAULT is assumed.

`UPDATE_SYSTEM_CONFIG`

resets the time to update system configuration values to the current time. This option causes running system components to reread their configuration values on the next polling cycle.

Considerations – MODIFY ALIAS

The MODIFY ALIAS statement:

- Changes the existing logical username or profile name entry in ZNSALT for the specified *`alias-username`*
- Generates a warning if *`alias-username`* is not a valid NonStop ODBC Server alias username
- Generates a warning if *`logical-username`* is not a valid NonStop ODBC Server logical username
- Generates a warning if *`profile-name`* is not a valid NonStop ODBC Server profile
- Generates an error if you are not a privileged user or the user associated with *`logical-username`*

Example

```
MODIFY ALIAS janet PROFILE mgr UPDATE_SYSTEM_CONFIG
```

REMOVE ALIAS

The REMOVE ALIAS statement removes the existing association between an alias username and a logical username. To execute REMOVE ALIAS, you must be a privileged user.

```
REMOVE ALIAS alias-username [ UPDATE_SYSTEM_CONFIG ]
```

alias-username

specifies the alias username.

`UPDATE_SYSTEM_CONFIG`

resets the time to update system configuration values to the current time. This option causes running system components to reread their configuration values on the next polling cycle.

Considerations – REMOVE ALIAS

The REMOVE ALIAS statement:

- Removes the specified alias username from ZNSALT
- Generates an error if *alias-username* is self-referencing
- Generates an error if *alias-username* cannot be found
- Generates an error if you are not a privileged user

Example

```
REMOVE ALIAS Jones
```

CONTROL Statements

The following statements allow you to control various aspects of query execution using NonStop SQL/MP CONTROL statements:

- ADD CONTROL
- REMOVE CONTROL

ADD CONTROL

The ADD CONTROL statement adds a new NonStop ODBC Server control statement entity to execute at start up time and re-initialization boundaries. A control statement entity is linked to a profile; the NonStop ODBC Server executes the statement when a NOS process is started or when you use a profile with an associated CONTROL statement.

To execute ADD CONTROL, you must be authorized to modify the system catalog.

```
ADD CONTROL control-name "control-text"
            [ UPDATE_SYSTEM_CONFIG ]
```

control-name

is an alphanumeric string up to 60 characters in length that specifies the name of the control statement entity; *control-name* must begin with an alphabetic character, followed by alphabetic, numeric, and underscore characters. The default for is NULL.

control-text

is an alphanumeric string up to 3900 characters in length, enclosed in double quotes ("), that specifies the text of the control statement. This text must follow the syntax for a NonStop SQL/MP CONTROL statement and is checked for validity only at run time. For more information about CONTROL statements, see the *NonStop SQL/MP Reference Manual*.

UPDATE_SYSTEM_CONFIG

resets the time to update system configuration values to the current time. This option causes running system components to reread their configuration values on the next polling cycle.

Considerations – ADD CONTROL

The ADD CONTROL statement:

- Creates a new entry for the control statement entity in ZNSCON
- Executes the statement when a NOS process is started or when you use a profile associated with a CONTROL statement

- Generates an error at run time if `control-text` is not a valid NonStop SQL/MP CONTROL statement
- Generates an error if you are not authorized to modify the system catalog

Example

```
ADD CONTROL late_bind "CONTROL QUERY BIND NAMES AT EXECUTION"
```

REMOVE CONTROL

The REMOVE CONTROL statement deletes an existing control statement entity in the ZNSCON table. To execute REMOVE CONTROL, you must be authorized to modify the system catalog.

```
REMOVE CONTROL control-name
               [ UPDATE_SYSTEM_CONFIG ]
```

`control-name`

specifies the name of the control statement entity you want to remove form the ZNSCON table.

`UPDATE_SYSTEM_CONFIG`

resets the time to update system configuration values to the current time. This option causes running system components to reread their configuration values on the next polling cycle.

Considerations – REMOVE CONTROL

The REMOVE CONTROL statement generates an error if you are not authorized to modify the system catalog.

Example

```
REMOVE CONTROL late_bind UPDATE_SYSTEM_CONFIG
```

DEFINE Statements

The following statements allow you to use DEFINEs with the SQL Communications Subsystem (SCS):

- ADD DEFINE
- REMOVE DEFINE

A DEFINE is a named set of attribute-value pairs associated with a process. You can use DEFINEs to pass information to a process when you start the process. For the NonStop ODBC Server, the DEFINE is linked to an SCS configuration entity and is propagated when the SCS starts a new process.

For more information about DEFINE classes, attributes, and values, see "DEFINEs" and "System DEFINEs" in the *NonStop SQL/MP Reference Manual.*

ADD DEFINE

The ADD DEFINE statement creates a new DEFINE entry in the ZNSDEF system table. The DEFINE takes effect when SCS starts a new process. To execute ADD DEFINE, you must be authorized to modify the system catalog.

```
ADD DEFINE define-name, [ CLASS define-class, ]
           define-attribute [attribute-value ]
           [, define-attribute [attribute-value ]...]
           [ FOR scs-processname ]
           [ UPDATE_SYSTEM_CONFIG ]
```

define-name

is the name of the new DEFINE; *define-name* is an alphabetic string from 2 to 24 characters in length. The first character must be an equal sign (=), and the second character must be a letter. The remaining name can be alphanumeric characters, hyphens (-), and underscores (_).

CLASS *define-class*

is an alphabetic string up to 16 characters in length that identifies the class of the new DEFINE. The default is MAP.

define-attribute

is an alphanumeric string up to 512 characters in length that identifies the DEFINE attribute.

define-value

is the value you assign to *define-attribute*. To omit *define-value*, enter two double quotes (""). For a list of DEFINE attributes and values, see the SET DEFINE command in the *TACL Reference Manual*.

`FOR` *`scs-processname`*

is an alphabetic string up to 15 characters in length in the following format that specifies the name of the SCS process to associate with this DEFINE statement:

`[\`*`node`*`.]$`*`process-name`*

All processes dependent on *`scs-processname`* are also affected by the new DEFINE. If you omit *`scs-processname`*, the DEFINE statement applies to all SCS processes and their dependent processes.

`UPDATE_SYSTEM_CONFIG`

resets the time to update system configuration values to the current time. This option causes running system components to reread their configuration values on the next polling cycle.

Considerations – ADD DEFINE

The ADD DEFINE statement:

- Generates an error if you are not authorized to modify the system catalog
- Generates an error if you do not specify DEFINE statement attributes in the correct order

Example

```
ADD DEFINE =_SQL_CMP_PARALLEL_ON, CLASS MAP, FILE noname FOR
$scs1
```

REMOVE DEFINE

The REMOVE DEFINE statement deletes an existing DEFINE entry and its related attributes from the ZNSDEF table. To execute REMOVE DEFINE, you must be authorized to modify the system catalog.

```
REMOVE DEFINE define-name
              [ FOR scs-processname ]
              [ UPDATE_SYSTEM_CONFIG ]
```

`define-name`

is the name of the DEFINE you want to remove.

`FOR` *`scs-processname`*

specifies the name of the SCS process associated with the DEFINE; it is an alphabetic string up to 15 characters in length in the following format:

`[\`*`node`*`.]$`*`process-name`*

All processes dependent on *scs-processname* are also affected. If you omit *scs-processname*, the DEFINE statement applies to all SCS processes and their dependent processes.

`UPDATE_SYSTEM_CONFIG`

resets the time to update system configuration values to the current time. This option causes running system components to reread their configuration values on the next polling cycle.

Example

```
REMOVE DEFINE =_SQL_CMP_PARALLEL_ON
```

INDEX Statements

Use the following statements to control whether a NOSCOM user can access an index on a NonStop SQL/MP base table:

- ADD INDEX
- REMOVE INDEX

ADD INDEX

The ADD INDEX statement makes an index on a NonStop SQL/MP base table visible to a NOSCOM user. No REFRESH of the entire database is necessary.

To execute ADD INDEX, you must be authorized to modify the current user catalog. However, the NonStop ODBC Server does not verify privileges on the underlying NonStop SQL/MP index.

```
ADD INDEX nssql-filename [ AS logical-objectname ]
```

nssql-filename

is the name of a Guardian file that specifies the physical location of the index; *nssql-filename* can be in any of the following formats:

```
filename
subvolume.filename
$volume.subvolume.filename
\node.filename
\node.subvolume.filename
\node.$volume.subvolume.filename
```

The NonStop ODBC Server fully qualifies *nssql-filename* before performing the INDEX operation. Any optional values you omit are determined by the user profile of the current process.

AS *`logical-objectname`*

specifies the logical name for the index; *`logical-objectname`* can be in any of the following formats:

```
object-name
logical-user-name.object-name
database-name.logical-user-name.object-name
database-name..object-name
```

Any value you specify for *`database-name`* must be a customized NonStop ODBC Server database. If you omit *`database-name`*, the default is the default database defined in the current profile. If the current profile does not specify a default database, then *`database`* defaults to MASTER unless a USE DATABASE statement is in effect.

The NonStop ODBC Server fully qualifies *`logical-objectname`* before performing the INDEX operation. Any optional values you omit are determined by the user profile of the current process.

For ADD INDEX, the NonStop SQL/MP base table must also be mapped in the specified NonStop ODBC Server database.

The *`logical-objectname`* you specify cannot currently be mapped to a different Guardian file name. If you specify *`logical-user-name`*, it must map to the owner of *`nssql-filename`*. If this association does not currently exist, it is automatically created when you execute ADD INDEX.

Considerations – ADD INDEX

The ADD INDEX statement:

- Creates new mapping entries in ZNUOBJ and ZNUIX for the specified index table
- Adds a mapping entry to ZNSUS, if *`logical-user-name`* is not currently mapped to the owner of *`nssql-filename`*
- Generates an error if *`nssql-filename`* is already mapped in the appropriate object file
- Generates an error if *`nssql-filename`* is not a valid NonStop SQL/MP index
- Generates an error if the index table you specify is not registered in the NonStop SQL/MP database
- Generates an error if the corresponding base table is not registered in this NonStop ODBC Server database
- Generates an error if you are not authorized to modify the current user catalog
- Generates an error if *`database-name`* is not a customized NonStop ODBC Server database
- Generates an error if *`logical-objectname`* is already mapped to a different Guardian file name

Example

```
ADD INDEX \sales.$db1.region1.ix AS db.o.ix
```

REMOVE INDEX

The REMOVE INDEX statement deletes an existing mapping entry for the specified index. The underlying NonStop SQL/MP index is unaffected. To execute REMOVE INDEX, you must be a privileged user.

```
REMOVE INDEX nssql-filename [ FROM database-name ]
```

nssql-filename

is the name of a Guardian file that specifies the physical location of the index; *nssql-filename* can be in any of the following formats:

```
filename
subvolume.filename
$volume.subvolume.filename
\node.filename
\node.subvolume.filename
\node.$volume.subvolume.filename
```

The NonStop ODBC Server fully qualifies *nssql-filename* before performing the INDEX operation. Any optional values you omit are determined by the user profile of the current process.

FROM *database-name*

specifies a customized NonStop ODBC Server database. If you omit *database-name*, the default is the default database defined in the current profile. If the current profile does not specify a default database, then *database* defaults to MASTER unless a USE DATABASE statement is in effect.

Considerations – REMOVE INDEX

The REMOVE INDEX statement:

- Deletes the mapping entry in ZNUOBJ for *nssql-filename*
- Deletes the mapping entry in ZNUIX associated with *nssql-filename*
- Generates an error if you are not a privileged user

Example

```
REMOVE INDEX \sales.$db1.region1.ix UPDATE_SYSTEM_CONFIG
```

Network Services (NET_SERVICE) Statements

Use the following statements to configure and manage network services:

- ADD NET_SERVICE
- MODIFY NET_SERVICE
- REMOVE NET_SERVICE

ADD NET_SERVICE

The ADD NET_SERVICE statement creates a new network service entity for a specified SCS process. To execute ADD NET_SERVICE, you must be a privileged user.

The ADD NET_SERVICE statement performs these functions:

- Creates a new entry for the network service in the ZNSNET table.
- If no value is specified for an attribute, uses a default value.
- Generates a warning if you specify TCP-IP for *net-protocol-name*, and *net-name* is not defined in your SERVICES file.
- Generates an error if you are not authorized to modify the system catalog.

The syntax for the ADD NET_SERVICE statement has two formats:

- TCP/IP and SPX/IPX protocols
- NETBIOS protocol

TCP/IP and SPX/IPX Protocols

Use the following syntax for the TCP/IP or SPX/IPX protocol:

```
ADD NET_SERVICE scs-processname
    NET_PROTOCOL { TCP/IP | SPX/IPX }
    NET_NAME net-name
    IOP_NAME iop-name
    [ SERVICES_FILENAME services-filename ]
    [ SO_KEEPALIVE { 0 | 1 } ]
    [ SO_OOBINLINE { 0 | 1 } ]
    [ SO_LINGER    { 0 | 1 } ]
    [ SO_REUSEADDR { 0 | 1 } ]
    [ UPDATE_SYSTEM_CONFIG ]
```

scs-processname

is the name of the SCS process for which you want to create a network service entity.

`NET_PROTOCOL { TCP/IP | SPX/IPX }`

specifies TCP/IP or SPX/IPX as the network protocol to use for this service.

`NET_NAME` *`net-name`*

identifies the network service; *`net-name`* is a string up to 16 characters in length that must match the name of the service in your system SERVICES file.

Caution. You can enter names for the NET_PROTCOL and NET_NAME names using lowercase letters because NOSUTIL upshifts each name. However, if you edit the system SERVICES file directly, you must enter each name using uppercase letters.

`IOP_NAME` *`iop-name`*

is an alphabetic string up to 15 characters in length that identifies a valid network process in the following format:

`[\`*`node.`*`]`*`$process-name`*

You set the default for this parameter when you install the NonStop ODBC Server. Each network protocol uses a different default name for a network process; the value you specify depends on your protocol and whether your node uses the default name.

`SERVICES_FILENAME` *`services-filename`*

is a fully qualified file name in Guardian format that specifies the location of the system SERVICES file. If you omit *`services-filename`* and a default value is not defined in ZNSSCFG, the current default for your system is used.

The following attributes configure socket operations for TCP/IP and SPX/IPX networks. These attributes correspond to socket options in `setsocketopt` and `setsocketopt_nw` syntax. The `setsocketopt` and `setsocketopt_nw` socket library routines are part of the NonStop TCP/IP and SPX/IPX products.

`SO_KEEPALIVE { 0 | 1 }`

specifies whether to keep a socket connection alive when not in use:

0	Do not keep a socket connection alive when not in use.
1	Keep a socket connection alive when not in use. (This is the default value.)

`SO_OOBINLINE { 0 | 1 }`

specifies whether TCP/IP keeps out-of-band data with normal data:

0	Keep out-of-band data separate from normal data.
1	Keep out-of-band data with normal data

If you keep out-of-band data with normal data, the application must discard normal data until the out-of-band data is read.

`SO_LINGER { 0 | 1 }`

specifies whether TCP/IP closes all connections gracefully and wait for data transfer to complete:

0	Do not require connections to close gracefully.
1	Always close connections gracefully.

The SO_LINGER option is provided only for compatibility. All C TCP/IP connections close gracefully.

`SO_REUSEADDR { 0 | 1 }`

specifies whether TCP/IP allows reuse of a local port address during a BIND operation:

0	Do not allow reuse of a local port address during a BIND operation.
1	Allow reuse of a local port address during a BIND operation.

NetBIOS Protocol

Use the following syntax for the NetBIOS protocol:

```
ADD NET_SERVICE scs-processname
    NET_PROTOCOL NETBIOS
    NET_NAME net-name
    IOP_NAME iop-name
    [ NET_QUALIFIER "net-qualifier-name" ]
    [ MLAN_DOMAIN sub-device-name ]
    [ MLAN_ADAPTOR { 0 | 1 } ]
    [ MLAN_GATEWAY #gateway-name ]
    [ UPDATE_SYSTEM_CONFIG ]
```

`scs-processname`

is the name of the SCS process for which to create or modify a network service entity.

`NET_PROTOCOL NETBIOS`

specifies NETBIOS as the network protocol to use for this service.

`NET_NAME` *`net-name`*

identifies the network service; *`net-name`* is a string up to 16 characters in length that must match the name of the service in your system SERVICES file.

Caution. You can enter values for the NET_PROTCOL and NET_NAME names using lowercase letters because NOSUTIL upshifts each value. However, if you edit the system SERVICES file directly, you must enter each name using uppercase letters.

`IOP_NAME` *`iop-name`*

is an alphabetic string up to 15 characters in length that identifies a valid network process in the following format:

`[\`*`node.`*`]`*`$process-name`*

You set the default for this parameter when you install the NonStop ODBC Server. Each network protocol uses a different default name for a network process; the value you specify depends on your protocol and whether your node uses the default name.

The following attributes help your application access the correct LAN in a multiple LAN environment. For more information about these Multilan objects, see the *Multilan/TLAM Programming Manual*.

`NET_QUALIFIER` *`net-qualifier-name`*

is an alphanumeric string up to 128 characters in length in the following format:

`"\PIPE\SQL\QUERY\`*`net-name`*`"`

where *`net-name`* is a string up to 12 characters that specifies your NetBIOS network name. This is also the MLAD system name for your device; it is not checked for validity. You must enclose the entire string in double quotes (").

`MLAN_DOMAIN` *`sub-device-name`*

is an alphanumeric string up to 8 characters in length that specifies a valid Multilan/TLAM subdevice name on the current node. A domain is a defined collection of devices that share a set of unique NetBIOS network names. A domain name begins with a pound sign (#), and is followed by an alphabetic character and up to 6 alphanumeric characters (#DOM1, for example). The value you specify is not checked for validity.

`MLAN_ADAPTOR { 0 | 1 }`

specifies the MLAN adapter to use for this network service. An adapter is an attachment device that provides a gateway access to a user LAN. The value can be 0 or 1. This value is not checked for validity.

`MLAN_GATEWAY` *`#gateway-name`*

is an alphanumeric string up to 8 characters that specifies a valid Multilan/TLAM gateway name on the current node. A gateway is a device attached to two networks of the same or different types. The gateway allows devices on both networks to communicate with each other. A gateway name begins with a pound sign (#), and is followed by an alphabetic character and up to 6 alphanumeric characters (#GW1, for example). This value is not checked for validity.

Example

```
ADD NET_SERVICE $svc4 NET_PROTOCOL TCP/IP NET_NAME odbctst1 &
    IOP_NAME \netsys.$lam1
```

MODIFY NET_SERVICE

The MODIFY NET_SERVICE statement modifies one or more of the attributes of an existing system network service for a specified SCS process in the ZNSNET table. To execute MODIFY NET_SERVICE, you must be authorized to modify the system catalog.

The MODIFY NET_SERVICE statement generates a warning if you specify TCP/IP for NET_PROTOCOL and *net-name* is not defined in your TCP/IP SERVICES file.

The syntax for the ADD NET_SERVICE statement has two formats:

- TCP/IP and SPX/IPX protocols
- NetBIOS protocol.

TCP/IP and SPX/IPX Protocols

Use the following syntax for the TCP/IP or SPX/IPX protocol:

```
MODIFY NET_SERVICE scs-processname
       [ NET_PROTOCOL { TCP/IP | SPX/IPX } ]
       [ NET_NAME net-name ]
       [ IOP_NAME iop-name ]
       [ SERVICES_FILENAME services-filename ]
       [ SO_KEEPALIVE { 0 | 1 } ]
       [ SO_OOBINLINE { 0 | 1 } ]
       [ SO_LINGER { 0 | 1 } ]
       [ SO_REUSEADDR { 0 | 1 } ]
       [ UPDATE_SYSTEM_CONFIG ]
```

NetBIOS Protocol

Use the following syntax for the NetBIOS protocol:

```
MODIFY NET_SERVICE scs-processname
       [ NET_PROTOCOL NETBIOS ]
       [ NET_NAME net-name ]
       [ IOP_NAME iop-name ]
       [ NET_QUALIFIER "net-qualifier-name" ]
       [ MLAN_DOMAIN sub-device-name ]
       [ MLAN_ADAPTOR mlan-adaptor
       [ MLAN_GATEWAY mlan-gateway ]
       [ UPDATE_SYSTEM_CONFIG ]
```

For a description of the parameters for the MODIFY NET_SERVICE statement, refer to the description of the ADD NET_SERVICE statement.

△ **Caution.** If you specify a new value for NET_PROTOCOL, NET_NAME, or IOP_NAME for an SCS process that is currently running, the SCS process will terminate abnormally.

Example

```
MODIFY NET_SERVICE $svc4 services_filename $ntwrk.odbc.srvc
```

REMOVE NET_SERVICE

The REMOVE NET_SERVICE statement removes the specified system network service. To execute REMOVE NET_SERVICE, you must be a privileged user.

```
REMOVE NET_SERVICE scs-processname [ UPDATE_SYSTEM_CONFIG ]
```

scs-processname

is the name of the SCS process associated with network service entry in the ZNSNET table you want to remove.

Considerations – REMOVE NET_SERVICE

The REMOVE NET_SERVICE statement:

- Removes the corresponding network service entry in ZNSNET
- Generates a warning if *scs-processname* is referenced by an SCS entity in ZNSSCS
- Generates an error if you are not a privileged user

Example

```
REMOVE NET_SERVICE $svc4
```

PROCEDURE Statements

Use the following statements to configure and manage stored procedures:

- ADD PROCEDURE
- ADD PROCEDURE_COLUMNS
- MODIFY PROCEDURE
- REMOVE PROCEDURE

ADD PROCEDURE

The ADD PROCEDURE statement adds an entry to the ZNUPROC table for procedure name mapping.

To execute ADD PROCEDURE, you must be authorized to modify the current user catalog.

```
ADD PROCEDURE logical-procedure-name
   PATHMON_NAME pathmon-processname
   SERVERCLASS pathway-server-class-name
   [ SERVICE service-name ]
   [ NUM_RESULT_SETS result-set-count ]
   [ RETURN_STATUS { N | Y } ]
   [ MAX-BUFFER-LEN ipc-message-buffer-len ]
   [ REMARKS "remarks-text" ]
```

logical-procedure-name

specifies the name of the new stored procedure in the following format:

```
[ [database.]schema.]procedure-name[;number ]
[  database..       ]
```

database

specifies a customized database in the current node. The default is the default database defined in the current profile. If the current profile does not specify a default database, the default for *database* is MASTER, unless a USE DATABASE statement is in effect.

schema

specifies a logical username in the current ZNSUS table. The default is the username of the current user.

procedure-name

is an alphanumeric string up to 60 characters in length that specifies the name of the procedure; *procedure-name* must begin with an alphabetic character, followed by alphabetic, numeric, and underscore characters.

number

is an integer in the range 0 through 32767 that is appended to the procedure name to identify the procedure within a group of procedures with the same name. The default is 1.

This number applies only to Transact-SQL users. To allow procedure sharing between Transact-SQL and CORE SQL users, HP recommends omitting it.

For more information about database, schema, and procedure names see Names on page 4-5.

`PATHMON_NAME` *`pathmon-processname`*

specifies the external Guardian name of the Pathmon process; it is an alphanumeric string up to 15 characters in length in the following format:

`[\`*`node-name`*`.]$`*`pathmon-processname`*

For example, a *`pathmon-processname`* might be $PM or \TESS.$PMN.

`node` specifies the node where the process is running. If *`node`* is omitted, the process location defaults to the current node.

`SERVERCLASS` *`pathway-server-class-name`*

specifies the name of the server class to which calls are to be sent (for example, EMP-SERVER). It is an alphanumeric string up to 15 characters, beginning with a letter and optionally containing hyphens (-). This name must conform to the Pathway server class naming rules, must be left justified in the buffer, and can contain trailing blanks.

`SERVICE` *`service-name`*

is used for a Pathway server class program to identify a specific service. *`service-name`* has the same format as the *`procedure-name`* part of the three-part procedure name and defaults to that name if omitted.

ADD PROCEDURE does not enforce the uniqueness of *`service-name`*; it is the Pathway programmer’s responsibility to do so.

`NUM_RESULT_SETS` *`result-set-count`*

specifies how many result sets are to be returned; *`result-set-count`* is an integer in the range 0 through 32,767. The default is zero (0).

A result set is a set of rows returned from a SELECT statement. If the stored procedure contains SELECT statements, the caller receives a result set for each SELECT statement. A result set includes the column names followed by zero or more rows of data.

`RETURN_STATUS {Y | N }`

specifies whether the stored procedure returns a status value in addition to any output parameter values and result sets:

Y	Return a status value.
N	Do not return a status value. N is the default.

`MAX-BUFFER-LEN` *`ipc-message-buffer-len`*

specifies the maximum size of the interprocess communication buffer; it is an integer in the range 0 through 32000. The default is 32000 bytes.

The Pathway server class program should also use MAX_BUFFER_LEN to allocate the message buffer for I/O. If the maximum buffer size is different between the requester (the NonStop ODBC Server) and the Pathway server class program, the lesser of the two sizes is used.

`REMARKS` *`remarks-text`*

specifies arbitrary comments for this stored procedure; it is a string up to 254 characters in length enclosed in double quotation (") marks.

Considerations – ADD PROCEDURE

The ADD PROCEDURE statement:

- Adds an entry in the ZNUPROC table for the specified procedure
- Generates an error if you are not authorized to modify the current user catalog

If an error occurs, the actions of ADD PROCEDURE are rolled back.

Example

The following ADD PROCEDURE statement defines a stored procedure named SELECT_DEPT3000, owned by user SQL_DAVE, in the NonStop ODBC Server catalog \TESS.$DATA1.DB. The procedure has the Pathmon process name $EMPSP and server name EMP-DATA-DML, returns one result set, and has a maximum reply buffer length of 2000 bytes:

```
ADD PROCEDURE tess_data1_db.sql_dave.select_dept3000 &
   PATHMON_NAME $empsp &
   SERVERCLASS EMP-DATA-DML &
   NUM_RESULT_SETS 1 &
   MAX_BUFFER_LEN 16000
```

ADD PROCEDURE_COLUMNS

The ADD PROCEDURE_COLUMNS statement adds one entry to the ZNUPCOL table for each parameter definition of a stored procedure.

```
ADD PROCEDURE_COLUMNS logical-procedure-name
     ( parameter-declaration [,parameter-declaration ]... )
```

`logical-procedure-name`

specifies the name of the stored procedure. For a description of this parameter, see the ADD PROCEDURE statement.

`parameter-declaration`

describes each parameter’s name, data type, and usage as follows:

`parameter-name parameter-type` `[` *`parameter-designation`* `]`

parameter-name

specifies the parameter name; *parameter-name* is an alphanumeric string up to 60 characters in length that must begin with an alphabetic character, followed by alphabetic, numeric, and underscore characters.

parameter-type

specifies the parameter data type as follows:

```
sql-data-type [ NULLABLE ] [ DEFAULT default-value ]
```

sql-data-type

is one of the following data types:

```
CHAR[ACTER] [ (length) ]
VARCHAR [ (length) ]
SMALLINT [ SIGNED | UNSIGNED ]
INT [ SIGNED | UNSIGNED ]
LARGEINT
NUMERIC [ ( precision [, scale ] ) ]  [ SIGNED   ]
                                      [ UNSIGNED ]
FLOAT [ (precision) ]
REAL
DOUBLE PRECISION
DECIMAL ( precision [, scale ] ) [ SIGNED   ]
                                  [ UNSIGNED ]
DATETIME [ start-date-time TO ] end-date-time
DATE
TIME
TIMESTAMP [(seconds-precision)]
```

The *precision* value is either the length in bytes or the total number of significant digits for numeric data types. The *scale* value defaults to zero if it is omitted for a data type that requires *scale*.

The values for *length*, *precision*, and *scale* depend on the particular SQL data type. See the *NonStop SQL/MP Reference Manual* for descriptions of the various data types.

The PROCEDURE statements follow these restrictions:

Data Type	Parameters	Value Range Restrictions
CHAR	*length*	1 < *length* < 4061
VARCHAR	*length*	1 < *length* < 4059
SMALLINT		*precision* = 5
INT		*precision* = 10
LARGEINT		*precision=19*
FLOAT	*precision* *scale*	0 < *precision* <= 54 0 <= *scale* <= *precision*

Data Type	Parameters	Value Range Restrictions
REAL		*precision* = 22
DOUBLE PRECISION		*precision* = 54
NUMERIC	*precision* *scale*	0 < *precision* <= 18 0 <= *scale* < *precision*
DECIMAL	*precision* *scale*	0 < *precision* <= 18 0 <= *scale* < *precision*
DATE		Format: YEAR to DAY
		Julian restrictions
TIME		Format: HOURS to SECOND
TIMESTAMP	Fractional seconds	6 digits microseconds

All of the preceding data types can be NULLABLE. Only SMALLINT, INT, NUMERIC, and DECIMAL can be either SIGNED or UNSIGNED (the default is SIGNED); LARGEINT can be SIGNED only.

default-value

is one of the following:

```
literal
date-time-literal
SYSTEM
CURRENT
NULL
```

The values for *default-value* depend on the particular SQL data type involved; for example, *date-time-literal* and CURRENT are valid for types DATETIME, DATE, TIME, and TIMESTAMP only. See the *NonStop SQL/MP Reference Manual* for detailed descriptions of appropriate values for these various data types.

parameter-designation

specifies either INPUT or INPUT/OUTPUT. INPUT specifies that the parameter is passed as input to the stored procedure; INPUT/OUTPUT specifies that the parameter is passed as input and is also returned by the procedure. INPUT/OUTPUT is the default.

Considerations – ADD PROCEDURE_COLUMNS

The ADD PROCEDURE_COLUMNS statement:

- Adds one entry in the ZNUPCOL table for each parameter specified

If an error occurs, the actions of ADD PROCEDURE_COLUMNS are rolled back.

Example

The following ADD PROCEDURE_COLUMNS statement adds a parameter definition to the stored procedure named SELECT_DEPT3000, owned by user SQL_DAVE, in the NonStop ODBC Server catalog \TESS.$DATA1.DB:

```
ADD PROCEDURE_COLUMNS tess_data1_db.sql_dave.select_dept3000 &
  tenure INT NULLABLE INPUT/OUTPUT
```

MODIFY PROCEDURE

The MODIFY PROCEDURE statement modifies the entry for a stored procedure in the ZNUPROC table and all entries associated with the procedure in the ZNUPCOL table.

To execute MODIFY PROCEDURE, you must be a privileged user.

```
MODIFY PROCEDURE logical-procedure-name
       PATHMON_NAME pathmon-processname
       SERVERCLASS server-class-name
     [ SERVICE service-name ]
     [ NUM_RESULT_SETS result-set-count ]
     [ RETURN_STATUS { Y | N } ]
     [ MAX_BUFFER_LEN ipc-message-buffer-len ]
     [ REMARKS "remarks-text" ]
```

logical-procedure-name

specifies the name of the stored procedure you want to modify. For a description of the MODIFY PROCEDURE parameters, see the ADD PROCEDURE statement.

Considerations – MODIFY PROCEDURE

The MODIFY PROCEDURE statement:

- Modifies the specified procedure in the ZNUPROC catalog table
- Modifies all entries associated with the procedure in the ZNUPCOL table
- Generates an error if you are not authorized to modify the current user catalog
- If you change the value of a single attribute in MODIFY PROCEDURE, you must restate every attribute-value pair.

If an error occurs, the effects of MODIFY PROCEDURE are rolled back.

Example

```
MODIFY PROCEDURE tess_data1_db.sql_dave.select_dept3000 &
   PATHMON_NAME $empsp &
   SERVICE EMP-DATA-DML &
   NUM_RESULT_SETS 1 &
   MAX_BUFFER_LEN 2000
```

REMOVE PROCEDURE

The REMOVE PROCEDURE deletes the entry for a stored procedure from the ZNUPROC table and all entries associated with the procedure from the ZNUPCOL table. To execute REMOVE PROCEDURE, you must be a privileged user.

```
REMOVE PROCEDURE logical-procedure-name
```

logical-procedure-name

specifies the name of the stored procedure you want to delete.

For Transact-SQL users only, a number appended to the name identifies a procedure in a group of procedures with the same name that you want to delete. If you omit the number, all procedures in the group are deleted.

Considerations – REMOVE PROCEDURE

The REMOVE PROCEDURE statement:

- Deletes the specified procedure from the ZNUPROC and ZNUOBJ catalog tables
- Deletes all entries associated with the procedure from the ZNUPCOL table
- Generates an error if you are not a privileged user

If an error occurs, the effects of REMOVE PROCEDURE are rolled back.

Example

```
REMOVE PROCEDURE tess_data1_db.sql_dave.select_dept3000
```

PROFILE Statements

The following PROFILE statements affect named profile configuration:

- ADD PROFILE
- MODIFY PROFILE
- REMOVE PROFILE

ADD PROFILE

The ADD PROFILE statement defines the configuration of a new named profile. To execute ADD PROFILE, you must be a privileged user.

```
ADD PROFILE profile-name
   [ DEFAULT_DATABASE database-name ]
   [ DEFAULT_SCHEMA schema-name ]
   [ DEFAULT_LOCATION network-volume ]
   [ DEFAULT_SECURITY security-string ]
   [ TRA_MODE_ON { Y | N } ]
   [ TRA_NAME trace-name ]
   [ ACC_MODE_ON { Y | N } ]
   [ ACC_LOGTABLE_NAME acc-name ]
   [ ACC_LEVEL { SESSION | SQL_STATEMENT } ]
   [ GOV_MODE_ON { Y | N } ]
   [ GOV_NAME gov-name ]
   [ QST_MODE_ON { Y | N } ]
   [ SQL_ACCESS_MODE { RO | RW } ]
   [ SQL_CURSOR_MODE { RO | RW } ]
   [ SQL_DIALECT { TDM_CORE | TDM_TSQL } ]
   [ SQL_MAX_STATEMENT_CACHE user-statement-cache-limit ]
   [ SQL_TXN_ISOLATION { 0 | 1 | 2 | 3 } ]
   [ SQL_UNSUPPORTED { E | W | I } ]
   [ OBJ_NAME_CACHE ]
   [ STMT_CACHE_LEVEL ]
   [ CON_MODE_ON { Y | N } ]
   [ CON_NAME control-name ]
   [ CLOSE_TABLES_PER_SESSION { Y | N } ]
   [ UPDATE_SYSTEM_CONFIG ]
```

profile-name

specifies the profile to create; *profile-name* is an alphanumeric string up to 60 characters. It must begin with an alphabetic character, followed by alphabetic, numeric, and underscore characters in length. The name must be unique. DEFAULT is reserved for the profile of the DEFAULT server class.

DEFAULT_DATABASE *database-name*

specifies the default NonStop ODBC Server database for this profile; *database-name* is an alphanumeric string up to 60 characters in length. It must match a valid entry in the ZNSDB mapping table. The default is MASTER.

DEFAULT_SCHEMA *schema-name*

specifies the default NonStop ODBC Server schema to use for qualifying object references in a DML SQL statement. However, it cannot be used for the qualification of the object name for DDL statements. These statements must always use the authenticated process ID which is mapped as an ODBC user for object qualification.

`schema-name` is an alphanumeric string up to 60 characters in length. It must begin with an alphabetic character, followed by alphabetic, numeric, and underscore characters.

The default for this parameter is the logical username of the current user.

`DEFAULT_LOCATION` *`network-volume`*

specifies the default location of objects created using this profile; *`network-volume`* is an alphanumeric string up to 60 characters in length and must be a valid Guardian volume name in the following format:

`[\`*`node.`*`]`*`$volume`*

If you omit *`network-volume`*, objects are placed in the subvolume where the catalog for the current database resides.

`DEFAULT_SECURITY` *`security-string`*

is a four-character alphabetic string that specifies the Guardian read, write, execute, and purge (RWEP) access for NonStop SQL/MP objects created by a user with this profile. This value affects CREATE DATABASE, CREATE TABLE, CREATE VIEW, and CREATE INDEX statements. The default is the default Guardian security for the current user.

For more information about security for NonStop SQL/MP objects, see the “Security” entry in the *NonStop SQL/MP Reference Manual*.

`TRA_MODE_ON { Y | N }`

specifies whether to use an existing trace entity in this profile:

Y	Perform tracing using the existing trace entity specified in *`trace-name.`*
N	Do not perform tracing for this profile.

`TRA_NAME` *`trace-name`*

specifies the name of an existing trace entity to include in this profile; *`trace-name`* is an alphanumeric string up to 60 characters in length. It must begin with an alphabetic character, followed alphabetic, numeric, and underscore characters.

If the trace entity you specify for *`trace-name`* does not already exist, tracing features are not enabled for this profile. See ADD TRACE on page 7-109 to learn how to create a new trace entity.

The NonStop ODBC Server uses the trace entity you specify only if you also specify a value of Y for the TRA_MODE_ON keyword.

`ACC_MODE_ON { Y | N }`

specifies whether to include query accounting in this profile:

Y	Perform accounting and use the existing accounting log table you specify in *`acc-name`*.
N	Do not perform accounting for this profile.

`ACC_LOGTABLE_NAME` *`acc-name`*

specifies the name of the existing accounting log table to use for this profile; *`acc-name`* is an alphanumeric string up to 182 characters in length in the following format:

`database.owner.``ZNUMTRX`

See CREATE ACC_LOG on page 7-40 for information about creating a new accounting log table.

The NonStop ODBC Server uses the accounting log table you specify only if you also specify Y for ACC_MODE_ON. When ACC_MODE_ON is set to Y, if *`acc-name`* is either invalid or NULL, the resource governing environment is not initialized and governing is not enforced.

`ACC_LEVEL { SESSION | SQL_STATEMENT }`

specifies the level of detail for accounting log table entries:

`SESSION`	A single log table entry, created when the connection terminates, represents all activity during that session.
`SQL_STATEMENT`	Each executed SQL statement has a corresponding entry in the accounting log table.

`GOV_MODE_ON { Y | N }`

specifies whether to use an existing governing policy entity for this profile:

Y	Use the governing policy entity you specify in *`gov-name`*.
N	Do not use a governing policy entity for this profile.

`GOV_NAME` *`gov-name`*

specifies the name of the existing governing policy entity to include in this profile; *`gov-name`* is an alphanumeric string up to 60 characters in length. It must begin with an alphabetic character, followed by alphabetic, numeric, and underscore characters.

If the governing policy entity you specify for *`gov-name`* does not already exist, governing features are not enabled for this profile. See ADD GOVERNING on page 7-75 to learn how to create a new governing policy entity.

The NonStop ODBC Server uses the governing policy entity you specify only if you also specify Y for the GOV_MODE_ON keyword.

`QST_MODE_ON { Y | N }`

specifies whether to log query status for each statement executed:

Y	Log query status for each statement executed.
N	Do not log query status for each statement executed.

`SQL_ACCESS_MODE { RO | RW }`

specifies the default locking when the user does not explicitly state the access intentions:

RO	Read-only access; INSERT, UPDATE, or DELETE statements return errors
RW	Read/write access

If you specify RO, any INSERT, UPDATE, or DELETE statements return errors.

`SQL_CURSOR_MODE { RO | RW }`

specifies the default used by the parser/translator to modify cursor definitions when the user does not explicitly state access intentions:

RO	Read-only access
RW	Read/write access

`SQL_DIALECT {` TDM_CORE `|` TDM_TSQL `}`

specifies the SQL dialect that will be used by the client:

TDM_CORE	CORE SQL
TDM_TSQL	Transact-SQL

The client and server must use the same SQL dialect; otherwise, errors such as “bad syntax” can occur.

`SQL_MAX_STATEMENT_CACHE` *`user-statement-cache-limit`*

specifies the maximum number of user statements that can be cached; *`user-statement-cache-limit`* is an INT number that ranges from 1 through 32,767. Zero (0) indicates that the user statement caching mechanism is OFF.

`SQL_TXN_ISOLATION { 0 | 1 | 2 | 3 }`

is a numeric character that specifies the default transaction isolation level:

SQL_TXN_ISOLATION Value		**NonStop SQL/MP Access**
`0`	`SQL_TXN_READ_UNCOMMITTED`	`BROWSE`

1	`SQL_TXN_READ_COMMITTED`	`STABLE`
2	`SQL_TXN_REPEATABLE_READ`	`REPEATABLE`
3	`SQL_TXN_SERIALIZABLE`	`REPEATABLE` (serializable)

`SQL_UNSUPPORTED { E | W | I }`

specifies the action taken by the parser/translator when it detects a syntax error for dialect conflicts or nonconforming syntax:

E	Exceptions are to be reported as errors.
W	Exceptions are to be reported as warnings.
I	Exceptions are not to be reported (that is, ignored)

This option can be useful in speeding up processing in situations where it is safe to ignore syntax errors resulting from features not supported by the NonStop ODBC Server, such as GRANT/REVOKE.

`OBJ_NAME_CACHE`

specifies the cache behavior for object names:

Y	Store object names in cache indefinitely.
N	Do not to store object names in cache.
S	Store object names in cache only during the current session.

If you store object names in cache indefinitely, existing rules for invalidating an entry in the name cache due to nonexistence still apply.

`STMT_CACHE_LEVEL`

is reserved for future use.

`CON_MODE_ON { Y | N }`

specifies whether to use an existing control statement entity in this profile:

Y	Use the control statement entity you specify in *`control-name`*.
N	Do not use a control statement entity.

`CON_NAME` *`control-name`*

specifies the name of an existing control statement entity to include in this profile; *`control-name`* is an alphanumeric string up to 60 characters in length. It must begin with an alphabetic character, followed by alphabetic, numeric, and underscore characters.

`CLOSE_TABLES_PER_SESSION { Y | N }`

specifies whether NonStop SQL/MP tables are closed at the end of a session:

Y	Close tables at the end of the session.
N	Do not close tables at the end of the session. N is the default.

`UPDATE_SYSTEM_CONFIG`

resets the time to update system configuration values to the current time. This option causes running system components to reread their configuration values on the next polling cycle.

Considerations – ADD PROFILE

The ADD PROFILE statement:

- Creates entries in ZNSPROF that define the configuration of the specified profile
- Generates a warning if *`database-name`*, *`schema-name`*, *`trace-name`*, *`gov-name`*, or *`control-name`* cannot be found in the appropriate mapping table
- Generates an error if *`profile-name`* already exists in ZNSPROF
- Generates an error if the system catalog is not customized for the NonStop ODBC Server

Example

```
ADD PROFILE myprofl SQL_DIALECT TDM_CORE SQL_UNSUPPORTED I
```

MODIFY PROFILE

The MODIFY PROFILE statement modifies profile attributes in the ZNSPROF table.

To execute MODIFY PROFILE, you must be authorized to modify the system catalog.

```
MODIFY PROFILE profile-name
   [ DEFAULT_DATABASE database-name ]
   [ DEFAULT_SCHEMA schema-name ]
   [ DEFAULT_LOCATION network-volume ]
   [ DEFAULT_SECURITY security-string ]
   [ TRA_MODE_ON { Y | N } ]
   [ TRA_NAME trace-name ]
   [ ACC_MODE_ON { Y | N } ]
   [ ACC_LOGTABLE_NAME acc-name ]
   [ ACC_LEVEL { SESSION | SQL_STATEMENT } ]
   [ GOV_MODE_ON { Y | N } ]
   [ GOV_NAME gov-name ]
   [ QST_MODE_ON { Y | N } ]
   [ SQL_ACCESS_MODE { RO | RW } ]
   [ SQL_CURSOR_MODE { RO | RW } ]
   [ SQL_DIALECT { TDM_CORE | TDM_TSQL } ]
   [ SQL_MAX_STATEMENT_CACHE user-statement-cache-limit ]
   [ SQL_TXN_ISOLATION { 0 | 1 | 2 | 3 } ]
   [ SQL_UNSUPPORTED { E | W | I } ]
   [ OBJ_NAME_CACHE { N | Y | S } ]
   [ STMT_CACHE_LEVEL cache-level ]
   [ CON_MODE_ON { Y | N } ]
   [ CON_NAME control-name ]
   [ CLOSE_TABLES_PER_SESSION { Y | N } ]
   [ UPDATE_SYSTEM_CONFIG ]
```

profile-name

specifies the profile you want to modify. For a description of the MODIFY PROFILE parameters, see the ADD PROFILE statement.

Considerations – MODIFY PROFILE

The MODIFY PROFILE statement:

- Modifies the entry for the specified profile name in the ZNSPROF table
- Generates an error if the profile name is referenced in any logical username record in ZNSUS
- Generates a warning if *database-name*, *schema-name*, *trace-name*, *gov-name*, or *control-name* cannot be found in the appropriate mapping table
- Generates an error if you are not authorized to modify the system catalog
- Generates an error if the system catalog is not customized for the NonStop ODBC Server

Example

```
MODIFY PROFILE SQL_DIALECT TDM_TSQL SQL_UNSUPPORTED W
```

REMOVE PROFILE

The REMOVE PROFILE statement removes a profile entry from the ZNSPROF table.

To execute REMOVE PROFILE, you must be a privileged user.

```
REMOVE PROFILE profile-name [ UPDATE_SYSTEM_CONFIG ]
```

`profile-name`

specifies the profile you want to remove.

Considerations – REMOVE PROFILE

The REMOVE PROFILE statement:

- Removes the entry for *`profile-name`* from ZNSPROF
- Generates an error if *`profile-name`* is referenced in any logical username record in ZNSUS
- Generates an error if you are not a privileged user
- Generates an error if the system catalog is not customized for the NonStop ODBC Server

Example

```
REMOVE PROFILE myprofl
```

Query Status Log (QST_LOG) Statements

Use the following statements for NonStop ODBC Server query status log (QST_LOG) tables:

- CREATE QST_LOG
- DROP QST_LOG

CREATE QST_LOG

The CREATE QST_LOG statement creates a new query status log table named ZNUQST and registers the table as a NonStop ODBC Server object in the specified database.

```
CREATE QST_LOG IN database-name [ SECURE security-string ]
```

`IN` *`database-name`*

is an alphanumeric string up to 60 characters in length that specifies the name of the database in which to register the query status log table. The database you specify must be a customized NonStop ODBC Server database.

`SECURE` *`security-string`*

is a four-character alphabetic string that specifies the Guardian read, write, execute, and purge (RWEP) access for the query status log table. For more information about security for NonStop SQL/MP objects, see the “Security” entry in the *NonStop SQL/MP Reference Manual*.

Considerations – CREATE QST_LOG

The CREATE QST_LOG statement:

- Generates an error if *`database-name`* is not a customized NonStop ODBC Server database
- Generates an error if the associated SQL CREATE TABLE statement fails for either resource or privilege reasons.

Example

```
CREATE QST_LOG IN sf_admin_cat SECURE NONO
```

DROP QST_LOG

The DROP QST_LOG statement drops an existing query status log table. To execute DROP QST_LOG, you must be a privileged user.

```
DROP QST_LOG  FROM database-name
```

`FROM` *`database-name`*

specifies the name of the database from which to drop the query status log table; *`database-name`* is an alphanumeric string up to 60 characters in length. This database must be a customized NonStop ODBC Server database.

Considerations – DROP QST_LOG

The DROP QST_LOG statement:

- Removes the mapping entry for the log table in *`database-name`*
- Preserves the entry in ZNSPROF for the log table
- Generates an error if you do not have sufficient privileges to execute DROP QST_LOG
- Generates an error if the associated SQL DROP TABLE statement fails for either resource or privilege reasons

Example

```
DROP QST_LOG FROM sf_admin_cat
```

Resource Governing Statements

Use the following statements to configure and manage resource governing policy entities that control various aspects of SQL statement execution:

- ADD GOVERNING
- MODIFY GOVERNING
- REMOVE GOVERNING

ADD GOVERNING

The ADD GOVERNING statement creates a new governing policy entity or adds one or more attribute-value pairs to an existing entity. To execute ADD GOVERNING, you must be a privileged user.

```
ADD GOVERNING gov-name
              GOV_ATTRIBUTE gov-attribute
              LIMIT_VALUE limit-value
              GOV_ACTION gov-action
            [ LOG_QST_ON { N | Y } ]
            [ UPDATE_SYSTEM_CONFIG ]
```

gov-name

specifies the name of the governing policy entity; *gov-name* is an alphanumeric string up to 60 characters in length that must begin with an alphabetic character and can be followed by alphabetic, numeric, and underscore characters.

If *gov-name* does not already exist, the ADD GOVERNING statement creates a new governing policy entity. If *gov-name* already exists in the ZNSGOV table, the ADD GOVERNING statement appends the new attribute-value pair or pairs you specify to the existing governing policy entity.

If any limit is reached, the NonStop ODBC Server takes the *gov-action* you specify for that *gov-attribute* and *limit-value*.

`GOV_ATTRIBUTE` *`gov-attribute`*

specifies one of the following aspects of query execution to apply to *`gov-action`*:

ELAPSED_TIME	Maximum number of seconds in wall-clock time to wait for an SQL statement to complete, measured as follows: ● For an EXECUTE IMMEDIATE statement, the elapsed time begins when the statement executes and ends when the statement completes execution. ● For PREPARE and EXECUTE statements, the elapsed time begins when the PREPARE statement executes and ends when the statement completes execution; it includes the time required for any DESCRIBE, OPEN, or FETCH statements. Valid with CONTINUE, COMMIT, ROLLBACK, and PRIORITY [++ **|** --] *n* governing actions.
ESTIMATED_COST	Maximum estimated cost that is acceptable for a query. Valid with STOP and PRIORITY [++ **|** --] *n* governing actions.
ROWS_ACCESSED	Maximum number of rows to access during a SELECT statement. Valid with CONTINUE, COMMIT, ROLLBACK, and PRIORITY [++ **|** --] *n* governing actions.
ROWS_USED	Maximum number of rows to return for a SELECT statement. Valid with CONTINUE, COMMIT, ROLLBACK, and PRIORITY [++ **|** --] *n* governing actions.
EXECUTION_TIME	Maximum number of seconds in CPU time to wait for an SQL statement to complete, measured from the time the statement is executed. Valid with CONTINUE, COMMIT, ROLLBACK, and PRIORITY [++ **|** --] *n* governing actions.

`LIMIT_VALUE` *`limit-value`*

specifies an INT(8) number greater than or equal to -1 that specifies the maximum resource amount to use for a particular query.   -1 indicates there is no limit.

`GOV_ACTION` *`gov-action`*

specifies the action you want the NonStop ODBC Server to take when a query exceeds the *`limit-value`*. *`gov-action`* can be only one of the following:

COMMIT	Stop and commit the transaction.
CONTINUE	Continue without any changes.
PRIORITY	Change the query priority as follows:

"PRIORITY ++*n*"	raises the priority by *n*
"PRIORITY --*n*"	lowers the priority by *n*
"PRIORITY *n*"	changes the priority to *n*

To specify a value for PRIORITY that includes blank spaces, enclose both the keyword and value in quotation marks.

ROLLBACK	Stop and roll back the transaction.
STOP	Return an error indicating the statement cannot be executed due to policies on exceeding resources (applies only to the ESTIMATED_COST option).

`LOG_QST_ON { Y | N }`

indicates whether to log the query status for each statement executed:

Y	Log the query status.
N	Do not log the query status. N is the default.

`UPDATE_SYSTEM_CONFIG`

resets the time to update system configuration values to the current time. This option causes running system components to reread their configuration values on the next polling cycle.

Considerations – ADD GOVERNING

The ADD GOVERNING statement:

- Creates a new entry for the governing policy entity in the ZNSGOV system table or adds the new set of attribute value-pairs to an existing entity
- Generates an error if you are not a privileged user

After you add a governing policy, you must also set the GOV_MODE_ON and GOV_NAME options using the ADD PROFILE or MODIFY PROFILE statement to update the respective profile to enable resource governing for the policy. For more information about these options, see ADD PROFILE on page 7-66.

Examples

The following ADD GOVERNING statement creates a governing policy entity named q3 that lowers the execution priority to 50 if an SQL statement takes more than 60 minutes to complete:

```
ADD GOVERNING q3 GOV_ATTRIBUTE EXECUTION_TIME &
   LIMIT_VALUE 60 GOV_ACTION "PRIORITY 50"
```

The following statement adds a new *gov-attribute, limit-value*, and *gov-action* combination to the existing governing policy entity named q3:

```
ADD GOVERNING q3 GOV_ATTRIBUTE ESTIMATED_COST &
   LIMIT_VALUE 20 GOV_ACTION STOP
```

If you execute both ADD GOVERNING statements shown so far in this example, q3 does not execute an SQL statement if the estimated cost is greater than 20 and lowers the execution priority to 50 if the statement takes more than 60 minutes to complete.

A single *gov-attribute* can have more than one *limit-value* and *gov-action* combination associated with it. The following statement modify q3 to commit the transaction if the SQL statement takes more than 90 minutes to complete.

```
ADD GOVERNING q3 GOV_ATTRIBUTE EXECUTION_TIME &
   LIMIT_VALUE 90 GOV_ACTION COMMIT
```

If you execute all ADD GOVERNING statements shown in this example, q3 does not execute an SQL statement if the estimated cost is greater than 20, lowers the execution priority to 50 if the statement takes more than 60 minutes to complete, and stops and commits the transaction if the statement takes more than 90 minutes to complete.

MODIFY GOVERNING

The MODIFY GOVERNING statement modifies values for existing attributes in an existing governing policy entry. (MODIFY GOVERNING modifies values for existing attributes. To add a new set of attribute-value pairs to an existing governing policy entity, use ADD GOVERNING.) To execute MODIFY GOVERNING, you must be a privileged user.

```
MODIFY GOVERNING gov-name
                 GOV_ATTRIBUTE gov-attribute
                 LIMIT_VALUE limit-value
               [ GOV_ACTION gov-action ]
               [ LOG_QST_ON { N | Y } ]
               [ UPDATE_SYSTEM_CONFIG ]
```

gov-name

is an alphanumeric string up to 60 characters in length that specifies the name of the governing policy entity you want to modify.

For a description of the MODIFY GOVERNING statement parameters, see ADD GOVERNING on page 7-75.

Considerations – MODIFY GOVERNING

The MODIFY GOVERNING statement:

- Modifies the corresponding entry in the ZNSGOV system table
- Generates an error if *gov-name* is not a valid NonStop ODBC Server governing policy entity
- Generates an error if you are not a privileged user

Example

The following example shows how to use MODIFY GOVERNING to change *limit-value* from 90 to 100 minutes for the *gov-attribute* EXECUTION_TIME and *gov-action* COMMIT in the existing governing policy entity q3:

```
MODIFY GOVERNING q3 GOV_ATTRIBUTE EXECUTION_TIME &
   LIMIT_VALUE 100 GOV_ACTION COMMIT
```

REMOVE GOVERNING

The REMOVE GOVERNING statement removes one or more existing governing policy entities or a single attribute-value pair in an existing governing policy entity. To execute REMOVE GOVERNING, you must be a privileged user.

```
REMOVE GOVERNING gov-name
                 [ GOV_ATTRIBUTE gov-attribute ]
                 [ LIMIT_VALUE limit-value ]
                 [ UPDATE_SYSTEM_CONFIG ]
```

gov-name

is an alphanumeric string up to 60 characters in length that specifies the name of the governing policy entity. If you specify only *gov-name*, all governing policy entities with this name are removed.

GOV_ATTRIBUTE *gov-attribute*

specifies one of the following aspects of query execution: ELAPSED_TIME, ESTIMATED_COST, ROWS_ACCESSED, ROWS_USED, and EXECUTION_TIME. For a description of these parameters, see the ADD GOVERNING statement.

LIMIT_VALUE *limit-value*

specifies the value for the attribute-value pair you want to remove.

`UPDATE_SYSTEM_CONFIG`

resets the time to update system configuration values to the current time. This option causes running system components to reread their configuration values on the next polling cycle.

Considerations – REMOVE GOVERNING

The REMOVE GOVERNING statement:

- Deletes the corresponding entry or entries in the system table ZNSGOV
- Generates an error if you are not authorized to modify the system catalog

Example

```
REMOVE GOVERNING q3 UPDATE_SYSTEM_CONFIG
```

SERVERCLASS Statements

Use the following statements to configure and manage server class definitions and processes:

- ADD SERVERCLASS
- MODIFY SERVERCLASS
- REMOVE SERVERCLASS
- START SERVERCLASS
- STOP SERVERCLASS

ADD SERVERCLASS

The ADD SERVERCLASS statement creates a new server class configuration entity in the ZNSSER table. To execute ADD SERVERCLASS, you must be authorized to modify the system catalog.

```
ADD SERVERCLASS server-name
      G_USERNAME user-name
    [ PRIORITY nos-priority ]
    [ CPU_LIST (cpu-list) ]
    [ AVAILABLE_SERVERS available-servers ]
    [ MAX_SERVERS max-servers ]
    [ PROFILE profile-name ]
    [ INIT_HEAP_SIZE_KB init-heap-size-kb ]
    [ MAX_HEAP_SIZE_KB max-heap-size-kb ]
    [ IDLE_DELETE_DELAY_SEC delete-delay-seconds ]
    [ NOS_CREATE_OPTIONS nos-create-options ]
    [ NOS_DEBUG_OPTIONS nos-debug-options ]
    [ NOS_RUN_OPTIONS "nos-run-options" ]
    [ SWAPVOL swapvol ]
    [ LOGIN_TIMEOUT_SEC seconds ]
    [ CANCEL_TIMEOUT_SEC seconds ]
    [ UPDATE_SYSTEM_CONFIG ]
```

server-name

is an alphanumeric string up to 32 characters in length that specifies the name of the server class definition you want to add.

G_USERNAME *user-name*

is an alphanumeric character string up to 17 characters in length that specifies the Guardian user name of the owner of this server class in the following format:

```
group.user
```

PRIORITY *nos-priority*

is an INT(2) number in the range -1 through 199 that specifies the priority of the NOS process. A -1 value means that the NOS process inherits the priority of the calling process.

CPU_LIST *cpu-list*

is a an alphanumeric string up to 37 characters in length that identifies the processors in which to run the server process for this server class. It has the following format:

(*cpu-number*[,*cpu-number*]...)

Enclose the one or more values you specify for *cpu-list* in parentheses. If you specify a value of -1 for *cpu-list*, the PROCESS_CREATE_ procedure automatically assigns the primary CPU.

`AVAILABLE_SERVERS` *`available-servers`*

is an unsigned integer number in the range 0 through 254 that specifies the number of server processes to start for this server class when SCS is started; *`available-servers`* must be less than or equal to the value you specify for *`max-servers`*.

`MAX_SERVERS` *`max-servers`*

is an unsigned integer number in the range 0 through 254 that specifies the maximum number of server processes to start for this server class when SCS is started; *`max-servers`* must be greater than or equal to the value you specify for *`available-servers`*.

`PROFILE` *`profile-name`*

is an alphanumeric character string up to 60 characters in length that specifies the name of the profile for this server class; *`profile-name`* must begin with an alphabetic character, followed by alphabetic, numeric, and underscore characters.

`INIT_HEAP_SIZE_KB` *`init-heap-size-kb`*

is an INT(2) number in the range 500 through 32767 that specifies an initial heap size in kilobytes for the server class process. The value you specify determines the initial number of kilobytes in a memory segment used to cache SQL statements.

`MAX_HEAP_SIZE_KB` *`max-heap-size-kb`*

is an INT(2) number in the range 500 through 32767 that specifies the maximum heap size in kilobytes for the server class process. The value you specify determines the maximum number of kilobytes for a memory segment if additional memory is needed to cache an SQL statement. The value you specify for *`max-heap-size-kb`* must be greater than *`init-heap-size-kb`*.

`IDLE_DELETE_DELAY_SEC` *`delete-delay-seconds`*

is an INT(4) value in the range 0 through 2,147,483,647, where a value greater than 0 specifies the number of seconds to wait after the connection terminates before reducing the number of running server processes to *`available-servers`*; a value of 0 specifies that SCS not delete the NOS process after the connection terminates. The default is 0 (zero).

`NOS_CREATE_OPTIONS` *`nos-create-options`*

is an INT(2) number that specifies create options for the NOS process. It can be one of the following :

6	specifies to inherit DEFINEs from the calling process, but ignore DEFINEs stored in ZNSDEF
22	specifies to inherit DEFINEs from the calling process, and include at least one DEFINE from ZNSDEF
23	specifies to inherit DEFINEs from the calling process, include at least one DEFINE from ZNSDEF

`NOS_DEBUG_OPTIONS` *`nos-debug-options`*

is an INT(2) number that specifies debugging options for the NOS process. It can be one of the following:

0	specifies not to use debugging features
4	specifies to create a SAVEABEND file when NOS abends
8	specifies to enter DEBUG mode at _MAIN
9	specifies to enter INSPECT mode at _MAIN and create a SAVEABEND file upon trap
12	specifies to enter DEBUG mode at _MAIN, ignore the program default for debugging, and create a SAVEABEND file upon trap
13	specifies to enter INSPECT mode at _MAIN, ignore the program default for debugging, and create a SAVEABEND file upon trap

`NOS_RUN_OPTIONS` *`nos-run-options`*

specifies run options for the NOS process; *`nos-run-options`* is an alphanumeric string up to 240 characters in length enclosed by double quotes ("). If you specify more than one option, separate each option with a space. *`nos-run-options`* can have these values:

`FREE_RESOURCES=`*`n`*

specifies that ESP and SORTPROG processes used by an SQL statement are released when the statement is closed if the cost of the statement is greater than 10 to the *n*th power.

`AGGREGATE_AS_INT={Y|N}`

Y causes the result set of each aggregate function to be converted to an integer value. The NOS process translates an aggregate function as:

```
CAST (aggregate_function) AS INTEGER)
```

`PRIMARY_EXTENT=`*n*   `SECONDARY_EXTENT=`*n*   `MAX_EXTENTS=`*n*

specifies values for the primary and secondary extent sizes and the maximum number of extents used in the creation of SQL tables and indexes. The same values are used for both tables and indexes. You can specify none, one, two, or all options.

If you do not specify any options, the NOS process uses the NonStop ODBC default values as: `PRIMARY_EXTENT=16`, `SECONDARY_EXTENT=64`, and `MAX_EXTENTS=940` (allowing tables and indexes up to 123 MB).

If you omit an option, the NOS process uses the SQL default value for that option as: `PRIMARY_EXTENT=16`, `SECONDARY_EXTENT=64`, and `MAX_EXTENTS=160`.

These options allow you to create tables and indexes larger than 123 MB. For example, if `PRIMARY_EXTENT=100`, `SECONDARY_EXTENT=800`, and `MAX_EXTENTS=940`, you can create a table or index up to 1.54 GB.

`AUDITED_TABLES={Y|N}`

Y specifies that CREATE TABLE commands executed by the serverclass have the AUDIT clause appended. Y is the default.

N specifies that CREATE TABLE commands executed by the serverclass have the NO AUDIT clause appended. (Although the CREATE INDEX and CREATE VIEW commands have no AUDIT clause, indexes and views are also created as unaudited objects when this option causes the underlying tables to be unaudited.)

When N is specified, NOS does not begin or end transactions for DML commands; for example, INSERT or SELECT will be executed without a transaction.

`GOV_ERROR = { Y | N }`

lets you choose between the `SQL_SUCCESS_WITH_INFO` message and the `SQL_ERROR` message when the resource governing limit is reached.

Y specifies that, when the resource governing limit is reached, NOS treats the situation as an error and replies with a `SQL_ERROR` message.

N specifies that, when the resource governing limit is reached, NOS treats the situation as a warning and sends a `SQL_SUCCESS_WITH_INFO` message.

The text is the same in both the error and the warning messages.

`SWAP` *`swap-vol`*

is an alphabetic character string up to 24 characters in length that specifies the name of a swap volume for the server class process in the following format:

`[\`*`node.`*`]$`*`volume`*

`LOGIN_TIMEOUT_SEC` *`seconds`*

is an INT(4) value that specifies the time in seconds SCS waits before sending a logon denied message to a client who is trying to logon to a NonStop ODBC Server. This option applies only when the `MAX_SERVERS` option has reached its limit. SCS measures *`seconds`* from when it first receives the client's logon request. A value less than or equal to zero (0) disables the option, The default is -1.

`CANCEL_TIMEOUT_SEC` *`seconds`*

is an INT(2) value that specifies the time in seconds that SCS waits before stopping a NonStop ODBC Server process after an SQLCancel request.

Use the SQLCancel feature as follows:

1. Set the CANCEL_TIMEOUT_SEC attribute to 0 or more seconds.
2. Set the SQLSetStmtOption option on the client: set SQL_ASYNC_ENABLE to SQL_ASYNC_ENABLE_ON.
3. Call either SQLPrepare, SQLPrepare followed by SQLExecute, or SQLExecDirect for a statement such as insert, update, delete, or select.
4. Make repeated calls to SQLPrepare, SQLExecute, or SQLExecDirect to determine if the function is still processing; if it is still processing, it returns SQL_STILL_EXECUTING.
5. Call the SQLCancel function to cancel the SQL processing. SQLCancel should return SQL_SUCCESS and trigger a timer for the server, which times out after the time specified for CANCEL_TIMEOUT_SEC has elapsed.
6. Continue calling SQLPrepare, SQLExecute, or SQLExecDirect until the function returns either SQL_SUCCESS or SQL_ERROR with a specific error communication link failure.
    - SQL_SUCCESS indicates that the SQL processing could not be canceled because it finished before the time specified for CANCEL_TIMEOUT_SEC had elapsed.
    - SQL_ERROR with a communication link failure indicates that the SQL processing has been canceled and that the application cannot proceed until a new connection is made. (Because the SQL processing took longer than the amount of time specified for CANCEL_TIMEOUT_SEC, the SCS process terminated the NOS process that was processing the SQL statement. SQL_ERROR indicates that the termination was successful, and a communication link failure indicates that the connection between the client and server has been terminated. A new connection must be made available for communication between client and server to proceed.)

△ **Caution.** If the NOS process is terminated while AUTOCOMMIT is on, there is a remote possibility that data corruption might occur. This might happen if the SCS process terminates the NOS process between the time when the NOS process ends the transaction and the time when it replies to the client.

`UPDATE_SYSTEM_CONFIG`

resets the time to update system configuration values to the current time. This option causes running system components to reread their configuration values on the next polling cycle.

Considerations – ADD SERVERCLASS

The ADD SERVERCLASS statement also creates a new entry in the ZNSMAP table that maps the new server class entity to *`scs-processname`*. If you do not specify *`scs-processname`*, a mapping entry is not created (and no warning issued). ADD SERVERCLASS generates a warning if *`scs-processname`* is not a valid NonStop ODBC Server configuration entity in the ZNSSCS table.

If you do not specify a value for an optional attribute, ADD SERVERCLASS either uses a default value from the ZNSCFG table or a NULL value.

Example

```
ADD SERVERCLASS alpha &
   CPU_LIST (2,4,7) &
   AVAILABLE_SERVERS 2 &
   MAX_SERVERS 10 &
   PROFILE prof3 &
   INIT_HEAP_SIZE 1000 &
   MAX_HEAP_SIZE 3500 &
   IDLE_DELETE_DELAY_SEC 120 &
   NOS_CREATE_OPTIONS 22 &
   NOS_RUN_OPTIONS "FREE_RESOURCES=5 &
      PRIMARY_EXTENT=100 SECONDARY_EXTENT=800 MAX_EXTENTS=940" &
   NOS_DEBUG_OPTIONS 13
```

MODIFY SERVERCLASS

MODIFY SERVERCLASS modifies one or more of the attribute values for an existing server class configuration entity in the ZNSSER table.

To execute MODIFY SERVERCLASS, you must be a privileged user.

```
MODIFY SERVERCLASS server-name
       [ PRIORITY nos-priority ]
       [ CPU_LIST (cpu-list) ]
       [ AVAILABLE_SERVERS available-servers ]
       [ MAX_SERVERS max-servers ]
       [ G_USERNAME user-name ]
       [ PROFILE profile-name ]
       [ INIT_HEAP_SIZE_KB init-heap-size-kb ]
       [ MAX_HEAP_SIZE_KB max-heap-size-kb ]
       [ IDLE_DELETE_DELAY_SEC delete-delay-seconds ]
       [ NOS_CREATE_OPTIONS create-options ]
       [ NOS_DEBUG_OPTIONS nos-debug-options ]
       [ NOS_RUN_OPTIONS "nos-run-options" ]
       [ SWAPVOL swapvol ]
       [ LOGIN_TIMEOUT_SEC seconds ]
       [ CANCEL_TIMEOUT_SEC seconds ]
       [ UPDATE_SYSTEM_CONFIG ]
```

server-name

specifies the name of the server class definition you want to modify.

For a description of the MODIFY SERVERCLASS parameters, see the ADD SERVERCLASS statement.

Considerations – MODIFY SERVERCLASS

The MODIFY SERVERCLASS statement generates an error if you are not a privileged user or a warning if *scs-processname* is not a valid NonStop ODBC Server configuration entity found in ZNSSCS.

If you do not specify a value for an optional attribute, MODIFY SERVERCLASS either uses a default value from the ZNSCFG table or a NULL value.

Example

```
MODIFY SERVERCLASS alpha AVAILABLE_SERVERS 4 PROFILE prof1
```

REMOVE SERVERCLASS

REMOVE SERVERCLASS removes a configuration entry for the specified server class in the ZNSSER table. To execute REMOVE SERVERCLASS, you must be a privileged user.

```
REMOVE SERVERCLASS server-name [UPDATE_SYSTEM_CONFIG]
```

server-name

specifies the name of the server class definition you want to remove.

Considerations – REMOVE SERVERCLASS

The REMOVE SERVERCLASS statement generates an error if you are not a privileged user.

Example

```
REMOVE SERVERCLASS alpha
```

START SERVERCLASS

The START SERVERCLASS statement starts a specified server class. To execute START SERVERCLASS, you must be a privileged user.

```
START SERVERCLASS server-name OF scs-processname
                 [ UPDATE_SYSTEM_CONFIG ]
```

server-name

specifies the name of the server class definition you want to start.

scs-processname

specifies the name of the SCS process associated with the server class definition.

Example

```
START SERVERCLASS alpha
```

STOP SERVERCLASS

The STOP SERVERCLASS statement stops a specified server class. The NonStop ODBC server stops the server class gracefully after all client connections have been stopped. (To restart the server class, use the START SERVERCLASS statement.) To execute STOP SERVERCLASS, you must be a privileged user.

```
STOP SERVERCLASS server-name OF scs-processname
                 [ UPDATE_SYSTEM_CONFIG ]
```

server-name

specifies the name of the server class definition you want to stop.

scs-processname

specifies the name of the SCS process associated with the server class definition.

Example

```
STOP SERVERCLASS alpha
```

Server Class Mapping (SMAP) Statements

Use the following statements to map a server class to an SCS configuration entity:

- ADD SMAP
- MODIFY SMAP
- REMOVE SMAP

ADD SMAP

The ADD SMAP creates a new mapping between an existing server class and an SCS configuration entity.

To execute ADD SMAP, you must be authorized to modify the system catalog.

```
ADD SMAP scs-processname server-name
         [ USE_AS_DEFAULT { Y | N } ]
         [ UPDATE_SYSTEM_CONFIG ]
```

scs-processname

is an alphabetic string up to 15 characters in length that identifies a valid SCS process in the following format:

[\node.]*$process-name*

server-name

is an alphanumeric string up to 32 characters in length that identifies the name of the server class configuration entity to associate with the specified SCS configuration entity.

USE_AS_DEFAULT { Y | N }

specifies whether to define *server-name* as the default for *scs-processname*. You can designate only one *scs-processname* to *server-name* mapping entry as the designated default server class.

UPDATE_SYSTEM_CONFIG

resets the time to update system configuration values to the current time. This option causes running system components to reread their configuration values on the next polling cycle.

Considerations – ADD SMAP

The ADD SMAP statement:

- Creates a new entry in ZNSMAP that maps a server class configuration entity to an SCS configuration entity

- Generates a warning if *`scs-processname`* is not a valid NonStop ODBC Server SCS configuration entity found in ZNSSCS
- Generates a warning if *`server-name`* is not a valid NonStop ODBC Server server class configuration entity found in ZNSSER
- Generates an error if you are not a privileged user

Example

```
ADD SMAP myscs alpha USE_AS_DEFAULT Y UPDATE_SYSTEM_CONFIG
```

MODIFY SMAP

The MODIFY SMAP statement changes an existing mapping between a server class configuration entity and an SCS configuration entity. To execute MODIFY SMAP, you must be a privileged user.

```
MODIFY SMAP scs-processname server-name
         [ USE_AS_DEFAULT { Y | N } ]
         [ SHUTDOWN_SER   { Y | N } ]
         [ UPDATE_SYSTEM_CONFIG ]
```

`scs-processname`

is an alphabetic string up to 15 characters in length that identifies a valid SCS process in the following format:

`\`*`node.$process-name`*

`ser-name`

is an alphanumeric string up to 32 characters in length that identifies the name of the server class configuration entity associated with the specified SCS configuration entity.

`USE_AS_DEFAULT { Y | N }`

specifies whether to define *`server-name`* as the default for *`scs-processname`*. You can designate only one *`scs-processname`* to *`server-name`* mapping entry as the designated default server class.

`SHUTDOWN_SER { Y | N }`

specifies to *`scs-processname`* whether to shut down *`server-name`* as each current client connection terminates. The NOS server instance is also stopped, and no new instances are started for the *`server-name`* server class. The default is N.

`UPDATE_SYSTEM_CONFIG`

resets the time to update system configuration values to the current time. This option causes running system components to reread their configuration values on the next polling cycle.

Considerations – MODIFY SMAP

The MODIFY SMAP statement:

- Modifies an existing entry in ZNSMAP that maps a server class configuration entity to an SCS configuration entity
- Generates a warning if *`scs-processname`* is not a valid NonStop ODBC Server SCS configuration entity found in ZNSSCS
- Generates a warning if *`server-name`* is not a valid NonStop ODBC Server server class configuration entity found in ZNSSER
- Generates an error if you are not a privileged user

Example

```
MODIFY SMAP myscs alpha SHUTDOWN_SER Y
```

REMOVE SMAP

The REMOVE SMAP statement removes a mapping between an existing server class configuration entity and SCS configuration entity. To execute REMOVE SMAP, you must be a privileged user.

```
REMOVE SMAP scs-processname server-name
            [ UPDATE_SYSTEM_CONFIG ]
```

`scs-processname`

is an alphabetic string up to 15 characters in length that specifies a valid SCS process in the following format:

`[\`*`node.`*`]`*`$process-name`*

`server-name`

is an alphanumeric string up to 32 characters in length that identifies the name of the server class configuration entity associated with the specified SCS configuration entity.

`UPDATE_SYSTEM_CONFIG`

resets the time to update system configuration values to the current time. This option causes running system components to reread their configuration values on the next polling cycle.

Considerations – REMOVE SMAP

The REMOVE SMAP statement:

- Deletes the entry in ZNSMAP that maps *`scs-processname`* to *`server-name`*
- Generates an error if *`scs-processname`* is not a valid NonStop ODBC Server SCS configuration entity found in ZNSSCS
- Generates an error if *`server-name`* is not a valid NonStop ODBC Server server class configuration entity found in ZNSSER
- Generates an error if *`scs-processname`* and *`server-name`* are not mapped to each other and represented by an entry in ZNSUMAP
- Generates an error if you are not a privileged user

Example

```
REMOVE SMAP myscs alpha
```

SQL Communications Subsystem (SCS) Statements

Use the following statements to control how SCS configures and manages a dependent process:

- ADD SCS
- MODIFY SCS
- REMOVE SCS
- START SCS
- STATUS SCS
- STOP SCS

ADD SCS

The ADD SCS statement creates a new SCS named configuration with a set of attribute values in the ZNSSCS table.

To execute ADD SCS, you must be a privileged user.

```
ADD SCS scs-processname
    [ JOB_ID job-id ]
    [ SCS_OBJECT scs-object-file-name ]
    [ SCS_LIBRARY_FILE scs-library-file-name ]
    [ SCS_PRIORITY scs-priority ]
    [ CPU_PRIMARY cpu-primary ]
    [ CPU_BACKUP cpu-backup ]
    [ SWAPVOL swapvol ]
    [ DATAPAGES datapages ]
    [ EXT_SWAPFILE ext-swapvol ]
    [ SCS_CREATE_OPTIONS scs-create-options ]
    [ EMIT_EVENTS { N | Y } ]
    [ MEMORY_CHECK { N | Y } ]
    [ HOMETERM home-term-name ]
    [ IN_FILE input-file-name ]
    [ OUT_FILE output-file-name ]
    [ ERR_FILE error-file-name ]
    [ IN_BUFFER_SIZE_B input-buffer-size ]
    [ OUT_BUFFER_SIZE_B output-buffer-size ]
    [ DEFAULT_VOLUME [[\node.]$volume.]subvolume ]
    [ SCS_DEBUG_OPTIONS scs-debug-options ]
    [ SCS_RUN_OPTIONS "scs-run-options" ]
    [ NOS_OBJECT nos-object-file-name ]
    [ NOS_LIBRARY_FILE nos-library-file-name ]
    [ NOSUTIL_OBJECT nosutil-object-file-name ]
    [ NOSUTIL_LIBRARY_FILE nosutil-library-file-name ]
    [ NOSUTIL_PRIORITY nosutil-priority ]
    [ NOSUTIL_CPU nosutil-cpu ]
    [ NOSUTIL_CREATE_OPTIONS nosutil-create-options ]
    [ NOSUTIL_DEBUG_OPTIONS nosutil-debug-options ]
    [ NOSUTIL_RUN_OPTIONS "nosutil-run-options" ]
    [ UPDATE_SYSTEM_CONFIG ]
```

scs-processname

specifies a Guardian process name for the SCS process.

JOB_ID *job-id*

an INT(2) number greater than or equal to -1 that identifies the job to create for the new process. *job-id* is used for scheduling purposes.

`SCS_OBJECT` *`scs-object-file-name`*

specifies the physical location of the SCS object file; it is an alphabetic string up to 35 characters in length in the following format:

`[\`*`node.`*`]`*`$volume.subvolume.filename`*

`SCS_LIBRARY_FILE` *`scs-library-file-name`*

specifies the name of a user library for the new process; it is an alphabetic string up to 35 characters in length. Use this parameter to specify a library other than the default library.

`SCS_PRIORITY` *`scs-priority`*

is an INT(2) number in the range of -1 through 199 that specifies the execution priority of the created process. A value of -1 means that the process inherits the priority of the calling process.

`CPU_PRIMARY` *`cpu-primary`*

is an INT(2) number that specifies the processor in which to run the primary SCS process. The range depends on the number of processors on your node. You cannot specify the same value for *`cpu-primary`* as for *`cpu-backup`*. If you omit this parameter or specify NULL for *`cpu-primary`*, the SCS primary process will be started in the same CPU as the calling process (which is usually NOSUTIL).

`CPU_BACKUP` *`cpu-backup`*

is an INT(2) number that specifies the CPU in which to run the backup SCS process. The range depends on the number of CPUs on your node. You cannot specify the same value for *`cpu-backup`* as for *`cpu-primary`*. You must specify a valid CPU number to have a backup SCS process. If you omit this parameter or specify NULL for *`cpu-backup`*, a backup SCS process will not be started.

`SWAPVOL` *`swap-vol`*

specifies the swap volume for the user data stack segment of the process; it is an alphabetic string up to 17 characters in length in the following format:

`[\`*`node.`*`]`*`volume`*

`DATAPAGES` *`datapages`*

is an INT(2) number that specifies the size of the buffer that an SCS and NOS process use to communicate.

`EXT_SWAPFILE` *`ext-swapfile`*

specifies a swap file for the default extended data segment of the process; it is an alphabetic string up to 35 characters in length in the following format:

`[\`*`node.`*`]`*`$volume.subvolume.filename`*

`SCS_CREATE_OPTIONS` *`scs-create-options`*

is an INT(2) number that determines the environment of the SCS process :

6	Inherit DEFINEs from the calling process (NOSUTIL) but ignore DEFINEs stored in ZNSDEF.
22	Inherit DEFINEs from the calling process (NOSUTIL) and include DEFINEs from ZNSDEF.
23	Inherit DEFINEs from the calling process (NOSUTIL), include DEFINEs from ZNSDEF, and run the SCS process at a low PIN.

For more information about create options, see the *Guardian Procedure Calls Reference Manual.*

`EMIT_EVENTS { Y | N }`

specifies whether to use the Event Management Service (EMS) to record the state changes of an object. The object can be a SERVER or SESSION.

Y	Record the state change information.
N	Do not record the state change information.

`MEMORY_CHECK { Y | N }`

specifies whether to have the SCS internally check memory handling:

Y	Check memory handling.
N	Do not check memory handling.

`HOMETERM` *`home-term-name`*

specifies the name of the home terminal for the new process; it is an alphabetic string up to 24 characters in length in the following format:

`[\`*`node`*`.]$`*`ppppp`*`.#PTY`*`aaaa`*

where *ppppp* is a valid process name for TTY devices and *aaaa* is an arbitrary value that identifies a particular terminal. The arbitrary value is unique for each user session.

`IN_FILE` *`input-file-name`*

specifies an input file for the SCS process.

`OUT_FILE` *`output-file-name`*

specifies an output file for the SCS process.

`ERR_FILE` *`error-file-name`*

specifies the location of the error log file for the SCS process; it is an alphabetic string up to 35 characters in length in the following format:

`[\`*`node.`*`]`*`$volume.subvolume.filename`*

`IN_BUFFER_SIZE_B` *`input-buffer-size`*

is an integer in the range 2200 through 32000 that specifies the size of the communication buffer in bytes between NonStop TCP/IP or NonStop IPX/SPX and an SCS process.

`OUT_BUFFER_SIZE_B` *`output-buffer-size`*

is an integer in the range 2200 through 32000 that specifies the size of the communication buffer in bytes between an SCS process and a NOS process.

`DEFAULT_VOLUME [[\`*`node`*`.]$`*`volume`*`.]`*`subvolume`*

specifies the default values for the startup message.

`SCS_DEBUG_OPTIONS` *`scs-debug-options`*

is an INT(2) number that specifies debugging options for the process:

0	Do not use debugging features.
8	Enter DEBUG mode at _MAIN.
9	Enter INSPECT mode at _MAIN and create a SAVEABEND file upon trap.
12	Enter DEBUG mode at _MAIN, ignore the program default for debugging options, and create a SAVEABEND file upon a trap.
13	Enter INSPECT mode at _MAIN, ignore the program default for debugging options, and create a SAVEABEND file upon a trap.

`SCS_RUN_OPTIONS` *`scs-run-options`*

is an alphanumeric string value, internally formatted, up to 240 characters in length that specifies run options for the process.

`NOS_OBJECT` *`nos-object-file-name`*

specifies the valid 34-character NonStop SQL/MP program in the following format:

`[\`*`node.`*`]`*`$volume.subvolume.filename`*

`NOS_LIBRARY_FILE` *`nos-library-file-name`*

is an alphabetic string up to 35 characters in length that specifies the name of a user library for the NOS process. Use this parameter to specify a library other than the default library.

`NOSUTIL_OBJECT` *`nosutil-object-file-name`*

is a 34-character NonStop SQL/MP program.

`NOSUTIL_LIBRARY_FILE` *`nosutil-library-file-name`*

is an alphabetic string up to 35 characters in length that specifies the name of a user library for the NOSUTIL process. Use this parameter to specify a library other than the default library.

`NOSUTIL_PRIORITY` *`nosutil-priority`*

is an INT(2) number between -1 and 199 that specifies the priority of the NOSUTIL process. A value of -1 means to assign the NOSUTIL process the priority of the calling process minus 10 points.

`NOSUTIL_CPU` *`nosutil-cpu`*

is an INT(2) number that identifies the CPU in which to run the NOSUTIL process. The actual range of values depends on the number of CPUs on your node. If you specify -1 for *`nosutil-cpu`*, the PROCESS_CREATE_ procedure automatically assigns a CPU.

`NOSUTIL_CREATE_OPTIONS` *`nosutil-create-options`*

is an INT(2) number that specifies create options for the NOSUTIL process:

6	Inherit DEFINEs from the calling process, but ignore DEFINEs stored in ZNSDEF.
22	Inherit DEFINEs from the calling process, and include at least one DEFINE from ZNSDEF.
23	Inherit DEFINEs from the calling process, include at least one DEFINE from ZNSDEF, and run the NOSUTIL process at a low PIN.

`NOSUTIL_DEBUG_OPTIONS` *`nosutil-debug-options`*

is an INT(2) number that specifies debugging options for the NOSUTIL process:

0	Do not use debugging features.
10	Enter DEBUG mode at _MAIN**.**
11	Enter INSPECT mode at _MAIN and create a SAVEABEND file upon a trap.
13	Enter DEBUG mode at _MAIN, ignore the program default for debugging options, and create a SAVEABEND file upon a trap.
14	Enter INSPECT mode at _MAIN, ignore the program default for debugging options, and create a SAVEABEND file upon a trap.

`NOSUTIL_RUN_OPTIONS` *`nosutil-run-options`*

is an alphanumeric string value, internally formatted, up to 240 characters in length that specifies run options for the NOSUTIL process.

`UPDATE_SYSTEM_CONFIG`

resets the time to update system configuration values to the current time. This option causes running system components to reread their configuration values on the next polling cycle.

Considerations – ADD SCS

The ADD SCS statement generates an error if you are not authorized to modify the system catalog or a warning if *`network-service-name`* is not a registered NonStop ODBC Server network service name.

If you do not specify a value for an attribute, a default value is used from the ZNSCFG table for the attribute, or the value is set to NULL.

Example

```
ADD SCS $scs02 &
   CPU_PRIMARY 3 &
   EXT_SWAPFILE $data2.spare.swap &
   DATAPAGES 64 &
   SCS_CREATE_OPTIONS 22 &
   MEMORY_CHECK y
```

MODIFY SCS

The MODIFY SCS statement modifies one or more of the attribute values or adds a new attribute-value pair in an existing SCS named configuration in the ZNSSCS table.

To execute MODIFY SCS, you must be a privileged user.

```
MODIFY SCS scs-processname
      [ JOB_ID job-id ]
      [ SCS_OBJECT scs-object-file-name ]
      [ SCS_LIBRARY_FILE scs-library-file-name ]
      [ SCS_PRIORITY scs-priority ]
      [ CPU_PRIMARY cpu-primary ]
      [ CPU_BACKUP cpu-backup ]
      [ SWAPVOL swapvol ]
      [ DATAPAGES datapages ]
      [ EXT_SWAPFILE ext-swapvol ]
      [ SCS_CREATE_OPTIONS create-options ]
      [ EMIT_EVENTS { N | Y } ]
      [ MEMORY_CHECK { N | Y } ]
      [ HOMETERM home-term-name ]
      [ IN_FILE input-file-name ]
      [ OUT_FILE output-file-name ]
      [ ERR_FILE error-file-name ]
      [ IN_BUFFER_SIZE_B input-buffer-size ]
      [ OUT_BUFFER_SIZE_B output-buffer-size ]
      [ DEFAULT_VOLUME [[\node.]$volume.]subvolume ]
      [ SCS_DEBUG_OPTIONS scs-debug-options ]
      [ SCS_RUN_OPTIONS scs-run-options ]
      [ NOS_OBJECT nos-object-file-name ]
      [ NOS_LIBRARY_FILE nos-library-file-name ]
      [ NOSUTIL_OBJECT nosutil-object-file-name ]
      [ NOSUTIL_LIBRARY_FILE nosutil-library-file-name ]
      [ NOSUTIL_PRIORITY nosutil-priority ]
      [ NOSUTIL_CPU nosutil-cpu ]
      [ NOSUTIL_DEBUG_OPTIONS nosutil-debug-options ]
      [ NOSUTIL_RUN_OPTIONS "nosutil-run-options" ]
      [ UPDATE_SYSTEM_CONFIG ]
```

scs-processname

specifies the Guardian process name of the SCS named configuration you want to modify.

For a description of the MODIFY SCS parameters, see the ADD SCS statement.

If you specify the SCS_CREATE_OPTIONS attribute for MODIFY SCS, the options apply only to the SCS process.

Considerations – MODIFY SCS

The MODIFY SCS statement generates an error if you are not authorized to modify the system catalog or a warning if `network-service-name` cannot be found in the ZNSNET table.

Example

```
MODIFY SCS $scs02 scs_library_file bugfix
```

REMOVE SCS

The REMOVE SCS statement removes an SCS named configuration from the ZNSSCS table. To execute REMOVE SCS, you must be a privileged user.

```
REMOVE SCS scs-processname [ UPDATE_SYSTEM_CONFIG ]
```

`scs-processname`

specifies the Guardian process name of the SCS named configuration you want to remove from the ZNSSCS table.

Considerations – REMOVE SCS

The REMOVE SCS statement:

- Removes the corresponding entry for the SCS named configuration in ZNSSCS
- Generates an error if you are not a privileged user

Example

```
REMOVE SCS $myscs
```

START SCS

The START SCS statement creates a new SCS process with a specific named configuration from the ZNSSCS table. Configuration values you specify affect an SCS process only when it is started; existing SCS processes are unaffected.

In addition to the named configuration, START SCS searches for and sets any stored DEFINEs to associate with *`scs-processname`*.

```
START SCS scs-processname
```

Considerations – START SCS

The START SCS statement:

- Reads configuration values for the SCS named configuration in ZNSSCS

- Sets any defines stored in ZNSDEF for *`scs-processname`*
- Creates a new SCS process
- Generates an error, if *`scs-processname`* does not exist in ZNSSCS
- Generates an error, if NOSUTIL cannot create the SCS process
- Fails to start NOS processes, if the user is not a registered ODBC user

Example

```
START SCS $myscs
```

STATUS SCS

The STATUS SCS statement returns information about an existing SCS process.

```
STATUS SCS scs-processname
```

Executing STATUS SCS returns the following information:

```
PROCESS file name
PRIORITY
HOMETERM
PROCESS time
CREATOR ID
PROCESS ACCESS ID
JOB ID
PROGRAM file name
SWAP file name
ERROR detail
```

Considerations – STATUS SCS

The STATUS SCS statement:

- Returns status information about the *`scs-processname`* process
- Generates an error if *`scs-processname`* does not exist in ZNSSCS
- Generates an error if NOSUTIL is unable to get information about *`scs-processname`*

Example

```
STATUS SCS $myscs
```

STOP SCS

The STOP SCS statement stops an SCS process. To execute STOP SCS, you must be a privileged user.

```
STOP SCS scs-processname [ NOWAIT ]
```

scs-processname

specifies the process name of the SCS process you want to stop.

The SCS process stops when system configuration values are next updated. The NonStop ODBC Server gracefully shuts down each server process in each server class defined for *scs-processname* (that is, each server process continues running until the respective client session ends).

NOWAIT

causes the SCS process to stop immediately.

Considerations – STOP SCS

The STOP SCS statement generates an error if *scs-processname* does not exist in the ZNSSCS table, if NOSUTIL is unable stop the SCS process, or if you are not a privileged user.

To determine whether an SCS process is stopped, do not rely on the SCS_STOPPED flag (which is displayed by the INFO SCS statement). SCS_STOPPED does not necessarily indicate whether an SCS process has actually stopped. Instead, use the NOSCOM STATUS SCS *scs-processname* statement or the TACL STATUS *scs-processname* command.

SCS_STOPPED is an internal flag used for the graceful shutdown of SCS. If the SCS process is stopped, SCS_STOPPED can have these meanings:

- If SCS_STOPPED is Y, the previous instance of the SCS process was stopped gracefully (unless you stopped the process with a NOSCOM STOP SCS statement followed by a TACL STOP command).
- If SCS_STOPPED is N, the previous instance of the SCS process was not stopped gracefully.

Example

```
STOP SCS $myscs NOWAIT
```

System Configuration Default (SCFG) Statements

Use the following statements to configure and manage system configuration default values:

- ADD SCFG
- MODIFY SCFG
- REMOVE SCFG

ADD SCFG

The ADD SCFG statement adds a new system configuration default value in the system configuration table (ZNSSCFG). To execute ADD SCFG, you must be a privileged user.

```
ADD SCFG scfg-name "scfg-value" [ UPDATE_SYSTEM_CONFIG ]
```

scfg-name

specifies the attribute for which you are assigning a default value; *scfg-name* is an alphanumeric string up to 60 characters in length and must be the name of a column in the ZNSSCFG table. For a list of the ZNSSCFG column names, see Appendix D, Summary of System Installation Defaults,

The value you specify must be alphanumeric; there is no further check for validity.

scfg-value

specifies the default value for *scfg-name*. Enclose *scfg-value* in double quotes ("). There is no check for validity.

UPDATE_SYSTEM_CONFIG

resets the time to update system configuration values to the current time. This option causes running system components to reread their configuration values on the next polling cycle.

The following ADD SCFG statement sets TCP/IP as the default value for the NET_PROTOCOL attribute in the ZNSSCFG table:

```
ADD SCFG NET_PROTOCOL "TCP/IP"
```

MODIFY SCFG

The MODIFY SCFG statement modifies an existing system configuration default value in the ZNSSCFG table. To execute MODIFY SCFG, you must be a privileged user.

```
MODIFY SCFG scfg-name "scfg-value" [ UPDATE_SYSTEM_CONFIG ]
```

scfg-name

specifies the attribute for which you are modifying a default value; *scfg-name* is an alphanumeric string up to 60 characters in length and must be the name of a column in the ZNSSCFG table. For a list of the ZNSSCFG column names, see Appendix D, Summary of System Installation Defaults,

The value you specify must be alphanumeric; there is no further check for validity.

scfg-value

specifies the new default value for *scfg-name*. Enclose the value in quotation marks. There is no check for validity.

△ **Caution.** If you specify an invalid value for *scfg-value,* a subsequent NOSUTIL, NOS, or SCS process that uses that value for configuration might fail to start up.

Example

```
MODIFY SCFG NET_PROTOCOL "NETBIOS"
```

REMOVE SCFG

The REMOVE SCFG SCFG statement removes an existing system configuration attribute-value pair in the ZNSSCFG table. To execute REMOVE SCFG, you must be a privileged user.

```
REMOVE SCFG scfg-name [ UPDATE_SYSTEM_CONFIG ]
```

scfg-name

specifies the attribute (and corresponding value) you are removing from the ZNSSCFG table.

Caution. If you remove a system configuration attribute pair from ZNSSCFG, you must replace it in a subsequent ADD SCFG statement. If an attribute-value pair is missing from the ZNSSCFG table or the value you specified is not valid, a subsequent NOSUTIL, NOS, or SCS process that uses that value for configuration might fail to start up.

The following REMOVE SCFG statement removes the NET_PROTOCOL attribute and its associated value from the ZNSSCFG table:

```
REMOVE SCFG NET_PROTOCOL UPDATE_SYSTEM_CONFIG
```

TABLE Statements

The following TABLE statements affect whether a NOSCOM user can access a NonStop SQL/MP base table:

- ADD TABLE
- REMOVE TABLE

ADD TABLE

The ADD TABLE statement makes a NonStop SQL/MP base table visible to a NOSCOM user. No REFRESH of the entire database is necessary.

To execute ADD TABLE, you must be a privileged user. However, the NonStop ODBC Server does not verify privileges on the underlying NonStop SQL/MP table.

```
ADD TABLE nssql-filename [ AS logical-objectname ]
```

nssql-filename

is a Guardian file name that represents the physical location of the NonStop SQL/MP table. It can be in any of the following formats:

```
filename
subvolume.filename
$volume.subvolume.filename
\node.filename
\node.subvolume.filename
\node.$volume.subvolume.filename
```

The NonStop ODBC Server fully qualifies *nssql-filename* before performing the statement. Any optional values you omit are determined by the user profile of the current process.

AS *logical-objectname*

is the logical name you specify for the table. It can be in any of the following formats:

```
objectname
logical-user-name.objectname
database-name.logical-user-name.objectname
database-name..objectname
```

The value you specify for *database-name* must be a customized NonStop ODBC Server database.

The NonStop ODBC Server fully qualifies *logical-objectname* before performing the statement. Any optional values you omit are determined by the user profile of the current process.

Considerations – ADD TABLE

The ADD TABLE statement:

- Creates a new entry in ZNUOBJ for the specified NonStop SQL/MP base table
- Generates an error if *file-name* is already mapped in the appropriate object file
- Generates an error if *nssql-filename* is not a valid NonStop SQL/MP table

- Generates an error if the table you specify is not registered in the NonStop SQL/MP database
- Generates an error if you are not authorized to modify the system catalog
- Generates an error if *`database-name`* is not the name of a customized NonStop ODBC Server database
- Generates an error if *`logical-object-name`* is already mapped to a different Guardian file name

Example

```
ADD TABLE \persnl.$db1.region1.tblA
       AS persnl_db1_region1.dbo.summary_table_A
```

REMOVE TABLE

The REMOVE TABLE statement removes a mapping entry for the specified NonStop SQL/MP table. The underlying table is unaffected. To execute REMOVE TABLE, you must be a privileged user.

```
REMOVE TABLE nssql-filename [ FROM database-name ]
```

`nssql-filename`

is a Guardian filename that represents the physical location of the NonStop SQL/MP table. It can be in any of the following formats:

```
filename
subvolume.filename
$volume.subvolume.filename
\node.filename
\node.subvolume.filename
\node.$volume.subvolume.filename
```

The NonStop ODBC Server fully qualifies *`nssql-filename`* before performing the statement. Any optional values you omit are determined by the user profile of the current process.

`FROM` *`database-name`*

is an alphanumeric string up to 60 characters in length that specifies the NonStop ODBC Server customized database from which to remove the NonStop SQL/MP table.

The default is the current default database for the current user profile.

Considerations – REMOVE TABLE

The REMOVE TABLE statement:

- Deletes a mapping entry in ZNUOBJ for the NonStop SQL/MP table indicated by *`nssql-filename`*
- Deletes primary key and index entries for the table in ZNUIX (if they exist).
- Generates an error if you are not a privileged user

Example

```
REMOVE TABLE \persnl.$db1.region1.tblA
```

Trace Log (TRA_LOG) Statements

The following TRA_LOG statements affect NonStop ODBC Server trace log tables:

- CREATE TRA_LOG
- DROP TRA_LOG

CREATE TRA_LOG

The CREATE TRA_LOG statement creates a new trace log table. The table is registered as a NonStop ODBC Server object in the database. To execute CREATE TRA_LOG, you must be a privileged user.

```
CREATE TRA_LOG table-name [ SECURE "security-string" ]
```

`table-name`

specifies the qualified name of the trace log table to create.

`SECURE` *`security-string`*

is a four-character alphabetic string that specifies the Guardian read, write, execute, and purge (RWEP) access for the trace log table. For more information about security for NonStop SQL/MP objects, see the “Security” entry in the *NonStop SQL/MP Reference Manual*.

Considerations – CREATE TRA_LOG

The CREATE TRA_LOG statement:

- Generates an error if the database is not a customized NonStop ODBC Server database
- Generates an error if the associated SQL CREATE TABLE statement fails for either resource or privilege reasons

Example

```
CREATE TRA_LOG sf_disk01_tralog01 SECURE "COOO"
```

DROP TRA_LOG

The DROP TRA_LOG statement drops an existing trace log table. To execute DROP TRA_LOG, you must be a privileged user.

```
DROP TRA_LOG table-name
```

table-name

specifies the qualified name of the trace log table to drop.

Considerations – DROP TRA_LOG

The DROP TRA_LOG statement:

- Drops the specified trace log table
- Removes the mapping entry for the log table in the specified database
- Preserves the entry in ZNSPROF for the log table
- Generates an error if *database-name* is not a customized NonStop ODBC Server database
- Generates an error if you do not have sufficient privileges to execute DROP TRA_LOG
- Generates an error if the associated SQL DROP TABLE statement fails for either resource or privilege reasons

Example

```
DROP TRA_LOG sf_disk01_tralog01
```

TRACE Statements

Use the following statements to configure and manage named trace entities:

- ADD TRACE
- MODIFY TRACE
- REMOVE TRACE

ADD TRACE

The ADD TRACE statement adds and configures a new trace entity. To execute ADD TRACE, you must be a privileged user.

```
ADD TRACE trace-name
   [ TRA_LOGTABLE_NAME log-tablename ]
   [ LOG_TO_HOMETERM { Y | N } ]
   [ INPUT_STREAM { Y | N } ]
   [ OUTPUT_STREAM { Y | N } ]
   [ NSSQL { Y | N } ]
   [ TRA_ERROR { Y | N } ]
   [ CACHE_STATISTICS { Y | N } ]
   [ SP_WRITE { Y | N } ]
   [ SP_READ { Y | N } ]
   [ UPDATE_SYSTEM_CONFIG ]
```

trace-name

specifies the trace to create; *trace-name* is an alphanumeric string up to 60 characters in length and must be unique.

TRA_LOGTABLE_NAME *log-tablename*

specifies the name of the trace log table, stored as a fully qualified three-part logical name in the format *database.owner.name*. The default is NULL.

LOG_TO_HOMETERM { Y | N }

specifies whether trace records should be written to the terminal used to start the NonStop ODBC Server. Trace records can be written to the current server process home terminal only when the server is manually started. The setting has no effect on logging to a designated trace log.

INPUT_STREAM { Y | N }

specifies whether input records should be recorded as they are read from the SQL Communication Subsystem before decoding (values are written in hexadecimal).

OUTPUT_STREAM { Y | N }

specifies whether output records should be recorded as they are written to the SQL Communication Subsystem after decoding (values are written in hexadecimal).

NSSQL { Y | N }

specifies whether all SQL should be recorded as it is generated by the server and sent to NonStop SQL/MP (both client SQL and server catalog management SQL are logged).

TRA_ERROR { Y | N }

specifies whether all errors generated or detected by the server should be logged.

`CACHE_STATISTICS { Y | N }`

specifies whether cache statistics should be logged, with a log record being written out at the end of the user session.

`SP_WRITE { Y | N }`

specifies whether all PATHSEND write messages generated by stored procedures should be logged.

`SP_READ { Y | N }`

specifies whether all NonStop SQL/MP results returned from the invocation of stored procedures should be logged.

`UPDATE_SYSTEM_CONFIG`

resets the time to update system configuration values to the current time. This option causes running system components to reread their configuration values on the next polling cycle.

Considerations – ADD TRACE

The ADD TRACE statement:

- Creates a new entry in the ZNSTRA table that defines the trace options
- Generates an error if *`trace-name`* already exists in ZNSTRA
- Generates an error if the system catalog is not customized for the NonStop ODBC Server
- Generates an error if you are not a privileged user

See the *NonStop ODBC Server Installation and Management Manual* for information about using the trace feature.

Example

```
ADD TRACE trace4 &
    TRA_LOGTABLE_NAME newyork_disk2_sqldb.sql_dave.trace4 &
    LOG_TO_HOMETERM y TRA_ERROR y UPDATE_SYSTEM_CONFIG
```

MODIFY TRACE

The MODIFY TRACE statement changes the configuration of an existing trace entity. To execute MODIFY TRACE, you must be a privileged user.

```
MODIFY TRACE trace-name
     [ TRA_LOGTABLE_NAME log-tablename ]
     [ LOG_TO_HOMETERM { Y | N } ]
     [ INPUT_STREAM { Y | N } ]
     [ OUTPUT_STREAM { Y | N } ]
     [ NSSQL { Y | N } ]
     [ TRA_ERROR { Y | N } ]
     [ CACHE_STATISTICS { Y | N } ]
     [ SP_WRITE { Y | N } ]
     [ SP_READ { Y | N } ]
     [ UPDATE_SYSTEM_CONFIG ]
```

trace-name

specifies the trace to modify. For a description of the other MODIFY TRACE parameters, see the ADD TRACE statement.

Considerations – MODIFY TRACE

The MODIFY TRACE statement:

- Modifies the configuration options for the specified trace in ZNSTRA
- Generates an error if the system catalog is not customized for the NonStop ODBC Server
- Generates an error if you are not authorized to modify the system catalog

Example

```
MODIFY TRACE trace4 CACHE_STATISTICS Y UPDATE_SYSTEM_CONFIG
```

REMOVE TRACE

The REMOVE TRACE statement removes a configuration entry from ZNSTRA. To execute REMOVE TRACE, you must be a privileged user.

```
REMOVE TRACE trace-name [ UPDATE_SYSTEM_CONFIG ]
```

trace-name

specifies the trace to remove.

Considerations – REMOVE TRACE

The REMOVE TRACE statement:

- Removes the entry for the specified trace from ZNSTRA
- Generates an error if the system catalog is not customized for the NonStop ODBC Server
- Generates an error if you are not a privileged user

Example

```
REMOVE TRACE trace4
```

User Mapping (UMAP) Statements

Use the following statements to map a user and SCS process to a server class:

- ADD UMAP
- MODIFY UMAP
- REMOVE UMAP

ADD UMAP

The ADD UMAP statement associates an existing alias username (Guardian username and optionally, a profile) with the specified server class. To execute ADD UMAP, you must be a privileged user.

```
ADD UMAP scs-processname alias-username
         SER_NAME server-name
         [ UPDATE_SYSTEM_CONFIG ]
```

scs-processname

is an alphabetic string up to 15 characters in length that specifies the name of the SCS process in the following format:

`\`*`node`*`.$`*`process-name`*

alias-username

is an alphanumeric string up to 60 characters in length that specifies an alias username to associate with *logical-user-name*; *alias-username* must begin with an alphabetic character, followed by alphabetic, numeric, and underscore characters. This name must be unique and must already exist in ZNSALT.

`SER_NAME` *server-name*

is an alphanumeric string up to 32 characters in length that specifies the name of the server class configuration entity to associate with a user and profile; *server-name* must already be mapped to *scs-processname* in ZNSMAP.

`UPDATE_SYSTEM_CONFIG`

resets the time to update system configuration values to the current time. This option causes running system components to reread their configuration values on the next polling cycle.

Considerations – ADD UMAP

The ADD UMAP statement:

- Creates a new entry in ZNSUMAP that associates the user and SCS process to the specified server class
- Generates a warning if *`alias-username`* is not a valid NonStop ODBC Server alias
- Generates a warning if *`server-name`* is not a valid NonStop ODBC server class
- Generates an error if you are not authorized to modify the system catalog

Example

```
ADD UMAP $myscs sql_dave ser?? alpha
```

MODIFY UMAP

The MODIFY UMAP changes the mapping between an SCS process and alias username with a specified server class. To execute MODIFY UMAP, you must be a privileged user.

```
MODIFY UMAP scs-processname alias-username
            SER_NAME server-name
            [ UPDATE_SYSTEM_CONFIG ]
```

`scs-processname`

is an alphabetic string up to 15 characters in length that specifies the name of the SCS process in the following format:

`\`*`node`*`.$`*`process-name`*

`alias-username`

is an alphanumeric string up to 60 characters in length that specifies an alias username to associate with *`logical-user-name`*; *`alias-username`* must begin with an alphabetic character, followed by alphabetic, numeric, and underscore characters. This name must be unique and must already exist in ZNSALT.

`SER_NAME` *`server-name`*

is an alphanumeric string up to 32 characters in length that specifies the name of the server class configuration entity to associate with a user and profile; *`server-name`* must already be mapped to *`scs-processname`* in ZNSMAP.

`UPDATE_SYSTEM_CONFIG`

resets the time to update system configuration values to the current time. This option causes running system components to reread their configuration values on the next polling cycle.

Considerations – MODIFY UMAP

The MODIFY UMAP statement:

- Modifies an existing entry in ZNSUMAP that maps an SCS process and user to the specified server class
- Generates a warning if *`alias-username`* is not a valid NonStop ODBC Server alias
- Generates a warning if *`scs-processname`* and *`alias-username`* are not already mapped to each other and represented by an entry in ZNSUMAP
- Generates a warning if *`server-name`* is not a valid NonStop ODBC server class configuration entity represented by an entry in ZNSSER
- Generates an error if you are not authorized to modify the system catalog

Example

```
MODIFY UMAP $myscs sql_dave SER_NAME beta UPDATE_SYSTEM_CONFIG
```

REMOVE UMAP

The REMOVE UMAP statement remove an existing association between an SCS process and user association and a server class. To execute REMOVE UMAP, you must be a privileged user.

```
REMOVE UMAP scs-processname alias-username
            [ UPDATE_SYSTEM_CONFIG ]
```

Considerations – REMOVE UMAP

The REMOVE UMAP statement:

- Modifies an existing entry in ZNSUMAP that maps an SCS process and user to a server class
- Generates an error if you are not a privileged user

- Generates an error if *`scs-processname`* and *`alias-username`* are not mapped to each other and represented by an entry in ZNSUMAP

Example

```
REMOVE UMAP $myscs sql_dave
```

USER Statements

Use the following statements to map logical usernames to Guardian usernames:

- ADD USER
- MODIFY USER
- REMOVE USER

ADD USER

The ADD USER statement associates a logical username with a Guardian username and, optionally, with a profile. To execute ADD USER, you must be either the user identified by *`guardian-username`* or be a privileged user.

```
ADD USER logical-username guardian-username
         [ CHANGE_PASSWORD_OPTION { 0 | 1 | 2 } ]
         [ PROFILE profilename ]
         [ UPDATE_SYSTEM_CONFIG ]
```

`logical-username`

is an alphanumeric string up to 60 characters, beginning with a letter of the alphabet and optionally containing hyphens (-), that specifies a logical username to associate with *`guardian-username`*. The default for is NULL.

`guardian-username`

is an alphabetic string up to 17 characters in length that specifies the Guardian username to associate with *`logical-username`* in the format:

`group.user`

An error occurs if *`guardian-username`* already exists in ZNSUS. The default is the default for the current process.

`CHANGE_PASSWORD_OPTION { 0 | 1 | 2 }`

specifies the level of notification sent to users for Safeguard passwords that are expired or are about to expire:

0	No notification is sent. A user can log in only if the password is correct and has not expired, including the grace period. This is the default.
1	Notification is sent if the password is still in the grace period. A user can then change the password and log in.
2	Notification is sent if the password is about to expire or if it has expired but is still in the grace period. If the password has not yet expired, the user can change the password or cancel the dialog box. If the password has already expired, the user must change the password immediately.

`PROFILE` *`profilename`*

is an alphanumeric string up to 60 characters, beginning with a letter of the alphabet and optionally containing hyphens (-), that identifies the profile record in ZNSPROF to use when a user logs into the server with *`logical-username`*. An error occurs if *`profilename`* already exists in ZNSPROF.

If you omit the `PROFILE` parameter, the profile name DEFAULT is used.

`UPDATE_SYSTEM_CONFIG`

resets the time to update system configuration values to the current time. This option causes running system components to reread their configuration values on the next polling cycle.

Considerations – ADD USER

The ADD USER statement:

- Creates an entry in the ZNSUS table that associates *`logical-username`* with *`guardian-username`*
- Creates an entry in ZNSPROF for the default profile for this user
- Generates an error if *`logical-username`* or *`guardian-username`* already exists in ZNSUS
- Generates an error if you are not the user identified by *`guardian-username`* or not a privileged user
- Generates an error if the system catalog is not customized for the NonStop ODBC Server
- If an ADD USER operation fails for any reason, the mapping tables are unchanged.

Example

In the following example, ADD USER defines a user whose logical username is SQL_Jones and whose Guardian username is SQL.Jones. Jones uses the profile record PERSNL.

```
ADD USER SQL_Jones SQL.Jones PROFILE persnl
```

MODIFY USER

The MODIFY USER statement changes the logical username associated with a particular Guardian username.

Note. To change the profile name associated with a logical username, use MODIFY ALIAS.

To execute MODIFY USER, you must be a privileged user.

```
MODIFY USER logical-username guardian-username
            [ UPDATE_SYSTEM_CONFIG ]
```

logical-username

is an alphanumeric string up to 60 characters, beginning with a letter of the alphabet and optionally containing hyphens (-), that specifies a logical username to associate with *guardian-username*. The default for is NULL.

guardian-username

is an alphabetic string up to 17 characters in length that specifies the Guardian username to associate with *logical-username* in the format:

group.user

An error occurs if *guardian-username* already exists in ZNSUS. The default is the default for the current process.

Considerations – MODIFY USER

The MODIFY USER statement:

- Changes the *logical guardian-username* attributes in ZNSUS
- If you change *guardian-username*, updates the ownership of NonStop ODBC Server mapping table objects when the system catalog is next refreshed
- Generates an error if *logical-username* cannot be found
- Generates an error if the system catalog is not customized for the NonStop ODBC Server
- Generates an error if you are not a privileged user

Example

```
MODIFY USER SQL_Jones SQL.ODBC
```

REMOVE USER

The REMOVE USER statement removes the mapping between a logical username and a Guardian username.

Note. REMOVE USER does not remove associated alias usernames. Deleting the logical username has no effect on object attributes.

To execute REMOVE USER, you must be a privileged user.

```
REMOVE USER logical-username [ CASCADE ]
                             [ UPDATE_SYSTEM_CONFIG ]
```

`logical-username`

specifies the logical username you want to remove. The default is NULL.

`CASCADE`

removes all alias mappings in the ZNSALT table that refer to *`logical-username`*.

Considerations – REMOVE USER

The REMOVE USER statement:

- Removes the entry for *`logical-username`* from ZNSUS
- Deletes from ZNSALT the self-referencing row that corresponds to *`logical-username`*
- Generates an error if *`logical-username`* cannot be found in ZNSUS
- Generates an error if the system catalog is not customized for the NonStop ODBC Server
- Generates an error if you are not a privileged user

Example

```
REMOVE USER SQL_Smith
```

VIEW Statements

Use the following statements to control whether a NOSCOM user can access a specified view of a NonStop SQL/MP base table:

- ADD VIEW
- REMOVE VIEW

ADD VIEW

The ADD VIEW statement makes the specified view of a NonStop SQL/MP base table visible to a NOSCOM user. No REFRESH of the entire database is necessary.

To execute ADD VIEW, you must be a privileged user. However, the NonStop ODBC Server does not verify privileges on the underlying NonStop SQL/MP view.

```
ADD VIEW nssql-filename [ AS logical-objectname ]
```

nssql-filename

is an alphabetic string up to 34 characters in length that identifies a Guardian file that represents the physical location of the view; *nssql-filename* can be in any of the following formats:

```
filename
subvolume.filename
$volume.subvolume.filename
\node.filename
\node.subvolume.filename
\node.$volume.subvolume.filename
```

The NonStop ODBC Server fully qualifies *nssql-filename* before performing the VIEW operation. Any optional values you omit are determined by the user profile of the current process.

AS *logical-objectname*

is an alphanumeric string up to 182 characters in length that specifies the logical name of the view; *logical-objectname* can be in any of the following formats:

```
object-name
logical-user-name.object-name
database-name.logical-user-name.object-name
database-name..object-name
```

Any value you specify for *database-name* must be a customized NonStop ODBC Server database.

The NonStop ODBC Server fully qualifies *logical-objectname* before performing the VIEW operation. Any optional values you omit are determined by the user profile of the current process.

Considerations – ADD VIEW

The ADD VIEW statement:

- Creates a new mapping entry in ZNUOBJ for the specified view
- Adds a mapping entry to ZNSUS if *`logical-username`* is not currently mapped to the owner of *`nssql-filename`*
- Generates an error if you do not have read access to *`nssql-filename`*, or if you do not have write access to the mapping tables associated with *`logical-objectname`*
- Generates an error if *`nssql-filename`* is already mapped in the appropriate object file.
- Generates an error if *`nssql-filename`* is not a valid NonStop SQL/MP view
- Generates an error if the view you specify is not registered in the NonStop SQL/MP database
- Generates an error if you are not a privileged user
- Generates an error if *`database-name`* is not the name of a customized NonStop ODBC Server database
- Generates an error if *`logical-objectname`* is already mapped to a different Guardian file name

If an ADD VIEW operation fails for any reason, the mapping tables are unchanged

Example

```
ADD VIEW \sales.$db1.region2.vw AS vw2
```

REMOVE VIEW

The REMOVE VIEW statement deletes an existing mapping entry for the specified NonStop SQL/MP view. The underlying view is unaffected. To execute REMOVE VIEW, you must be a privileged user.

```
REMOVE VIEW nssql-filename
```

`nssql-filename`

is an alphabetic string up to 34 characters in length that identifies a Guardian file that represents the physical location of the view; *`nssql-filename`* can be in any of the following formats:

```
filename
subvolume.filename
$volume.subvolume.filename
\node.filename
```

\node.subvolume.filename
\node.$volume.subvolume.filename

The NonStop ODBC Server fully qualifies `nssql-filename` before performing the REMOVE VIEW statement. Any optional values you omit are determined by the user profile of the current process.

Considerations – REMOVE VIEW

The REMOVE VIEW statement:

- Removes a mapping entry in ZNUOBJ for the specified view
- Preserves the underlying NonStop SQL/MP object
- Generates a warning if *`nssql-filename`* is referenced by a profile entry in ZNSPROF

Example

```
REMOVE VIEW \sales.$db1.region2.vw
```

Catalog Support Procedures

ODBC provides several procedures for ODBC catalog support. These procedures are functions the ODBC driver supports, and are useful to users who build programs using the ODBC driver.

You invoke each procedure with a Microsoft Windows call. The driver issues the queries, which return information from the ODBC catalog by populating a list of columns. Some columns are in result sets, and others are in named program-generated views. The output is dynamic and is composed of elements of NonStop SQL for which there is no equivalent API functionality.

These are the procedures:

- SQLColumns
- SQLPrimaryKeys
- SQLProcedureColumns
- SQLProcedures
- SQLSpecialColumns
- SQLStatistics
- SQLTables

The following subsections provide information about these procedures. The view names are included for those procedures that generate views.

For more information about these procedures, see the *Microsoft ODBC 2.0 Programmer’s Reference and SDK Guide*.

SQLColumns

SQLColumns is a procedure that returns a list of user-requested column names and column information from user-requested tables. The view ZVUOCOL provides support for SQLColumns. ZVUOCOL is a view over COLUMNS, ZNUOBJ, and ZNUDT.

Input parameters for SQLColumns are:

- TABLE_QUALIFIER
- TABLE_OWNER
- TABLE_NAME
- COLUMN_NAME

Table 7-5. Result Set for the SQLColumns Procedure

Result Column	Data Type	Description
TABLE_QUALIFIER	VARCHAR (60)	Assigned null.
TABLE_OWNER	VARCHAR (60)	The owner of the table.
TABLE_NAME	VARCHAR (60)	The name of the table.
COLUMN_NAME	VARCHAR (30)	The name of the column, converted from CHAR (30).
DATA_TYPE	INT (2)	The type of data (described in TYPE_NAME).
TYPE_NAME	VARCHAR (60)	The corresponding name for the data type value in DATA_TYPE.
PRECISION	INTEGER	The precision of the requested column.
LENGTH	INTEGER	The length of the requested column.
SCALE	INT (2)	The scale of the requested column.
RADIX	INT (2)	The radix of the column data: 2 10
NULLABLE	INT (2)	Indicates whether the requested column allows null values: 0 SQL_NO__NULLS 1 SQL_NULLABLE
REMARKS	VARCHAR (254)	Assigned "" (blank).

SQLPrimaryKeys

SQLPrimaryKeys is a procedure that lists keys for base tables. It does not report keys for views. ODBC does not require views to have independent indexes.

SQLPrimaryKeys is a view on ZNOBJ, KEYS, COLUMNS, and INDEXES. A view specific to the NonStop ODBC Server is not provided for this result set.

Input parameters for SQLPrimaryKeys are:

- TABLE_QUALIFIER
- TABLE_OWNER
- TABLE_NAME

Table 7-6. Result Set for the SQLPrimaryKeys Procedure

Result Column	Data Type	Description
TABLE_QUALIFIER	VARCHAR (60)	Assigned null.
TABLE_OWNER	VARCHAR (60)	The table owner.
TABLE_NAME	VARCHAR (60)	The name of the table.
COLUMN_NAME	CHAR (30)	The name of the column.
KEY_SEQ	INT (2)	The sequence number of the column in column (key sequence number + 1).

SQLProcedureColumns

SQLProcedureColumns is a procedure that lists columns defined for a stored procedure. The columns can be input or output types. The values are determined at the time the procedure is added through NOSUTIL.

SQLProcedureColumns is a join over the mapping tables, ZNUPCOL and ZNUDT. The values for the result set are determined at the time NOSUTIL adds the procedure to the system.

Input parameters for SQLProcedureColumns are:

- PROCEDURE_QUALIFIER
- PROCEDURE_OWNER
- PROCEDURE_NAME
- COLUMN_NAME

If you omit COLUMN_NAME, SQLProcedureColumns lists all columns for the specified procedure.

Description

Table 7-7. Result Set for the SQLProcedureColumns Procedure

Result Column	Data Type	Description
PROCEDURE_QUALIFIER	VARCHAR (60)	Assigned null.
PROCEDURE_OWNER	VARCHAR (60)	The owner of the procedure.
PROCEDURE_NAME	VARCHAR (60)	The name of the procedure.
COLUMN_NAME	VARCHAR (60)	The name of the column or parameter.
COLUMN_TYPE	INT (2)	A column type indicator: 1 SQL_PARAM_INPUT 2 SQL_PARAM_INPUT_OUTPUT
DATA_TYPE	INT (2)	The encoded value indicating the data type of the parameter.
TYPE_NAME	VARCHAR (60)	The string identifier value indicating the data type of the parameter.
PRECISION	INT (2)	The logical precision of the requested column. The default is 1.
LENGTH	INT (2)	The logical length of the requested column. The default is 1.
SCALE	INT (2)	The logical scale of the requested column. The default is 10.
RADIX	INT (2)	The logical radix of the column data. The default is 10.
NULLABLE	INT (2)	Indicates whether the procedure column allows null values: 0 SQL_NO__NULLS 1 SQL_NULLABLE The default is 1.
REMARKS	VARCHAR (254)	Assigned null.

SQLProcedures

SQLProcedures is a procedure that lists the procedures that are defined for a user-requested database. The values for the result set are determined at the time the procedure is added to the catalog for execution access through NOSUTIL.

Input parameters for SQLProcedures are:

- PROCEDURE_QUALIFIER
- PROCEDURE_OWNER
- PROCEDURE_NAME

Table 7-8. Result Set for the SQLProcedures Procedure

Result Column	Data Type	Description
PROCEDURE_QUALIFIER	VARCHAR (60)	Assigned null.
PROCEDURE_OWNER	VARCHAR (60)	The owner of the procedure.
PROCEDURE_NAME	VARCHAR (60)	The name of the procedure.
NUM_INPUT_PARAMS	INT (2)	The number of input parameters defined for the procedure.
NUM_OUTPUT_PARAMS	INT (2)	The number of output parameters defined for the procedure.
NUM_RESULT_SETS	INT (2)	The number of result sets defined for the procedure.
REMARKS	VARCHAR (254)	Assigned "" (blank).

SQLSpecialColumns

SQLSpecialColumns is a procedure that lists the columns of a table that uniquely identify a row in a table.

Input parameters for SQLSpecialColumns are:

- COLTYPE, which must be:
  - SQL_BEST_ROWID, a define value set to 1
- TABLE_QUALIFIER
- TABLE_OWNER
- TABLE_NAME
- SCOPE, which must be:
  - SQL_SCOPE_TRANSACTION, a define value set to 1
- NULLABLE, which must be:
  - SQL_NO_NULLS, a define value set to 0

Table 7-9. Result Set for the SQLSpecialColumns Procedure (page 1 of 2)

Result Column	Data Type	Description
SCOPE	INT (2)	The duration of the result set. Assigned 2, indicating the result set is good for the duration of the session.
COLNAME	CHAR (30)	The name of the column.
DATATYPE	INT (2)	The mode-dependent code for data type.
TYPENAME	CHAR (30)	A character-string value associated with the value for DATATYPE.

Table 7-9. Result Set for the SQLSpecialColumns Procedure (page 2 of 2)

Result Column	Data Type	Description
PRECISION	INT (4)	The number of digits or bits of precision for numeric data types.
LENGTH	INT (4)	The number of bytes for nonnumeric data types.
SCALE	INT (2)	The scale of numeric data types.

SQLStatistics

SQLStatistics is a procedure that lists index information for base tables.

Input parameters for SQLStatistics are:

- TABLE_QUALIFIER
- TABLE_OWNER
- TABLE_NAME
- UNIQUE, which is one of the following:
  - SQL_INDEX_UNIQUE, a define value set to 0
  - SQL_INDEX_ALL, a define value set to 1

Table 7-10. Result Set for SQLProcedureColumns Procedure (page 1 of 2)

Result Column	Data Type	Description
TABLE_QUALIFIER	VARCHAR (60)	Assigned null.
TABLE_OWNER	VARCHAR (60)	The owner of the table.
TABLE_NAME	VARCHAR (60)	The name of the table.
NON_UNIQUE	INT (2)	Indicates whether the index allows duplicate values: 1 INDEXES.UNIQUEVALUE = “N” 0 INDEXES.UNIQUEVALUE = “Y”
INDEX_QUALIFIER	VARCHAR (60)	Assigned null.
INDEX_NAME	VARCHAR (60)	The name of the index.
TYPE	INT (2)	The subject of the statistics. If the value for INDEX_NAME is NULL, statistics are about the primary key on the indicated table. If it is not NULL, statistics are about other indexes.
SEQ_IN_INDEX	INT (2)	Index column sequence number.
COLUMN_NAME	VARCHAR (60)	The COLUMN IDENTIFIER.

Table 7-10. Result Set for SQLProcedureColumns Procedure (page 2 of 2)

Result Column	Data Type	Description
COLLATION	CHAR (1)	The collating sequence of the keys: A Ascending D Descending
CARDINALITY	INT (2)	Assigned null.
PAGES	INT (2)	Assigned null.

SQLTables

SQLTables lists database, schema name, and table information.

Input parameters for SQLTables are:

- TABLE_QUALIFIER
- TABLE_OWNER
- TABLE_NAME
- TABLE_TYPE

Table 7-11. Result Set for SQLTables Procedure

Result Column	Data Type	Description
TABLE_QUALIFIER	VARCHAR (60)	Assigned null.
TABLE_OWNER	VARCHAR (60)	The owner of the table.
TABLE_NAME	VARCHAR (60)	The name of the table.
TABLE_TYPE	VARCHAR (60)	The type of table: N or S System view T Base table V View
REMARKS	VARCHAR (254)	Comments.

8 HP NonStop ODBC Server Mapping Tables

The HP NonStop ODBC Server acts as a gateway to HP NonStop SQL/MP from external entities. As such, it must provide translation from external concepts and protocols to NonStop SQL/MP concepts and protocols. The NonStop ODBC Server creates and maintains a set of tables, views, and indexes called the NonStop ODBC Server mapping tables. Although the mapping tables are composed of views and indexes as well as tables, and although some are involved in mapping and provide support, all are collectively called mapping tables. The tables that do the actual mapping map ODBC or SQL Server names and data types to NonStop SQL/MP names and data types.

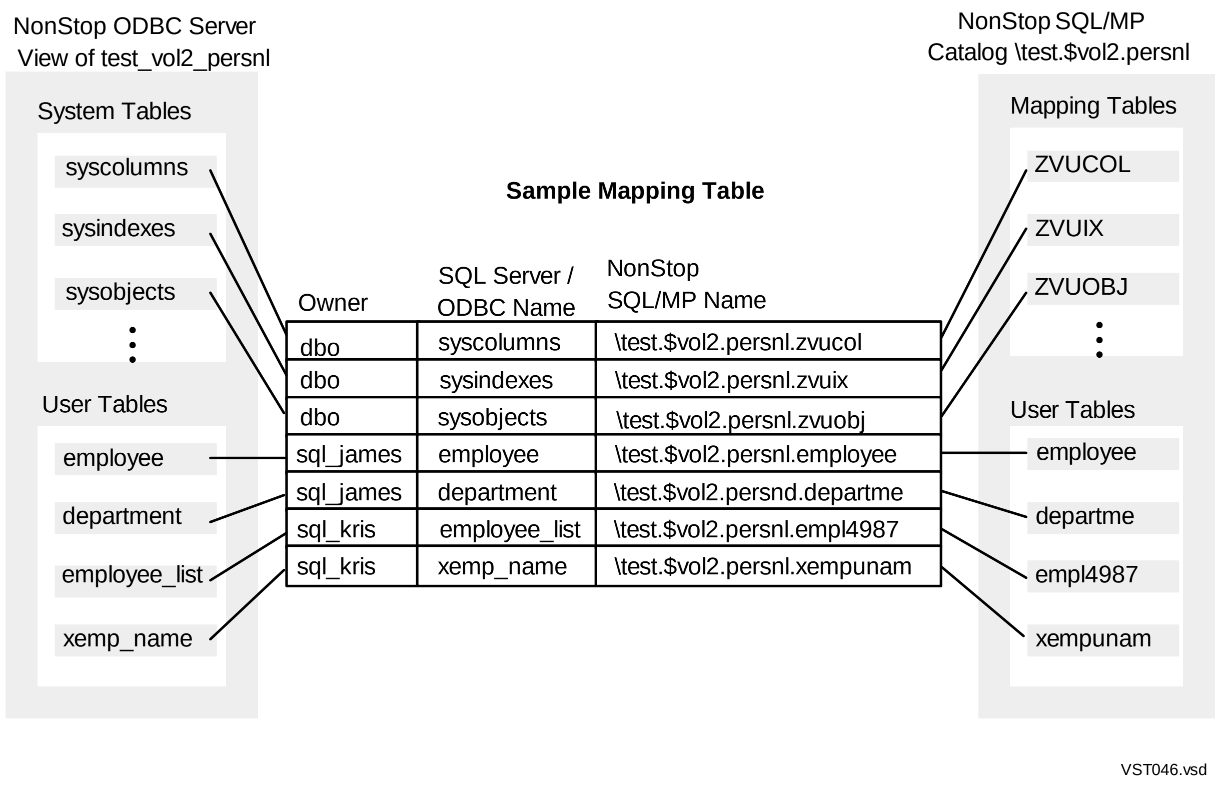
This section describes all of the tables, views, and indexes and the actions that can affect them. It includes the following information:

- Types of mapping
- Summary of the mapping tables
- Database and object mapping
- System table mapping
- Actions that affect mapping tables

Customized catalogs and the catalog utilities are described in Section 7, Managing Customized Catalogs.

Figure 8-1 shows a customized NonStop SQL/MP catalog (\TEST.$VOL2.PERSNL) and a portion of the table that maps the NonStop ODBC Server or SQL Server object names to NonStop SQL/MP object names.

Figure 8-1. A NonStop ODBC Server Mapping Table



Types of Mapping

The NonStop ODBC Server maps database names, object names, and system table names. The mapping is invisible to the user: users specify ODBC or SQL Server names, and the NonStop ODBC Server locates the corresponding NonStop SQL/MP name by using the mapping tables.

Database and Object Mapping

The NonStop ODBC Server maintains three mapping tables listing all databases, indexes, tables, and views for a customized catalog:

Mapping Table	Objects Mapped
ZNSDB	Databases
ZNUIX	Indexes
ZNUOBJ	Tables and views

When you specify an ODBC or SQL Server object name, the NonStop ODBC Server searches the corresponding mapping table and locates the NonStop SQL/MP name for that object.

For more information, see Database and Object Mapping on page 8-8.

System Table Mapping

The NonStop ODBC Server maintains tables and views that correspond to the SQL Server system tables. When you specify a system table name, the NonStop ODBC Server searches the object mapping table, ZNUOBJ, and locates the view name that corresponds to the system table.

For more information, see System Table Mapping on page 8-60.

Summary of the Mapping Tables

The mapping tables reside in the following subvolumes:

- The subvolume of the system catalog. These mapping tables are created when you install the NonStop ODBC Server.
- The subvolume of a customized user catalog. The NonStop ODBC Server creates these mapping tables when an existing catalog is customized or when a database is created using the NonStop ODBC Server.

Naming Conventions

Object name prefixes provide information about the object. “ZN” indicates a base table or index used by the NonStop ODBC Server to simulate TSQL and ODBC catalog structures on a NonStop SQL catalog. “ZV” indicates a view provided by the NonStop ODBC Server to simulate TSQL and ODBC catalog structures on a NonStop SQL catalog.

The prefixes in Table 8-1 indicate the object type and the location of the object.

Table 8-1. Naming Conventions for the Mapping Tables

		Location	
Prefix	**Object Type**	**Subvolume of the System Catalog**	**Subvolume of the User Catalog**
ZNS	Table or index	x	–
ZNU	Table or index	x	x
ZVS	View	x	–
ZVU	View	x	x

x Indicates the object exists in this catalog
– Indicates the object does not exist in this catalog

For example, ZVSDB is a view in the subvolume of the system catalog, and ZNUOBJ is a table in both the subvolume of the system catalog and the customized user catalogs.

Prefixes for column names contain information about column values, as described in Table 8-2.

Table 8-2. Naming Conventions for the Mapping Table Columns

Prefix	Column Value
ACC_	Relates to the resource accounting description.
G_	Relates to the HP NonStop Kernel operating system.
GOV_	Relates to the resource governing description.
N_	Relates to NonStop SQL/MP.
NOS_	Relates to the NonStop ODBC Server.
SCS_	Relates to or references an SQL Communication Subsystem (SCS) process.
SER_	Relates to or references a server class specification that is the NOS process configured with a specific set of process and environment attributes.
SQL_	Relates to the SQL environment of a running process.
T_	Relates to ODBC or SQL Server.

Default Values for NonStop ODBC Server Attributes

Each column in the master and user catalog tables is an attribute that has a single definition. That is, a column that appears in more than one table has the same format, referential integrity, and range of possible values in each table.

Default values for attributes are dynamic and are stored in the ZNSSCFG (System Configuration) table. These values are set or accessed at four times:

- During system installation
- When the system requires a change to a default
- When a NOS (the NonStop ODBC Server object program), NOSUTIL, or SCS process begins
- During session initialization

During System Installation

At system installation, the NOSUTIL utility process creates all tables and sets the initial default values. This process adds one row to the ZNSSCFG table for every column or named attribute that it tracks. See Appendix D, Summary of System Installation Defaults, for a list of system configuration default attributes stored in the ZNSSCFG table at installation time.

When the System Requires a Change

As a privileged user, you can change the default values at any time, without taking the system down. This feature allows you to customize the default values when and as your system requirements change. The result can be improved performance. For example, if the system is using too many processors, all at high priorities, you can limit the number of processors and reduce the priorities.

By taking advantage of the changeable default values, you receive the following benefits:

- The flexibility to change attributes without requiring changes to program code
- Visibility of the results
- More control over the system

You change the default values throughout the system by using the ADD, MODIFY, or REMOVE statement on the ZNSSCFG table. For information on changing default values for the duration of a process, see “During Session Initialization.”

When a Process Begins

When the NOSUTIL, NOS, and SCS processes start up, they each initialize their program default values to the values in the ZNSSCFG table. These values are in effect for a time specified in the ZNSSCFG table. The value is stored in the VALUE column of the row whose ITEM is “CHECK_INTERVAL_SECS”.

For more information on the CHECK_INTERVAL_SECS column, see ZNSSCFG (For System Configuration Values) on page 8-28. For a list of ZNSSCFG default values, see Appendix D, Summary of System Installation Defaults.

During Session Initialization

A user can change some defaults during a session by using the SET statement. The changed values override the values on the ZNSSCFG table and are in effect for the duration of the session.   At the end of the session, they revert to the values on the ZNSSCFG table. For information on the SET statement, see the subsection SET on page 4-92.

The system configuration default values are initialized during each session initialization (by NOS) when a change has been recorded since the last time a process read the value. The LAST_UPDATE_SYSTEM_CONFIG column in the ZNSSCFG table stores the date and time of the change.

Table Mapping Security

The mapping tables in the system catalog are created when the NonStop ODBC Server is installed. The tables are created with the security “NNNO” or “NONO,” depending on whether a given table is modified during normal operations. For example, ZNSDB is assigned security “NNNO” so users can create or drop databases, adding or deleting entries in ZNSDB (SYSDATABASES); ZNUDT is different. ZNUDT is assigned security “NONO,” as its values are static for a given release of the NonStop ODBC Server.

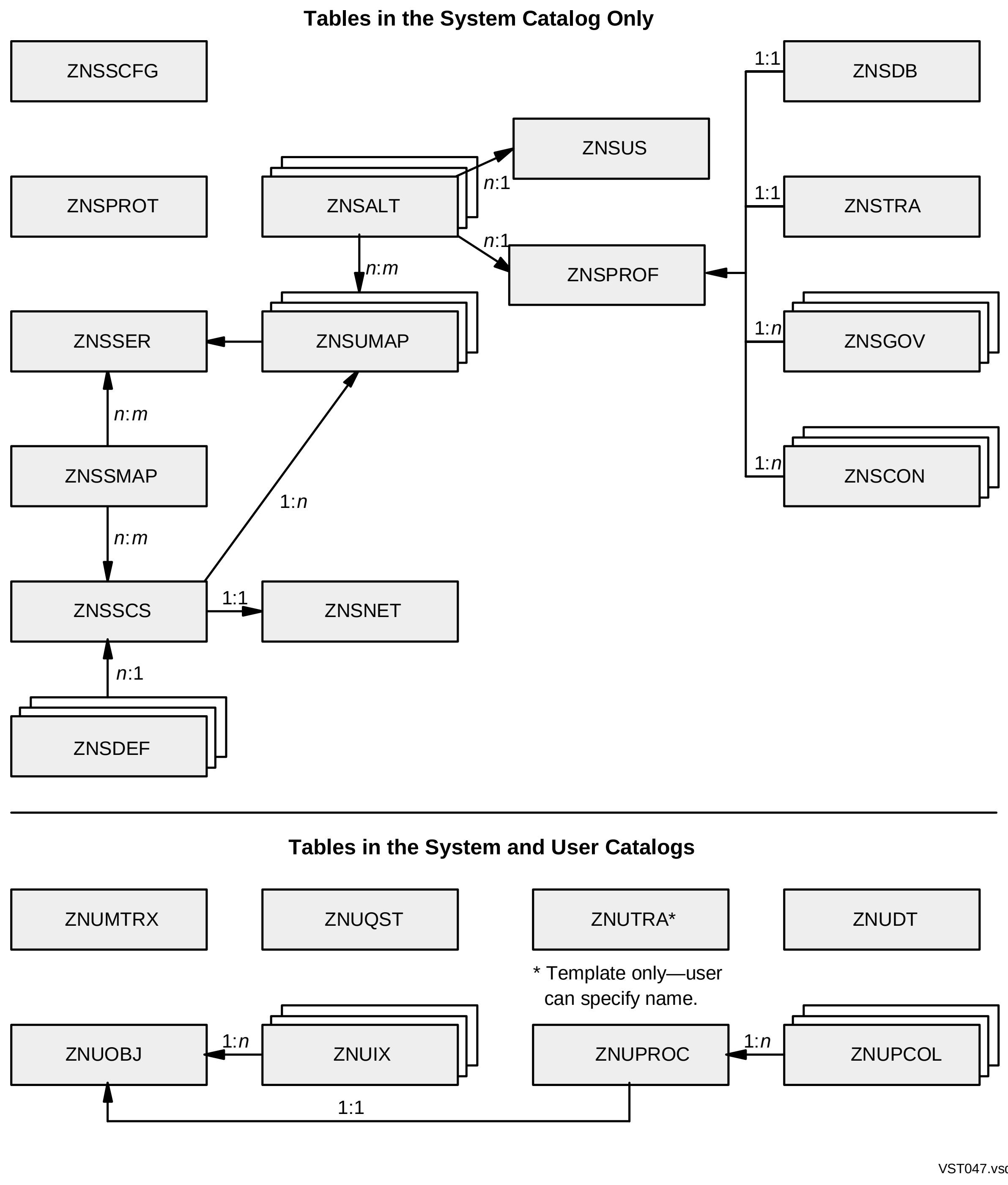
The mapping tables for user catalogs are created when an existing catalog is customized or when a database is created using the NonStop ODBC Server.

See also Renaming ODBC or SQL Server Objects on page 7-7.

Table Relationships

Figure 8-2 shows the relationships among the NonStop ODBC Server mapping tables:

Figure 8-2. Relationships Among NonStop ODBC Server Tables



Data Types

The NonStop ODBC Server logical data types described in this section are as follows:

Data Type	**Meaning**
INT (2)	2-byte integer
INT (4)	4-byte integer
INT (8)	8-byte integer
CHAR (*n*)	Fixed-length character string of *n* bytes
VARCHAR (*n*)	Variable-length character string of *n* bytes preceded by a 2-byte integer indicating the length of the string
TIMESTAMP	NonStop SQL/MP data type DATETIME YEAR TO FRACTION

Restoring Tables

A NonStop ODBC Server customized catalog is a NonStop SQL/MP catalog that has been modified using the NOSUTIL INSTALL statement or the equivalent NonStop ODBC server CREATE DATABASE command. Customization of the system catalog, as opposed to a user catalog, reflects the fact that configuration data is found only in the system catalog. The NonStop ODBC Server mapping tables in the user catalog are a subset of the NonStop ODBC Server mapping and configuration tables in the system catalog. The concept of restoring is, however, common to both. Restoring is the act of detecting missing tables, missing entity description rows in these tables, or missing foreign references in these tables, and then performing the necessary actions to correct the detected error in mapping or configuration.

The REFRESH statement is used to perform this process of restoration. REFRESH requires that the specified catalog is a customized catalog; that is, an entity description for this catalog exists in the system's ZNSDB database mapping table. An implicit requirement is that the specified catalog is a NonStop SQL/MP catalog. REFRESH begins by searching for each of the tables it uses for storing configuration and mapping data and their associated views and indexes, and re-creating the objects. REFRESH then performs a sequence of scans through these NonStop ODBC Server tables, removing foreign references that cannot be found and adding references for objects in the underlying NonStop SQL/MP catalog not registered in the NonStop ODBC Server tables.

NonStop ODBC Server configuration and mapping tables are treated by NonStop SQL/MP as user tables. For this reason, it is possible for the tables to be accidently deleted or modified, requiring REFRESH to correct the problems. Additionally, because the NonStop ODBC Server configuration and mapping tables are not updated automatically when DDL is performed on the NonStop SQL/MP catalog, REFRESH is needed to update the NonStop ODBC Server tables. When the only discrepancy is the mapping entry for a table, index, or view, object-specific commands, ADD and REMOVE, are provided to limit the restoration integrity to less than a complete REFRESH action sequence.

Database and Object Mapping

Table 8-3 and Table 8-4 are summary tables that provide the location and description of each database and object mapping table.

For a summary of system mapping tables, see System Table Mapping on page 8-60.

Table 8-3. Mapping Tables Residing Only With the System Catalog (page 1 of 2)

File Name	Object Type	Description
ZNSALT	Table	Maps alternate usernames to logical usernames.
ZNSALTI1	Index	Ensures that alternate usernames for this system are unique.
ZNSCON	Table	Describes the NonStop SQL/MP CONTROL statements that must be executed at initialization or reinitialization time.
ZNSCONI1	Index	Ensures that the lookup key relating one or more CONTROL option(s) to a user profile is unique.
ZNSDB	Table	Maps ODBC or SQL Server database names to NonStop SQL/MP catalog names.
ZNSDBI1	Index	Ensures that any database ID in ZNSDB is unique.
ZNSDBI2	Index	Is created on the T_DBID column of ZNSDB.
ZNSDEF	Table	Defines settings for process entities.
ZNSDUMMY	Table	Static table used for DLIB API.
ZNSGOV	Table	Contains accounting profile settings.
ZNSMSG	Table	Contains all error messages that could be generated by the NonStop ODBC Server.
ZNSNET	Table	Describes network service attributes.
ZNSPROF	Table	Contains user profile information for configuration and trace characteristics.
ZNSPROT	Table	Contains user permission information.
ZNSSCFG	Table	Contains system-wide configuration values.
ZNSSCS	Table	Describes SCS process attributes.
ZNSSER	Table	Describes NonStop ODBC Server server class attributes.
ZNSSMAP	Table	Describes mapping between SCS and server class
ZNSTRA	Table	Contains the user profile information for trace values.
ZNSUMAP	Table	Describes mapping between username and profile to server class.
ZNSUS	Table	Maps logical usernames to their assigned Guardian usernames.

Table 8-3. Mapping Tables Residing Only With the System Catalog (page 2 of 2)

File Name	Object Type	Description
ZNSUSI1	Index	Ensures that the logical username is unique for this system.
ZNSUSI2	Index	Ensures that the Guardian username is unique for all mappings from logical to system usernames.
ZNSVALUE	Table	Used internally by the NonStop ODBC Server to support the ODBC or SQL Server *spt_values* table.

Table 8-4 lists the mapping tables that reside in both the subvolume of the customized user catalogs and the subvolume of the NonStop SQL/MP system catalog.

Table 8-4. Mapping Tables Residing With Both the User Catalogs and the System Catalog

File Name	Object Type	Description
ZNUDT	Table	Used for data type translation to and from NonStop SQL.
ZNUIX	Table	Maps logical index names to Guardian names.
ZNUIXI1	Index	Ensures that any NonStop SQL/MP index name in ZNUIX is unique.
ZNUMTRX	Table	Used as the log of resource accounting. Is the default simple file name.
ZNUOBJ	Table	Maps ODBC or SQL Server table and view names to NonStop SQL/MP table and view names, and maps procedure names to NonStop ODBC Server procedure definitions in ZNUPROC.
ZNUOBJI1	Index	Ensures that any ODBC or SQL Server object ID in ZNUOBJ is unique.
ZNUOBJI2	Index	Ensures that any NonStop SQL/MP object name in ZNUOBJ is unique.
ZNUOBJI3	Index	Used internally by the NonStop ODBC Server.
ZNUPCLI1	Index	Ensures that any procedure parameter definition in ZNUPCOL is unique for a given procedure.
ZNUPCOL	Table	Describes parameters of stored procedures. Also provides support for ODBC catalog function SQLProcedureColumns.
ZNUPROC	Table	Associates stored procedure name with Pathway system and server class. Also provides support for the SQL Server system table SYSOBJECTS and the ODBC catalog function SQLProcedures.
ZNUQST	Table	Used for logging of resource accounting. Is a simple file name.
ZNUTRA	Table	Logs trace event messages. (The name ZNUTRA is defined only as a template. The actual name of the trace log can be any valid user-supplied name.)

The following subsections list the database and object mapping tables alphabetically and describe each table's purpose and its columns.

Many of the tables have a LAST_UPDATED column as the last column on the table. This datetime column is a timestamp that represents the time of the last add or modify executed on the table. For tables modified in batch, each row modified in the same batch of commands receives the same timestamp value. For information on batch queries, see Batch Queries on page 4-4.

In the table descriptions that follow, unless stated otherwise, primary key columns appear in the primary key in the same sequence as they appear in the table. The same is true for index columns and indexes.

Key columns and index column values are in ascending sequence.

ZNSALT (For Alternate Usernames)

The ZNSALT table maps alternate usernames to ODBC or SQL Server logical username. A user can designate different profile descriptions for use at session initiation for the same NonStop ODBC Server logical use name. More than one alias can point to the same profile and to the same logical username. A ZNSALT record is automatically added for each record added to the ZNSUS table.

ZNSALT resides only in the subvolume of the system catalog.

Table 8-5. Description of ZNSALT

Column Name	Primary Key	Unique Index	Data Type	Description
NOS_ALIASNAME	x	–	VARCHAR (60)	Alias username that can be used in place of the logical username for login (user authentication) or object reference. It is unique within a NonStop ODBC Server installation.
NOS_USERNAME	–	–	VARCHAR (60)	Logical username.
PROFILE_NAME	–	–	VARCHAR (60)	Profile to be used by the NonStop ODBC Server process when a client connects using this alias username.
LAST_UPDATED	–	–	DATETIME YEAR TO FRACTION (6)	Date and time this row was added or last modified.
CHANGE_PASSWORD_OPTION	-	-	CHAR(1)	Level of notification sent to users for expired (or about to expire) Safeguard passwords: 0-No notification is sent (the default). 1-Notification is sent if the password is in the grace period. 2-Notification is sent if the password is about to expire or if it has expired but is still in the grace period.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

The index ZNSALTI1 ensures that the NonStop ODBC Server logical username is unique for the system. ZNSALTI1 resides only in the subvolume of the system catalog.

ZNSCON (For Control Statements)

The ZNSCON table describes the NonStop SQL CONTROL directives that must be executed by the process at initialization or reinitialization time. For the syntax of NonStop SQL/MP CONTROL directives, see the *NonStop SQL/MP Reference Manual.*

ZNSCON is created during INSTALL, REFRESH, and UPGRADE operations. The table may not contain any rows. Its structure is supported to eliminate a future catalog version change.

ZNSCON resides only in the subvolume of the system catalog.

Table 8-6. Description of ZNSCON

Column Name	**Primary Key**	**Unique Index**	**Data Type**	**Description**
CON_NAME	–	–	VARCHAR (60)	Lookup key relating one or more CONTROL option(s) to a user profile.
CON_SEQ	–	–	INT (2)	Type of CONTROL directive; can be one of the following: 0 refers to the executor 1 refers to a query 2–*n* refers to a table (numbered in order of table creation)
CON_TEXT	–	–	VARCHAR (3900)	NonStop SQL/MP CONTROL directive.
LAST_UPDATED	–	–	DATETIME YEAR TO FRACTION (6)	Date and time this row was added or last modified.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

ZNSDB (For NonStop ODBC Server Databases)

The table ZNSDB maps ODBC or SQL Server database names to NonStop SQL/MP catalog names. Each row in ZNSDB represents one NonStop ODBC Server customized catalog. ZNSDB is populated by the INSTALL command.

ZNSDB resides only in the subvolume of the system catalog.

See also SYSDATABASES Catalog View (ZVSDB) on page 8-63.

Table 8-7. Description of ZNSDB

Column Name	Primary Key	Unique Index	Data Type	Description
N_CATALOG	x	–	CHAR (25)	Fully qualified NonStop SQL/MP catalog name in uppercase letters.
T_DBNAME	–	x	VARCHAR (60)	Fully qualified ODBC or SQL Server database name in uppercase letters.
T_CREATOR	–	–	INT (2) UNSIGNED	ODBC or SQL Server user ID of the database creator. If the database is created in NonStop SQL/MP and then customized, the creator is the user ID of the user who customizes the catalog.
T_DBID	–	x	INT (2) UNSIGNED	Internal ID used by the NonStop ODBC Server to satisfy requirements of TSQL users.
T_VERSION	–	–	INT (2)	NonStop ODBC Server version number.
CREATETIME	–	–	INT (8)	Date and time the catalog was created.
LAST_UPDATED	–	–	DATETIME YEAR TO FRACTION (6)	Date and time this row was added or last modified.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

The index ZNSDBI1 ensures that any database ID in ZNSDB is unique. The index is created on the T_DBID column of ZNSDB. ZNSDBI1 resides only in the subvolume of the system catalog.

The index ZNSDBI2 is created on the T_DBID column of ZNSDB. ZNSDBI2 resides only in the subvolume of the system catalog.

ZNSDEF (For NonStop ODBC Server DEFINEs)

The table ZNSDEF stores the names of the NonStop ODBC Server DEFINEs and values to be set prior to process initiation. ZNSDEF stores only attribute settings defined by the NonStop Kernel (as opposed to the ZNSSCFG table, which stores attribute settings defined by the ODBC server).

No values are loaded into ZNSDEF during system installation. The user is thereby able to establish DEFINE values to be used by the SQL executor and other collaborating processes. The NonStop ODBC server currently does not use DEFINEs as a means of setting run-time values.

ZNSDEF resides only in the subvolume of the system catalog.

Table 8-8. Description of ZNSDEF

Column Name	Primary Key	Unique Index	Data Type	Description
SCS_NAME	x	–	CHAR(16)	Process identity key.
DEFINE_NAME	x	–	CHAR (24)	NonStop Kernel DEFINE name.
DEFINE_CLASS		–	CHAR (16)	NonStop Kernel DEFINE class.
DEFINE_ATTRIBUTE	x	–	CHAR (16)	NonStop Kernel DEFINE attribute name.
DEFINE_VALUE		–	VARCHAR (512)	NonStop Kernel DEFINE attribute value.
LAST_UPDATED	–	–	DATETIME YEAR TO FRACTION (6)	Date and time this row was added or last modified.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

ZNSDUMMY (For Use With DLIB Applications)

The table ZNSDUMMY is used internally by the NonStop ODBC Server.

ZNSDUMMY resides only in the subvolume of the system catalog.

ZNSDUMMY contains one column, COL1, of data type INT. The column is used internally by the NonStop ODBC Server.

ZNSGOV (For Governing Policies)

The table ZNSGOV describes resource usage limits and server actions that occur when the limits are exceeded. The limits and actions are called governing profiles. For information on using the ZNSGOV table to log query status, see the *HP NonStop ODBC Server Installation and Management Manual.*

ZNSGOV resides only in the subvolume of the system catalog.

Table 8-9. Description of ZNSGOV (page 1 of 2)

Column Name	Primary Key	Unique Index	Data Type	Description
GOV_NAME	x	–	VARCHAR (60)	Unique governing profile name.
GOV_ATTRIBUTE	x	–	VARCHAR (60)	Process attribute for which the governing policy applies (see definitions following the table): ELAPSED_TIME ESTIMATED_COST EXECUTION_TIME ROWS_ACCESSED ROWS_USED
LIMIT_VALUE	x	–	INT (8)	Threshold value that, if exceeded, causes the governing action to occur. Valid values are from 0 to the maximum allowed for the column's data type.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

Table 8-9. Description of ZNSGOV (page 2 of 2)

Column Name	Primary Key	Unique Index	Data Type	Description
GOV_ACTION	–	–	VARCHAR (60)	Action to be taken if the process attribute exceeds the threshold value (see definitions following table): COMMIT CONTINUE PRIORITY [++ | --] *n* ROLLBACK STOP
LOG_QST_ON	–	–	CHAR (1)	Flag indicating whether governing actions are logged. (Can be overridden by the value in the QST_MODE_ON column in the ZNSPROF table.)
LAST_UPDATED	–	–	DATETIME YEAR TO FRACTION (6)	Date and time this row was added or last modified.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

The following are descriptions of the valid values for GOV_ATTRIBUTE:

Value	Description
ELAPSED_TIME	Wall clock time from the beginning to the end of the query
ESTIMATED_COST	Estimated cost for the query execution
EXECUTION_TIME	CPU usage time for this query
ROWS_ACCESSED	Number of rows accessed for this query
ROWS_USED	Number of rows fetched for this query

The following are descriptions of the valid values for GOV_ACTION:

Value	Description
COMMIT	The query is committed.
CONTINUE	The query continues as if nothing happened.
PRIORITY [++ | --] *n*	The priority of the query is increased or decreased by *n*., or set to *n*
ROLLBACK	The query is stopped, all resources associated with the statement are released, and the query is rolled back.
STOP	The query is stopped.

ZNSMSG (For Error Messages)

The table ZNSMSG contains all error messages that can be generated by the NonStop ODBC Server. The table maps a NonStop ODBC Server error code to its corresponding ODBC or SQL Server error code (where possible). It also contains error text and the ODBC or SQL Server severity level. For details on the error messages, see the *HP NonStop ODBC Server Messages Manual*.

ZNSMSG is a static table created but not populated at system installation time. It is loaded during the customization of catalogs (SYSCAT INSTALL). It serves TSQL users, but not ODBC users.

ZNSMSG resides only in the subvolume of the system catalog.

See also SYSMESSAGES Catalog View (ZVSMSG) on page 8-66.

Table 8-10. Description of ZNSMSG (page 1 of 2)

Column Name	**Primary Key**	**Unique Index**	**Data Type**	**Description**
N_PRODNUM	–	–	CHAR (5)	NonStop ODBC Server product number.
N_VERSION	–	–	CHAR (3)	NonStop ODBC Server version number.
N_ERROR_CODE	x	–	INT (4)	Error code generated by the NonStop ODBC Server or NonStop SQL/MP.
N_LANGUAGE	x	–	INT (2)	Not used.
N_SPECSYM	–	–	VARCHAR (4)	Special symbol (~) used by the error handler for error parameter substitutions.
T_ERROR_CODE	–	–	INT (4)	ODBC or SQL Server error code that corresponds to this message. If there is no corresponding ODBC or SQL Server error code, the code is 18001.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

Table 8-10. Description of ZNSMSG (page 2 of 2)

Column Name	Primary Key	Unique Index	Data Type	Description
T_CLASS	–	–	INT (2)	ODBC or SQL Server error severity level: 10 Warning 11–16 User-generated 17–18 Nonfatal error 19+ Fatal error
T_STATE	–	–	INT (2)	ODBC or SQL Server error detail level. This is not supported and is always –1.
N_ERROR_TEXT	–	–	VARCHAR (400)	Error text, which can be ODBC or SQL Server error text, NonStop SQL/MP text, or NonStop ODBC Server error text.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

ZNSNET (For Network Service Specifications)

The table ZNSNET contains the data used by the SCS process to initiate a service. Attributes are mutually exclusive according to the network protocol. This means that the table contains certain values if the protocol is TCP/IP and certain other values if the protocol is NETBIOS.

ZNSNET resides only in the subvolume of the system catalog.

Table 8-11. Description of ZNSNET (page 1 of 2)

Column Name	Primary Key	Unique Index	Data Type	Description
SCS_NAME	x	–	VARCHAR (60)	Unique value used by SCS as a foreign key.
NET_PROTOCOL	–	–	VARCHAR (60)	Network protocol: TCP/IP NetBIOS SPX/IPX
NET_NAME	–	–	CHAR (60)	Uppercase name format determined by NET_PROTOCOL. If NET_PROTOCOL is TCP/IP or SPX/IPX, the value must be a 1- to 16-character name in the TCP/IP System Services file. If the protocol is NetBIOS, the value must be a 1 to 12-character name.
SERVICES_FILENAME	–	–	CHAR (35)	Guardian file name indicating the location of the SERVICES file, when the location is not $SYSTEM.ZTCPIP.SERVICES. Used only if NET_PROTOCOL is TCP/IP or SPX/IPX.
IOP_NAME	–	–	CHAR (16)	Network-qualified IOP process name. Format is \node.$*iopnm*.
SO_KEEPALIVE	–	–	INT (4)	Startup value for a socket option. (See the *TCP/IP Applications and Utilities User Guide.)*

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

Table 8-11. Description of ZNSNET (page 2 of 2)

Column Name	Primary Key	Unique Index	Data Type	Description
SO_OOBINLINE	–	–	INT (4)	Startup value for a socket option. (See the *TCP/IP Applications and Utilities User Guide.)*
SO_LINGER	–	–	INT (4)	Startup value for a socket option. (See the *TCP/IP Applications and Utilities User Guide.)*
SO_REUSEADDR	–	–	INT (4)	Startup value for a socket option. (See the *TCP/IP Applications and Utilities User Guide.)*
NET_QUALIFIER	–	–	VARCHAR (128)	Service name, registered in the Multilan domain server, to which a client will connect. Used only if NET_PROTOCOL is NetBIOS. Format is \PIPE\SQL\QUERY\name (See the *Multilan Management Programming Manual.)*
MLAN_DOMAIN	–	–	VARCHAR (60)	Multilan/TLAM subdevice name. Used only if NET_PROTOCOL is NetBIOS. (See the *Multilan Management Programming Manual.)*
MLAN_ADAPTOR	–	–	INT (2)	Number of adaptor to be used on Multilan hardware card. Used only if NET_PROTOCOL is NetBIOS. Valid values are 0 and 1. (See the *Multilan Management Programming Manual.)*
MLAN_GATEWAY	–	–	CHAR (5)	Multilan/TLAM gateway. Used only if NET_PROTOCOL is NetBIOS. (See the *Multilan Management Programming Manual.)*
LAST_UPDATED	–	–	DATETIME YEAR TO FRACTION (6)	Date and time this row was added or last modified.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

ZNSPROF (For Profile Values)

The table ZNSPROF contains the user profile information used by the server to establish session-based defaults.

There is one ZNSPROF table per system, and it resides only in the subvolume of the system catalog.

ZNSPROF contains one row per user profile. At system installation time, it is loaded with a special profile, named DEFAULT, that has set in it the default values for each profile attribute. This profile is used to allow unknown (not added) users to connect with a default profile.

Table 8-12. Description of ZNSPROF (page 1 of 4)

Column Name	Primary Key	Unique Index	Data Type	Description
PROFILE_NAME	x	–	VARCHAR (60)	User-defined profile name that is unique in the system in which it is defined.
DEFAULT_DATABASE	–	–	VARCHAR (60)	Default database for object qualification in the NonStop ODBC server or in NOSUTIL for the user who indicates this profile. Internal default is MASTER, which allows initialization to a NonStop ODBC Server customized database without requiring direct user action during the user session.
DEFAULT_SCHEMA	–	–	VARCHAR (60)	Default schema name for object qualification (owner) in the NonStop ODBC server or in NOSUTIL for the user indicating this profile. This value affects only schema unqualified object references; it does not override an application-provided schema qualifier. If blank (length 0), object qualification uses the NOS_USERNAME value determined at login.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

Table 8-12. Description of ZNSPROF (page 2 of 4)

Column Name	Primary Key	Unique Index	Data Type	Description
DEFAULT_LOCATION	–	–	VARCHAR (60)	Default location for the creation of objects. When a user creates an object, it appears by default in the same subvolume as the NonStop ODBC Server customized catalog. The DEFAULT_LOCATION entry causes the object to be created on the specified volume instead. The value is defined as a disk volume name on the local node.
DEFAULT_SECURITY	–	–	CHAR (4)	Default security vector used during object creation.
TRA_MODE_ON	–	–	CHAR (1)	Flag indicating whether tracing is performed.
TRA_NAME	–	–	VARCHAR (60)	Trace configuration.
ACC_MODE_ON	–	–	CHAR (1)	Flag indicating whether resource accounting is performed.
ACC_LOGTABLE_ NAME	–	–	VARCHAR (182)	NonStop ODBC Server table object name.
ACC_LEVEL	–	–	CHAR (60)	Indicates accounting recording frequency: SESSION Summarize session SQL_STATEMENT Record each statement
GOV_MODE_ON	–	–	CHAR (1)	Flag indicating whether resource governing is performed.
GOV_NAME	–	–	VARCHAR (60)	Governing configuration.
QST_MODE_ON	–	–	CHAR (1)	Flag indicating whether query status is enabled.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

Table 8-12. Description of ZNSPROF (page 3 of 4)

Column Name	Primary Key	Unique Index	Data Type	Description
SQL_ACCESS_MODE	–	–	CHAR (2)	Default mode in the current session: RW Read-write RO Read-only
SQL_CURSOR_MODE	–	–	CHAR (2)	SQL cursor mode. Forces “FOR UPDATE” on cursor specifications that do not explicitly indicate BROWSE: RO Cursor defaults to FOR READ ONLY RW Cursor defaults to FOR UPDATE
SQL_DIALECT	–	–	CHAR (8)	Dialect affecting the NonStop ODBC Server parser. Is either TDM_CORE or TDM_TSQL.
SQL_MAX_ STATEMENT_CACHE	–	–	INT (2)	Upper limit on the number of SQL statements saved in the NonStop ODBC Server cache.
SQL_TXN_ISOLATION	–	–	CHAR (1)	Initial transaction isolation level.
SQL_UNSUPPORTED	–	–	CHAR (1)	Return message level for syntax errors and warnings: E Report as error W Report as warning I Ignore
OBJ_NAME_CACHE	–	–	CHAR (1)	Object name cache behavior.
STMT_CACHE_LEVEL	–	–	INT (4)	Reserved.
CON_MODE_ON	–	–	CHAR (1)	Flag indicating whether CONTROL statements apply to the session.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

Table 8-12. Description of ZNSPROF (page 4 of 4)

Column Name	Primary Key	Unique Index	Data Type	Description
CON_NAME	–	–	VARCHAR (60)	CONTROL statement list identifier.
CLOSE_TABLES_PER_ SESSION	–	–	CHAR (1)	Flag (Y or N) indicating whether NonStop SQL/MP tables are closed at the end of a session
LAST_UPDATED	–	–	DATETIME YEAR TO FRACTION (6)	Date and time this row was added or last modified.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

ZNSPROT (For User Permission Data)

The table ZNSPROT contains user permission information. It has one row for each combination of T_ACTION and T_PROTECTTYPE. The table is implemented to provide compatibility with the SQL Server system table SYSPROTECTS. The NonStop ODBC Server, however, does not support the statements that affect SYSPROTECTS, so ZNSPROT grants all users authority to use all statements.

ZNSPROT resides only in the subvolume of the system catalog.

See also SYSPROTECTS Catalog View (ZVUPROT) on page 8-69.

Table 8-13. Description of ZNSPROT

Column Name	Primary Key	Unique Index	Data Type	Description
T_ACTION	–	–	INT (2)	An integer corresponding to a Transact-SQL statement: 193 SELECT 195 INSERT 196 DELETE 197 UPDATE 224 EXECUTE
T_PROTECTTYPE	–	–	INT (2)	Protection type. Contains the integer 205, which signifies GRANT permission.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

ZNSSCFG (For System Configuration Values)

The table ZNSSCFG stores configuration values and bookkeeping data on changes made to these values. ZNSSCFG stores only attribute settings defined by the ODBC server (as opposed to ZNSDEF, which stores attribute settings defined by the NonStop Kernel), When NOSUTIL installs the NonStop ODBC Server system catalogs, it loads the system default configuration values into the ZNSSCFG table.

ZNSSCFG resides only in the subvolume of the system catalog.

The ZNSSCFG table stores three values that are not attribute default values:

- LAST_UPDATE_SYSTEM_CONFIG,
- CHECK_INTERVAL_SECONDS,
- QST_LOGTABLE_NAME

LAST_UPDATE_SYSTEM_CONFIG represents the last time the administrator wanted to make available changes to the system configuration as represented by rows in the configuration tables. The administrator can cause this value to be automatically updated by indicating UPDATE_SYSTEM_CONFIG when performing ADD, MODIFY, or REMOVE statements against the configuration entities, or more explicitly by using MODIFY on ZNSSCFG to change the LAST_UPDATE_SYSTEM_CONFIG item.

Whenever a row is added to any of these tables or whenever a change in any column in these tables occurs, these actions are reflected in the ZNSSCFG table as follows:

- The column name is stored in the ITEM column.
- The new value is stored in the VALUE column.
- The LAST_UPDATED column is updated with the date and time of the update or the addition.

Note that each value in ZNSSCFG is stored in ASCII format. When a value is read from ZNSSCFG to be used as a default, it is first converted to the target data type.

Values in ZNSSCFG are used only when creating an entity (ADD or CREATE) or when starting system processes (START SCS).

The database administrator can change the values on the ZNSSCFG table by using the ADD, MODIFY, or REMOVE statement at any time, without bringing the system down. The changes affect the default values throughout the system. If you change a value on the ZNSSCFG table, be aware that changes to the ZNSSCFG table values are not propagated to the other mapping tables.

Using the SET statement, you can override values in the ZNSSCFG table for the duration of a session. The SCS, NOS, and NOSUTIL programs use the new default values. When the session ends, the default values revert to the values in ZNSSCFG.

Storage of System Information

ZNSSCFG contains two special rows that represent data that pertains to the entire collection of data stored on ZNSSCFG. These are the rows that contain the values "LAST_UPDATE_SYSTEM_CONFIG" and "CHECK_INTERVAL_SECONDS" in the ITEM column. The LAST_UPDATE_SYSTEM_CONFIG row describes the last time the system configuration was updated and the CHECK_INTERVAL_SECONDS row tells for how long the update is effective.

The term "default" is dynamic within a system installation. When a process begins, the programs retrieve the values from the ZNSSCFG table. These values are in effect for a time specified in the VALUE column for the ITEM that has a value of "CHECK_INTERVAL_SECS" in the ZNSSCFG table.

Valid values are as follows:

- "–1" means no polling takes place to determine whether the value is still in effect.
- "0" means the value of the column or attribute is in effect for the length of the process.
- A number other than "–1" or "0" represents the number of seconds the value for the column or attribute is in effect. Usually the number is fairly high, such as "3000."

Duration of Default Values

The time that a default value is in effect is measured from the time the CHECK_INTERVAL_SECONDS value was last updated.

The NOS, SCS, and NOSUTIL processes use the CHECK_INTERVAL_SECS value in different ways:

- The NOS process checks the value of CHECK_INTERVAL_SECS at the beginning of a session and does not check it through the remainder of the session when it ends with a disconnect.
- Once an SCS process begins, SCS may check the value for CHECK_INTERVAL_SECS more than once during a session. The SCS process uses the NOSUTIL process to perform this check.
- NOSUTIL checks the value in CHECK_INTERVAL_SECS with every message it receives.

If the value in CHECK_INTERVAL_SECONDS shows that the value for the column or attribute is no longer in effect, the process (NOS, SCS, or NOSUTIL) determines a course of action.

When No Default Values Are Found

The NOSUTIL, NOS and SCS processes have some default values built into their programs. Whenever a NOSUTIL, NOS, or SCS process cannot find a default value on ZNSSCFG, the process continues by using the default values built into their programs.

If the default value for the attribute on the ZNSSCFG table is “None,” the process reports an error. If the process is SCS or NOS, the process stops. If the process is NOSUTIL, the command fails.

For more information on default values, see Default Values for NonStop ODBC Server Attributes on page 8-4. For a list of the system configuration default values at system installation time, see Appendix D, Summary of System Installation Defaults.

Table 8-14. Description of ZNSSCFG

Column Name	Primary Key	Unique Index	Data Type	Description
ITEM	x	–	VARCHAR (60)	Column or attribute name.
VALUE	–	–	VARCHAR (240)	Default value for the column or attribute.
LAST_UPDATED	–	–	DATETIME YEAR TO FRACTION (6)	Last time this component of the system configuration was changed.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

You can update the LAST_UPDATE_SYSTEM_CONFIG value in two ways:

- Execute an ADD, MODIFY, or REMOVE statement on any other configuration component, using the optional parameter for updating the system configuration.
- Execute a MODIFY statement on the LAST_UPDATE_SYSTEM_CONFIG row.

Recovering From SQL Errors

You can use items in the ZNSSCFG table to specify that the system take a particular action when an SQL error or warning that you specify is encountered. To enable this feature, add the COLLECT_SQL_ERROR_INFO attribute to the ZNSSCFG table and set its value to Y (setting its value to N disables the feature), as follows:

```
ADD SCFG COLLECT_SQL_ERROR_INFO “Y”
```

Then add the SQL_ERROR_*sqlcode* or SQL_WARNING_*sqlcode* attribute, where *sqlcode* is the error or warning code of interest, and specify an action code. To cause NOS to abend when it encounters SQL error 8013, enter the following:

```
ADD SCFG SQL_ERROR_8013 “1” UPDATE_SYSTEM_CONFIG
```

To cause the system to create a SAVEABEND file when NOS abends, set the NOS_DEBUG_OPTIONS attribute in the serverclass to 4. For example:

```
MODIFY SERVERCLASS serverclass-name NOS_DEBUG_OPTIONS 4
```

The attributes COLLECT_SQL_ERROR_INFO, SQL_ERROR_*sqlcode*, and SQL_WARNING_*sqlcode* are not available in the ZNSSCFG table by default; they are only present when added using the ADD SCFG command.

ZNSSCS (for SQL Communication Subsystem Specifications)

The ZNSSCS table stores the definitions of the SCS process configuration. (It replaces the user-specified edit files used in previous versions of NonStop ODBC.)

You can specify any value in the configuration that is used by the PROCESS_CREATE_ procedure or can be passed in the startup message to the SCS process. For columns with a PROCESS_CREATE_ option in the description, see the *Guardian Procedure Calls Reference Manual* for a definition.

ZNSSCS resides only in the subvolume of the system catalog.

Table 8-15. Description of ZNSSCS (page 1 of 2)

Column Name	Primary Key	Unique Index	Data Type	Description
SCS_NAME	x	–	CHAR (16)	Unique SCS name.
JOB_ID	–	–	INT (2)	PROCESS_CREATE_ option.
SCS_OBJECT	–	–	VARCHAR (35)	Qualified file name.
SCS_LIBRARY_FILE	–	–	VARCHAR (35)	PROCESS_CREATE_ option.
SCS_PRIORITY	–	–	INT (2)	PROCESS_CREATE_ option.
CPU_PRIMARY	–	–	INT (2)	PROCESS_CREATE_ option.
CPU_BACKUP	–	–	INT (2)	PROCESS_CREATE_ option.
SWAPVOL	–	–	VARCHAR (17)	PROCESS_CREATE_ option.
DATAPAGES	–	–	INT (2)	Common Kernel startup option.
EXT_SWAPFILE	–	–	VARCHAR (35)	PROCESS_CREATE_ option.
SCS_CREATE_ OPTIONS	–	–	INT (2)	PROCESS_CREATE_ option.
EMIT_EVENTS	–	–	CHAR (1)	Emit EMS events option.
PIPE_TEST	–	–	CHAR (1)	Obsolete; not used.
MEMORY_CHECK	–	–	CHAR (1)	SCS diagnostic parameter.
HOMETERM	–	–	VARCHAR (24)	PROCESS_CREATE_ option.
IN_FILE	–	–	VARCHAR (35)	PROCESS_CREATE_ option.
OUT_FILE	–	–	VARCHAR (35)	PROCESS_CREATE_ option.
ERR_FILE	–	–	VARCHAR (35)	PROCESS_CREATE_ option.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

Table 8-15. Description of ZNSSCS (page 2 of 2)

Column Name	Primary Key	Unique Index	Data Type	Description
IN_BUFFER_SIZE_B	–	–	INT (2)	Byte count in the IPC send buffer between SCS and NOS and between NOS and NOSUTIL.
OUT_BUFFER_SIZE_B	–	–	INT (2)	Byte count in the IPC return buffer between SCS and NOS and between NOS and NOSUTIL.
DEFAULT_VOLUME	–	–	VARCHAR (26)	PROCESS_CREATE_ option.
SCS_DEBUG_OPTIONS	–	–	INT (2)	PROCESS_CREATE_ option.
SCS_RUN_OPTIONS	–	–	VARCHAR (240)	SCSOBJ startup message.
NOS_OBJECT	–	–	CHAR (34)	Qualified file name.
NOS_LIBRARY_FILE	–	–	CHAR (35)	PROCESS_CREATE_ option.
NOSUTIL_OBJECT	–	–	CHAR (34)	Qualified file name.
NOSUTIL_LIBRARY_FILE	–	–	CHAR (35)	PROCESS_CREATE_ option.
NOSUTIL_PRIORITY	–	–	INT (2)	PROCESS_CREATE_ option.
NOSUTIL_CPU	–	–	INT (2)	PROCESS_CREATE_ option.
NOSUTIL_CREATE_OPTIONS	-	-	INT (2)	PROCESS_CREATE_ option.
NOSUTIL_DEBUG_OPTIONS	–	–	INT (2)	PROCESS_CREATE_ option.
NOSUTIL_RUN_OPTIONS	–	–	VARCHAR (240)	NOSUTIL start message.
SCS_STOPPED	–	–	CHAR (1)	Flag used internally for the graceful shutdowm of SCS. It can be set only with START SCS or STOP SCS. For more information, see STOP SCS on page 7-102.
LAST_UPDATED	–	–	DATETIME YEAR TO FRACTION (6)	Date and time this row was added or last modified.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

ZNSSER (For NonStop ODBC Server Process Definitions)

The ZNSSER table stores the definitions of the NonStop ODBC server process configuration by server class. It replaces the user-specified edit files used in previous versions. You can specify any value that is used by PROCESS_CREATE_ or any value that can be passed in the startup message to the SCS process. For more information about columns with a PROCESS_CREATE_ option in the description, see the *Guardian Procedure Calls Reference Manual.*

ZNSSER resides only in the subvolume of the system catalog.

Table 8-16. Description of ZNSSER (page 1 of 2)

Column Name	Primary Key	Unique Index	Data Type	Description
SER_NAME	x	–	CHAR (32)	Server class name.
PRIORITY	–	–	INT (2)	PROCESS_CREATE_ option.
CPU_LIST	–	–	VARCHAR (37)	Comma-delimited list of CPUs to be used by the SCS parent.
AVAILABLE_ SERVERS	–	–	INT (2) UNSIGNED	Number of available servers in this named server class.
MAX _SERVERS	–	–	INT (2) UNSIGNED	Maximum number of servers allowed in this named server class.
G_USERNAME	–	–	CHAR (17)	Guardian username of the owner of the server class in the following format: `group.user`
PROFILE_NAME	–	–	VARCHAR (60)	Profile name to be used by the NonStop ODBC Server at startup initialization.
INIT_HEAP_SIZE_KB	–	–	INT (2)	Number of 1024-byte blocks allocated in the NonStop ODBC Server at startup initialization.
MAX_HEAP_SIZE_KB	–	–	INT (2)	Limit of 1024-byte blocks the NonStop ODBC Server can allocate over the life of the process.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

Table 8-16. Description of ZNSSER (page 2 of 2)

Column Name	Primary Key	Unique Index	Data Type	Description
IDLE_DELETE_ DELAY_SEC	–	–	INT (4)	Number of seconds a process can remain idle before being deleted, if more than the minimum number of servers exist (as defined by the value for MIN_SERVERS). If 0 seconds, the process is not deleted.
NOS_CREATE_ OPTIONS	–	–	INT (2)	PROCESS_CREATE_ option.
NOS_DEBUG_ OPTIONS	–	–	CHAR (1)	PROCESS_CREATE_ option.
NOS_RUN_ OPTIONS	–	–	VARCHAR (240)	NonStop ODBC Server startup message.
SWAPVOL	–	–	CHAR (17)	PROCESS_CREATE_ option.
LOGIN_TIMEOUT_ SEC	-	-	INT(4)	Number of seconds SCS waits before sending a logon denied message to a client who is trying to logon to a NonStop ODBC server.
CANCEL_TIMEOUT_ SEC	-	-	INT (2)	Number of seconds SCS waits before stopping a NOS process after an SQLCancel request.
LAST_UPDATED	–	–	DATETIME YEAR TO FRACTION (6)	Date and time this row was added or last modified.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

ZNSSMAP (For Server Class to SCS Description Mapping)

The table ZNSSMAP links the server class description with an SCS description.

ZNSSMAP resides only in the subvolume of the system catalog.

Table 8-17. Description of ZNSSMAP

Column Name	Primary Key	Unique Index	Data Type	Description
SCS_NAME	x	–	CHAR (16)	SCS name.
SER_NAME	x	–	CHAR (32)	Server class name.
USE_AS_DEFAULT	–	–	CHAR (1)	Flag indicating whether the server class specification is the default for this SCS.
SHUTDOWN_SER	–	–	CHAR (1)	Flag indicating whether this server class specification is removed from this SCS configuration.
LAST_UPDATED	–	–	DATETIME YEAR TO FRACTION (6)	Date and time this row was added or last modified.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

ZNSTRA (For Trace Data)

The table ZNSTRA contains the trace description pointed to by a user profile.

Table 8-18. Description of ZNSTRA (page 1 of 2)

Column Name	Primary Key	Unique Index	Data Type	Description
TRA_NAME	x	–	VARCHAR (60)	A unique key for associating a trace definition with a profile.
TRA_LOGTABLE_ NAME	–	–	VARCHAR (182)	NonStop ODBC Server table name to which the log is written.
TRA_LEVEL	–	–	VARCHAR (60)	Reserved for future use.
LOG_TO_ HOMETERM	–	–	CHAR (1)	Record copying trace indicator: Y TRACE records are copied to the HOMETERM of the NSODBC server process N TRACE records are not copied to the HOMETERM
INPUT_STREAM	–	–	CHAR (1)	Read messages trace indicator: Y TRACE records each message read by the NSODBC server N TRACE does not record each message read
OUTPUT_STREAM	–	–	CHAR (1)	Written messages trace indicator: Y TRACE records each message written by the NSODBC server N TRACE does not record each message written by the NSODBC server.
NSSQL	–	–	CHAR (1)	SQL statement trace indicator: Y SQL statements prepared or executed are traced N SQL statements are not traced

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

Table 8-18. Description of ZNSTRA (page 2 of 2)

Column Name	Primary Key	Unique Index	Data Type	Description
TRA_ERROR	–	–	CHAR (1)	Error condition trace indicator: Y Error conditions are traced N Error conditions are not traced
CACHE_STATISTICS	–	–	CHAR (1)	Cache statistics trace indicator: Y SQL statement cache success/miss statistics are traced N SQL statement cache success/miss statistics are not traced
SP_WRITE	–	–	CHAR (1)	Writes to stored procedures indicator: Y Records written to stored procedures are traced N Records written to stored procedures are not traced
SP_READ	–	–	CHAR (1)	Reads from stored procedures indicator: Y Records read from stored procedures are traced N Records read from stored procedures are not traced
LAST_UPDATED	–	–	DATETIME YEAR TO FRACTION (6)	Date and time this row was last updated.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

ZNSUMAP (For User and Profile Name to Server Class Mapping)

The ZNSUMAP table links the username and profile name in the login record (determined from the ZNSALT table) and the server class (SER_NAME) to which the login message is routed.

ZNSUMAP resides only in the subvolume of the system catalog.

Table 8-19. Description of ZNSUMAP

Column Name	Primary Key	Unique Index	Data Type	Description
SCS_NAME	x	–	CHAR (16)	NonStop SCS process name.
NOS_ALIASNAME	x	–	VARCHAR (60)	NonStop ODBC Server alias name.
SER_NAME	–	–	CHAR (32)	NonStop ODBC Server server class name.
LAST_UPDATED	–	–	DATETIME YEAR TO FRACTION (6)	Date and time this row was added or last modified.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

ZNSUS (For Logical Username to Guardian Username Mapping)

The ZNSUS table maps logical ODBC or SQL Server usernames to their Guardian username counterparts.

ZNSUS resides only in the subvolume of the system catalog.

Table 8-20. Description of ZNSUS

Column Name	Primary Key	Unique Index	Data Type	Description
NOS_USERNAME	x	–	VARCHAR (60)	Logical username.
G_USERNAME	–	x	CHAR (17)	Corresponding Guardian username for the logical username.
NOS_UID	–	x	INT (2) UNSIGNED	A unique ID, system-generated during ADD USER.
LAST_UPDATED	_	_	DATETIME YEAR TO FRACTION (6)	Date and time this row was added or last modified.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

The ZNSUSI1 index ensures that the NonStop ODBC Server NOS_UID is unique for the system. ZNSUSI1 resides only in the subvolume of the system catalog.

The ZNSUSI2 index ensures that the Guardian username is unique for these user mappings. ZNSUSI2 resides only in the subvolume of the system catalog.

ZNSVALUE (For ODBC or SQL Server `spt_values` Support)

The table ZNSVALUE is used internally by the NonStop ODBC Server to support the ODBC or SQL Server *`spt_values`* table. The table is used by DBLIB applications.

The table contains one column called NUMBER, of data type INT (2). The table has 16 rows. The NUMBER column contains values from 1 to 16. The column is used internally by the NonStop ODBC Server.

ZNSVALUE resides only in the subvolume of the system catalog.

ZNUDT (For Data Types Mapping)

The static table ZNUDT maps ODBC or SQL Server data types to NonStop SQL/MP data types and vice versa. ZNUDT, TABLES, COLUMNS, and ZNUOBJ are joined to form the view ZVUCOL, which corresponds to the system table SYSCOLUMNS. ZNUDT is used only for this purpose.

ZNUDT is loaded during SYSCAT INSTALL or USERCAT INSTALL. It describes the mapping between NonStop SQL and the SQL Server API and the ODBC API. It corresponds to the TDS encoding/decoding routines in the NonStop ODBC Server. It is used in the view that generates the result set for the DBLIB API, SYSCOLUMNS, or the ODBC CLI function SQLColumns.

ZNUDT resides in each user catalog.

See also SYSTYPES Catalog View (ZVUDT) on page 8-70 and SQLColumns Procedure (ZVUOCOL) on page 8-60.

Table 8-21. Description of ZNUDT (page 1 of 2)

Column Name	Primary Key	Unique Index	Data Type	Description
N_DTYPE	x	–	CHAR (18)	NonStop SQL/MP data type code to join with the COLUMNS table.
N_PRECLO	x	–	INT (2)	Low end of the precision range (0 if not applicable).
N_PRECHI	x	–	INT (2) UNSIGNED	High end of the precision range (0 if not applicable).
N_SCALELO	x	–	INT (2)	Low end of the scale range (0 if not applicable).
N_SCALEHI	x	–	INT (2)	High end of the scale range (0 if not applicable).
T_DTYPE	–	–	INT (2)	ODBC or SQL Server data type code.
T_USERTYPE	–	–	INT (2)	ODBC or SQL Server user-defined data type code.
T_ODBCTYPE	–	–	INT (2)	ODBC user-defined data type code.
T_NAME	x	–	VARCHAR (30)	ODBC or SQL Server data type name, such as INT, FLOAT, or DATETIME.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

Table 8-21. Description of ZNUDT (page 2 of 2)

Column Name	Primary Key	Unique Index	Data Type	Description
T_NULLABLE	–	–	INT (2)	Indicates whether the ODBC or SQL Server data type allows null values.
N_NULLABLE	x	–	CHAR (1)	Indicates whether the NonStop SQL/MP data type allows null values.
T_MODE	x	–	CHAR (1)	Indicates whether this data type applies to TDM_TSQL mode or TDM_ODBC mode: T TSQL definition O ODBC definition
NOS_DISPLAY	–	–	CHAR (1)	SYSTYPES display indicator: " " Do not display type in SYSTYPES view T Display type in SYSTYPES
LAST_UPDATED	–	–	DATETIME YEAR TO FRACTION (6)	Date and time this row was last updated.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

ZNUIX (For Logical Index Names to Guardian Names Mapping)

The table ZNUIX maps ODBC or SQL Server index names to NonStop SQL/MP index names. This mapping is needed because ODBC or SQL Server index names are bound to their table names and need not be unique, whereas NonStop SQL/MP index names must be unique.

ZNUIX resides in both the subvolume of the system catalog and a customized user catalog.

Table 8-22. Description of ZNUIX

Column Name	Primary Key	Unique Index	Data Type	Description
T_INAME	x	–	CHAR (60)	Fully qualified index name in uppercase letters.
T_TABLEID	x	–	INT (4)	Object ID (T_OBJID in ZNUOBJ) of the table.
T_STATUS	–	–	INT (2)	Used internally by the NonStop ODBC Server to record whether an index is unique.
N_INAME	–	x	CHAR (34)	Fully qualified NonStop SQL/MP index name in uppercase letters.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

The index ZNUIXI1 ensures that any NonStop SQL/MP index name in ZNUIX is unique. The index is created on the N_INAME column of ZNUIX.

ZNUIXI1 resides in both the subvolume of the system catalog and a customized user catalog.

ZNUMTRX (For Resource Accounting Log Data)

The ZNUMTRX table is the default simple file name used as the log for resource accounting. It is the log table name ACC_LOGTABLE_NAME found in the ZNSPROF record, pointed to by the current user profile. When creating the resource accounting log, you can specify any other simple file name, however the definition of the table is fixed.

ZNUMTRX resides in both the subvolume of the system catalog and a customized user catalog.

The NOS process enters all values into this table.

The ZNUMTRX table is created as an unaudited table so that concurrency among multiple users is maximized. A log entry row only appends to the table during the time the log is active. For information on activating the log, see the *HP NonStop ODBC Server Installation and Management Manual.*

The result sets referred to in some of the descriptions in the ZNUMTRX table are the output that results from execution of a SELECT statement.

Table 8-23. Description of ZNUMTRX (page 1 of 6)

Column Name	Primary Key	Unique Index	Data Type	Description
CLIENT_ID	–	–	VARCHAR (60)	A unique identifier that links this accounting log sequence to a client session.
SESSION_ID	–	–	INT (8) UNSIGNED	A unique identifier that associates log entries within an accounting activity.
START_TIME	–	–	TIMESTAMP	Time the accounting activity started (milliseconds).
END_TIME	–	–	TIMESTAMP	Time the accounting activity stopped (milliseconds).
LOGON_USERNAME	–	–	VARCHAR (60)	Username provided by the user, stored in the login record; used to determine the environment settings for the current session.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

Table 8-23. Description of ZNUMTRX (page 2 of 6)

Column Name	Primary Key	Unique Index	Data Type	Description
APPLICATION	–	–	VARCHAR (30)	Application provided by the user, stored in the login record; used to determine the environment settings for the current user session.
G_USERNAME	–	–	CHAR (17)	Guardian username determined from the LOGON_USERNAME at the start of the session.
PROFILE_NAME	–	–	VARCHAR (60)	Profile name for the current session.
SER_NAME	–	–	VARCHAR (32)	Server class name for the current session.
NODE_NAME	–	–	CHAR (8)	Node name of the reported process.
CPU_PIN	–	–	CHAR (7)	CPU and program identification number for the current session.
START_PRIORITY	–	–	INT (2) UNSIGNED	Priority used for execution.
ACC_LEVEL	–	–	VARCHAR (60)	Resource usage sample frequency: SESSION Summarize session SQL_STATEMENT Record each statement
IN_MESSAGES	–	–	INT (8)	Number of message reads done by the NonStop ODBC Server.
IN_MESSAGE_B	–	–	INT (8)	Number of bytes in the message reads done by the NonStop ODBC server.
OUT_MESSAGES	–	–	INT (8)	Number of message writes done by the NonStop ODBC server.
OUT_MESSAGE_B	–	–	INT (8)	Number of bytes in the message writes done by the NonStop ODBC server.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

Table 8-23. Description of ZNUMTRX (page 3 of 6)

Column Name	Primary Key	Unique Index	Data Type	Description
SCS_EXECUTION_ TIME	–	–	INT (8)	Time in microseconds that the SCS spends managing the session between the accounting start time and end time.
NSODBC_ EXECUTION_TIME	–	–	INT (8)	Amount of NonStop ODBC server execution time, in microseconds, spent executing in the server process, excluding NSSQL_EXECUTION_TIME. Includes: ● The CPU time used by the application code for message decoding and encoding, SQL management, data conversion, cache management, statement parsing and translation, and result set manipulation ● The CPU time used in direct invocation of Guardian System Library calls by the ODBC Server process, NOS.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

Table 8-23. Description of ZNUMTRX (page 4 of 6)

Column Name	Primary Key	Unique Index	Data Type	Description
NSSQL_EXECUTION_ TIME	–	–	INT (8)	Amount of NonStop SQL/MP execution time, in microseconds, between the accounting start time and end time. This value can be greater than NSODBC_ELAPSED_TIME due to parallelism in the processing. Includes: ● Total CPU time used by the SQL Executor and all Guardian System Library calls made by the SQL Executor ● Total CPU time used by all supporting ESP processes ● Total CPU time used by all supporting SORTPROG processes
NSODBC_ELAPSED_ TIME	–	–	INT (8)	Time in microseconds elapsed under NonStop ODBC server process control between the accounting start time and end time. It reflects cumulative wall clock time as measured each time control is given to the NonStop ODBC Server, NOS, until it responds with a message. This includes any wait time between multipacket return streams.
NSSQL_ELAPSED_ TIME	–	–	INT (8)	Time in microseconds elapsed while under NonStop SQL/MP executor control between the accounting start time and end time. It reflects the cumulative wall clock time as measured each time control is given to the SQL master executor until it releases back to the application.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

Table 8-23. Description of ZNUMTRX (page 5 of 6)

Column Name	Primary Key	Unique Index	Data Type	Description
STMT_TYPE	–	–	CHAR (1)	Type of statement being processed: S SELECT D DELETE U UPDATE I INSERT L DDL C DCL O Other
STMT_STATUS	–	–	CHAR (1)	Current status of the statement query: C Statement completed P Statement prepared X Statement cancelled by user Q NonStop SQL error occurred
STMT_ORIGIN	–	–	CHAR (1)	Flag indicating whether a statement is from a client or internally generated: U User generated I Internally generated
ESTIMATED_COST	–	–	INT (8)	Estimated cost of the NonStop SQL/MP PREPARE statement.
STMT_CACHED	–	–	CHAR (1)	Indicates whether the statement was already in cache: N Not cached Y Cached for the first time R Reused from cache
RECORDS_ ACCESSED	–	–	INT (8)	Number of records used generating the intermediate and final result sets.
RECORDS_USED	–	–	INT (8)	Number of records used generating the intermediate and final result sets.
DISC_READS	–	–	INT (8)	Number of physical disk reads performed.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

Table 8-23. Description of ZNUMTRX (page 6 of 6)

Column Name	Primary Key	Unique Index	Data Type	Description
MESSAGES	–	–	INT (8)	Number of messages sent for this OPEN, including the OPEN message.
MESSAGE_BYTES	–	–	INT (8)	Number of message bytes sent and received for this file OPEN.
LOCK_WAITS	–	–	INT (8)	Number of times the executor waited for a lock request to complete.
LOCK_ ESCALATIONS	–	–	INT (8)	Number of times a lock was escalated to a file-level lock.
LAST_GOV_ ATTRIBUTE	–	–	VARCHAR (60)	Indicator that describes whether this entry was generated when a governing action was triggered or by a change of statement status. An empty string indicates it was by a change of statement status. If not empty, the column contains the reason an accounting record is created.
LAST_LIMIT_VALUE	–	–	INT (8)	Limit value at accounting entry time.
LAST_GOV_ACTION	–	–	VARCHAR (60)	Action taken when the limit of the governing attribute was exceeded.
STMT_TEXT	–	–	VARCHAR (3000)	The SQL statement as prepared.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

ZNUOBJ (For Logical Object Names Mapping)

The table ZNUOBJ maps ODBC or SQL Server table and view names to NonStop SQL/MP object names.

ZNUOBJ resides in both the subvolume of the system catalog and a customized user catalog.

Table 8-24. Description of ZNUOBJ (page 1 of 2)

Column Name	Primary Key	Unique Index	Data Type	Description
T_UID	x	–	INT (2) UNSIGNED	Numeric user ID of the owner.
T_OBJNAME	x	x	VARCHAR (60)	Fully qualified ODBC or SQL Server object name in uppercase letters.
T_UNAME	–	x	VARCHAR (60)	Logical username in uppercase letters.
T_OBJTYPE	–	–	CHAR (2)	Code indicating the object type: N NonStop SQL/MP catalog table or NonStop ODBC Server mapping table S System table view T Temporary table U User table V View
T_OBJID	–	x	INT (4)	Object ID used internally by the NonStop ODBC Server to satisfy requirements of TSQL users.
N_OBJNAME	–	x	CHAR (34)	Fully qualified NonStop SQL/MP object name in uppercase letters.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

Table 8-24. Description of ZNUOBJ (page 2 of 2)

Column Name	Primary Key	Unique Index	Data Type	Description
N_OBJTYPE	–	–	CHAR (2)	NonStop SQL/MP object type: TA Table VI View
N_ORIGIN	–	–	CHAR (1)	Code indicating the type of statement that created the object: T Transact-SQL or CORE SQL N NonStop SQL/MP
N_SESSIONID	–	–	INT (8)	Session ID used internally by the NonStop ODBC Server.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

The index ZNUOBJI1 ensures that the user ID and server name combination is unique. The index is created on the T_UNAME and T_OBJNAME columns of ZNUOBJ. ZNUOBJI1 resides in both the subvolume of the system catalog and a customized user catalog.

The index ZNUOBJI2 ensures that any NonStop SQL/MP object name in ZNUOBJ is unique. The index is created on the N_OBJNAME column of ZNUOBJ. ZNUOBJI2 resides in both the subvolume of the system catalog and a customized user catalog.

The index ZNUOBJI3 is used internally by the NonStop ODBC Server. The index is created on the T_OBJID column of ZNUOBJ. ZNUOBJI3 resides in both the subvolume of the system catalog and a customized user catalog.

The resource accounting log files ZNUMTRX, ZNUQST, and the default tracelog ZNUTRA, are considered user-tables and therefore have a T_OBJTYPE of “U”.

ZNUPCOL (For Stored Procedure Parameters and Results)

The table ZNUPCOL contains information about the parameters and result sets of stored procedures. The table contains one row for each input parameter, each input/output parameter, and, optionally, each column in the result set. ZNUPCOL also provides support for the ODBC catalog function SQLProcedureColumns.

ZNUPCOL resides in both the subvolume of the system catalog and a customized user catalog.

See also ZNUPROC (For Stored Procedure Attributes) on page 8-55.

Table 8-25. Description of ZNUPCOL (page 1 of 2)

Column Name	Primary Key	Unique Index	Data Type	Description
T_OBJID	x	x	INT (4)	Object ID used internally by the NonStop ODBC Server to satisfy requirements of TSQL users.
T_NAME	x	–	VARCHAR (60)	Column/parameter name.
T_SEQ	–	x	INT (2)	Sequence of the column.
T_TYPE	–	–	INT (2)	Column type indicator: 1 SQL_PARAM_INPUT. 2 SQL_PARAM_INPUT_ OUTPUT.
N_DTYPE	–	–	CHAR (18)	NonStop SQL/MP data type name.
N_FSTYPE	–	–	INT (2)	File-system SQL data type.
N_PREC	–	–	INT (2)	Precision of the column data.
N_LEN	–	–	INT (2)	Length of the column data.
N_SCALE	–	–	INT (2)	Scale factor of the column data.
N_RADIX	–	–	INT (2)	Radix of the column data (2 or 10).
T_NULLABLE	–	–	INT (2)	Transact-SQL nullable indicator: 0 Not nullable 1 Nullable
N_NULLABLE	–	–	CHAR (1)	Nullable indicator: Y Column is nullable. N Column is not nullable.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

Table 8-25. Description of ZNUPCOL (page 2 of 2)

Column Name	Primary Key	Unique Index	Data Type	Description
N_ DATETIMESTARTFIELD	–	–	INT (2)	Most significant field to be used in stored DATETIME values: 1 Year 2 Month 3 Day 4 Hour 5 Minute 6 Second 7 Fraction 0 Not a DATETIME value
N_ DATETIMEENDFIELD	–	–	INT (2)	Least significant field (1–7) to be used in stored DATETIME values.
N_DEFAULTCLASS	–	–	CHAR (1)	Default value class: N No default U Default *value* D Default null
N_DEFAULTVALUE	–	–	VARCHAR (254)	The default value.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

The index ZNUPCLI1 ensures that any procedure parameter definition in ZNUPCOL is unique for a given procedure. The index is created on the T_OBJID and T_SEQ columns of ZNUPCOL.

ZNUPCLI1 resides in both the subvolume of the system catalog and a customized user catalog.

ZNUPROC (For Stored Procedure Attributes)

The table ZNUPROC contains the combined attributes required to support execution of stored procedures for both CORE SQL users and Transact-SQL users. The table contains one row for each stored procedure supported by the NonStop ODBC Server. ZNUPROC also provides support for the ODBC catalog function SQLProcedures and the SQL Server catalog SYSOBJECTS.

ZNUPROC resides in both the subvolume of the system catalog and a customized user catalog.

See also ZNUPCOL (For Stored Procedure Parameters and Results) on page 8-53.

Table 8-26. Description of ZNUPROC (page 1 of 2)

Column Name	Primary Key	Unique Index	Data Type	Description
T_OBJID	x	–	INT (4)	Object ID used internally by the NonStop ODBC Server to satisfy requirements of TSQL users.
T_PROCNUM	–	–	INT (2)	Procedure number in a procedure group.
N_PATHNAME	–	–	CHAR (15)	Pathway Monitor name (`[\`*`node`*`.]$`*`process`*).
N_SERVERCLASS	–	–	CHAR (15)	Pathway server class name.
N_SERVICE	–	–	VARCHAR (60)	Pathway service name (takes the procedure name in ADD PROCEDURE).
T_NUMIPARMS	–	–	INT (2)	Number of input parameters.
T_NUMOPARMS	–	–	INT (2)	Number of output parameters.
T_NUMRESULTS	–	–	INT (2)	Number of result sets.
T_RETURNSTAT	–	–	INT (2)	Is status returned? 0 No 1 Yes

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

Table 8-26. Description of ZNUPROC (page 2 of 2)

Column Name	Primary Key	Unique Index	Data Type	Description
N_CREATETIME	–	–	TIMESTAMP	Date and time the procedure was added.
N_MAX_BUFFER_LEN	–	–	INT (2)	Maximum length of the IPC buffer.
REMARKS	–	–	VARCHAR (254)	Explanatory comments. Has a static value of zero length; comments can be entered through SQLCI, but not through the API.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

N_PATHNAME identifies the Pathway system (\TESS.$PMON, for example). N_SERVERCLASS names the Pathway server class to which the ServerClass_Send call is sent; it is an SQL identifier. N_SERVICE allows the stored procedure implementation in the server class to be named. It generally corresponds to a Pathmaker service name, beginning with a letter and containing, letters, numerals, and, optionally, hyphens (-).

ZNUQST (For Query Status Data)

The table ZNUQST is the simple file name used for logging current activity. It is the log table name in the ZNSPROF record of the current user profile.

The NonStop ODBC server process enters the values on this table.

The ZNUQST table is created as an unaudited table so that concurrency is maximized among multiple users. Log entry rows only append to the table during the time the log is active. For information on activating the log, see the *HP NonStop ODBC Server Installation and Management Manual.*

ZNUQST resides in both the subvolume of the system catalog and a customized user catalog.

Table 8-27. Description of ZNUQST (page 1 of 2)

Column Name	Primary Key	Unique Index	Data Type	Description
CLIENT_ID	–	–	VARCHAR (60)	A unique identifier used to link this accounting status entry to a client session.
SESSION _ID	–	–	INT (4)	A unique identifier generated by the NonStop ODBC Server.
START_TIME	–	–	TIMESTAMP	Time (in milliseconds) the query started the current status.
LOGON_USERNAME	–	–	VARCHAR (60)	Value provided in the logon record by the user. It is used to determine the current user session environment settings.
G_USERNAME	–	–	CHAR (17)	Guardian username determined from the LOGON_USERNAME at the start of the session.
PROFILE_NAME	–	–	VARCHAR (60)	Profile name determined for the current session.
SER_NAME	–	–	VARCHAR (32)	Server class name for the current session.
NODE_NAME	–	–	CHAR (8)	Node name for reported statistic.
CPU_PIN	–	–	CHAR (7)	CPU and program identification numbers for the current session.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

Table 8-27. Description of ZNUQST (page 2 of 2)

Column Name	Primary Key	Unique Index	Data Type	Description
START_PRIORITY	–	–	INT (2) UNSIGNED	Priority at which execution is being performed.
TRANSACTION_ID	–	–	CHAR (22)	Current transaction ID.
STMT_TYPE	–	–	CHAR (1)	Type of statement being processed: S SELECT D DELETE U UPDATE I INSERT L DDL C DCL O Other
STMT_ORIGIN	–	–	CHAR (1)	Client generated or internally generated? C Client generated I Internally generated
ESTIMATED_COST	–	–	INT (8)	Estimated cost of the NonStop SQL/MP PREPARE statement.
STMT_CACHED	–	–	CHAR (1)	Was the statement already in cache? N Not cached Y Cached for the first time R Reused from cache
STMT_TEXT	–	–	VARCHAR (3000)	The prepared SQL statement.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

ZNUTRA (For Trace Records Logging)

The name ZNUTRA is a template file name for the table used for logging trace records. The actual name of the table is one chosen by the user. It is the log table name found in the ZNSTRA record of the current user profile.

ZNUTRA resides in both the subvolume of the system catalog and a customized user catalog.

The CREATE TRA_LOG statement creates the log table and the DROP TRA_LOG statement drops it. The log is an unaudited table. Log entry rows are appended to it only when the log is active. To make the log active, set TRA_MODE_ON to “Y” in the ZNSPROF table.

Table 8-28. Description of ZNUTRA

Column Name	Primary Key	Unique Index	Data Type	Description
ID	–	–	INT (2)	Value used to link this trace entry to a session.
DEBUG_FLAG	–	–	CHAR (10)	Trace state for which trace text is taken: LOGON TIME IN_STREAM OUT_STREAM CACHE ERROR INFO NSSQL PROFILE TRACE
DEBUG_STRING	–	–	VARCHAR (60)	Trace text.

x Indicates the column is part of the primary key or is a unique index
– Indicates the column is not part of the primary key or is not a unique index

System Table Mapping

The NonStop ODBC Server maintains tables and views that correspond to the SQL Server system tables. The prefix “SYS” indicates a view specific to TSQL that is provided by the NonStop ODBC Server and corresponds to a TSQL system catalog table or view.

When you specify a system table name, the NonStop ODBC Server searches the object mapping table, ZNUOBJ, and locates the view name that corresponds to the system table.

The subsections that follow list the view definitions for TSQL compatibility.

SQLColumns Procedure (ZVUOCOL)

The view ZVUOCOL satisfies the SQLColumns query for ODBC access. It translates the column information in COLUMNS into the NonStop ODBC Server column description form.

ZVUOCOL resides in both the subvolume of the system catalog and a customized user catalog.

See also ZNUDT (For Data Types Mapping) on page 8-42 and SQLColumns on page 7-122.

Table 8-29. Description of ZVUOCOL

Column Name	Column Name in ZNUDT, ZNUOBJ, or COLUMNS	Description
T_UNAME	ZNUOBJ.T_UNAME	Fully qualified owner name in uppercase letters.
T_OBJNAME	ZNUOBJ.T_OBJNAME	Fully qualified ODBC or SQL Server object name in uppercase letters.
COLNAME	COLUMNS.COLNAME	Object ID used internally by the NonStop ODBC Server.
T_ODBCTYPE	ZNUDT.T_ODBCTYPE	Column/parameter name.
T_NAME	ZNUDT.T_NAME	Number (sequence) of this column.
PRECISION	COLUMNS.PRECISION	Degree of precision of this column (type-dependent; numeric or datetime only).
LENGTH	COLUMNS.COLSIZE	Length of the column data.
SCALE	COLUMNS.SCALE	Scale factor of this column (type-dependent; numeric or datetime only).
T_NULLABLE	ZNUDT.T_NULLABLE	Indicate whether the ODBC or SQL Server data type allows null values.

SQLProcedureColumns Procedure (ZVUPCOL)

The view ZVUPCOL satisfies the SQLProcedureColumns query for ODBC access. It translates the parameter information into the NonStop ODBC Server parameter description form.

ZVUPCOL resides in both the subvolume of the system catalog and a customized user catalog.

See also the following:

- ZNUDT (For Data Types Mapping) on page 8-42
- ZNUOBJ (For Logical Object Names Mapping) on page 8-51,
- ZNUPCOL (For Stored Procedure Parameters and Results) on page 8-53
- SQLColumns on page 7-122.

Table 8-30. Description of ZVUPCOL

Column Name	Column Name in ZNUDT, ZNUOBJ, or ZNUPCOL	Description
T_UNAME	ZNUOBJ.T_UNAME	Fully qualified owner name in uppercase letters.
T_OBJNAME	ZNUOBJ.T_OBJNAME	Fully qualified ODBC or SQL Server object name in uppercase letters.
COLUMN_NAME	ZNUPCOL.T_NAME	Column/parameter name.
COLUMN_TYPE	ZNUPCOL.T_TYPE	Column type indicator: 1 SQL_PARAM_INPUT 2 SQL_PARAM_INPUT_OUTPUT
DATA_TYPE	ZNUDT.T_ODBCTYPE	ODBC user-defined data type code.
TYPE_NAME	ZNUDT.T_NAME	ODBC or SQL Server data type name, such as INT, FLOAT, or DATETIME.
PRECISION	ZNUPCOL.N_PREC	Degree of precision of this column (type-dependent; numeric or datetime only).
LENGTH	ZNUPCOL.N_LEN	Length of the column data.
SCALE	ZNUPCOL.N_SCALE	Scale factor of this column (type-dependent; numeric datetime only).
RADIX	ZNUPCOL.N_RADIX	Radix of the column data (2 or 10).
T_NULLABLE	ZNUPCOL.T_NULLABLE	Indicates whether the ODBC or SQL Server data type allows null values.

SYSCOLUMNS Catalog View (ZVUCOL)

The catalog view SYSCOLUMNS lists the columns of tables and views in a NonStop SQL/MP catalog.

SYSCOLUMNS is represented by a join view over COLUMNS, TABLES, ZNSOBJ (the object mapping table), and ZNUDT (the data types mapping table). The view and mapping tables reside on each customized subvolume.

Sequence of rows is ascending by ID.

SYSCOLUMNS does not contain column definitions for the input, output, or result columns of the stored procedures in ZNUPROC.

Table 8-31. Description of SYSCOLUMNS

Column Name	**Column Name in ZNUIX, COLUMNS, or SYSTYPES**	**Description**
ID	ZNUIX.T_TABLEID	Object ID used internally by the NonStop ODBC Server.
NUMBER	None	No corresponding column name. Assigned 0.
COLID	COLUMNS.COLNUMBER	A number indicating the position of the column in the row of the table (the first column is 0).
STATUS	COLUMNS.NULLABLE	A number indicating whether the ODBC or SQL Server data type allows null values.
TYPE	COLUMNS.DATATYPE	ODBC or SQL Server data type code.
LENGTH	COLUMNS.COLSIZE	Byte length of the data in the column.
OFFSET	COLUMNS.OFFSET	Reserved for internal use by HP software.
USERTYPE	SYSTYPES.USERTYPE.	ODBC or SQL Server user-defined data type code.
CDEFAULT	None	No corresponding column name. Assigned –1.
DOMAIN	None	No corresponding column name. Assigned –1.
NAME	COLUMNS.COLNAME. “#”s and “$”s are represented as “_”s. Leading “_”s are represented as “U”s.	Column name, which is a NonStop SQL/MP identifier.
PRINTFMT	None	No corresponding column name. Assigned “ ” (blank).

SYSDATABASES Catalog View (ZVSDB)

The catalog view SYSDATABASES represents the customized NonStop SQL/MP catalogs on a node. It is supported by a view over a mapping table. It is updated when a NonStop ODBC Server user issues a CREATE or DROP DATABASE statement or the equivalent statement for USERCAT INSTALL or USERCAT DEINSTALL.

SYSDATABASES exists only with the NonStop SQL/MP system catalog that corresponds to the MASTER database. The MASTER database is assigned to the customized SQL system catalog for the given node.

Table 8-32. Description of SYSDATABASES

Column Name	Column Name in ZNSDB or CATALOGS	Description
NAME	ZNSDB.T_DBNAME	Fully qualified ODBC or SQL Server database name in uppercase letters.
DBID	ZNSDB.T_DBID	Internal ID used by the NonStop ODBC Server to satisfy requirements of TSQL users.
SUID	ZNSDB.T_CREATOR	ODBC or SQL Server user ID of the database creator. If the database is created in NonStop SQL/MP and then customized, the creator is the user ID of the user who customizes the catalog.
MODE	None	No corresponding column name. Assigned –1.
STATUS	None	No corresponding column name. Assigned –1.
VERSION	ZNSDB.T_VERSION	NonStop ODBC Server version number.
CRDATE	CATALOGS.CREATETIME	Date and time the catalog was created or customized.

SYSINDEXES Catalog View (ZVUIX)

The catalog view SYSINDEXES is defined for TSQL mode only. It lists the indexes in a NonStop SQL/MP catalog that are created by NonStop ODBC Server users. Indexes created by HP server users that are on tables in other catalogs are not visible through SYSINDEXES.

SYSINDEXES is represented by a view over INDEXES and a mapping table. The view and mapping tables reside on each customized subvolume.

Sequence of rows is ascending by ID and NAME.

SYSINDEXES is updated when a NonStop ODBC Server user issues a CREATE or DROP INDEX statement.

The SQL Server catalog table SYSKEYS is not supported with a specific view, but is instead satisfied with an SQL query generated by the driver directly against the mapping and catalog tables.

Table 8-33. Description of SYSINDEXES (page 1 of 2)

Column Name	Column Name in ZNUIX or INDEXES	Description
NAME	ZNUIX.T_INAME	the fully qualified ODBC or SQL Server index name in uppercase letters.
ID	ZNUIX.T_TABLEID	A random number generated by the system.
INDID	INDEXES.KEYTAG + 1	Primary key indicator.
DPAGES	None	No corresponding column name. Assigned –1.
RESERVED	None	No corresponding column name. Assigned –1.
USED	None	No corresponding column name. Assigned –1.
ROWS	None	No corresponding column name. Assigned –1.
FIRST	None	No corresponding column name. Assigned –1.
ROOT	None	No corresponding column name. Assigned –1.
DISTRIBUTION	None	No corresponding column name. Assigned –1.
USAGECNT	None	No corresponding column name. Assigned –1.
SEGMENT	None	No corresponding column name. Assigned –1.

Table 8-33. Description of SYSINDEXES (page 2 of 2)

Column Name	Column Name in ZNUIX or INDEXES	Description
STATUS	INDEXES.UNIQUEVALUE	A number used internally by the NonStop ODBC Server to record whether an index is unique.
ROWPAGE	None	No corresponding column name. Assigned –1.
MINLEN	INDEXES.ROWSIZE	Length of the packed index record.
MAXLEN	INDEXES.ROWSIZE	No corresponding column name. Assigned –1.
MAXIROW	None	No corresponding column name. Assigned –1.
KEYCNT	INDEXES.COLCOUNT	Number of columns in the index.
KEYS1	None	No corresponding column name. Assigned “ ” (blank).
KEYS2	None	No corresponding column name. Assigned “ ” (blank).

SYSMESSAGES Catalog View (ZVSMSG)

The catalog view SYSMESSAGES contains error codes and error text.

SYSMESSAGES is represented as a view over a mapping table. The view and mapping table reside only in the subvolume of the system catalog.

Sequence of rows is ascending by ERROR.

SYSMESSAGES is created when the NonStop ODBC Server is installed. It is provided for DLIB compatibility but is not used by the NonStop ODBC server or NOSUTIL program for generating error messages.

SYSMESSAGES is retained for backward compliance with the old gateway, SQLSG, and DLIB programs.

See also ZNSMSG (For Error Messages) on page 8-18.

Table 8-34. Description of SYSMESSAGES

Column Name	Attribute Name or Column Name in ZNSMSG	Description
ERROR	T_ERROR_CODE	ODBC or SQL Server error code that corresponds to this message. If there is no corresponding ODBC or SQL Server error code, the code is 18001.
SEVERITY	ZNSMSG.T_CLASS	THE ODBC or SQL Server error severity level: 10 Warning 11–16 User-generated 17–18 Nonfatal error 19+ Fatal error
DLEVEL	ZNSMSG.T_STATE	ODBC or SQL Server error detail level. Assigned –1.
DESCRIPTION	ZNSMSG.N_ERROR_TEXT	Error text. This is usually TSQL, but in some cases, it is NonStop SQL/MP.

SYSOBJECTS Catalog Table (ZVUOBJ)

The catalog view SYSOBJECTS lists the tables and views in a NonStop SQL/MP Catalog.

It includes user and SQL Server system tables but does not include stored procedures.

SYSOBJECTS is represented by a join view over TABLES and mapping tables. The view and mapping tables reside on each customized subvolume.

Sequence of rows is ascending by UID and NAME.

Each partition of a partitioned table is mapped to a distinct qualified TSQL table name without regard to the catalogs in which the partitions are registered. Therefore, you can access the same underlying NonStop SQL/MP table through multiple TSQL tables.

Table 8-35 lists and describes the columns in SYSOBJECTS. NonStop SQL/MP Catalog tables appear in the description as user tables.

Table 8-35. Description of SYSOBJECTS (page 1 of 2)

Column Name	Column Name in ZNUOBJ, ZNSUS, or TABLES	Description
NAME	ZNUOBJ.TABLENAME	Fully qualified ODBC or SQL Server object name in uppercase letters.
ID	None	Object ID used internally by the NonStop ODBC Server. Value is unique.
UID	ZNSUS.NOS_UID	User ID used internally by the NonStop ODBC Server.
TYPE	TABLES.TABLETYPE or “P” if the NAME is a stored procedure.	Code indicating the object type: N NonStop SQL/MP catalog table or NonStop ODBC Server mapping table S System table view T Temporary table U User table V View
USERSTAT	None	No corresponding column name. Assigned –1.
SYSSTAT	None	No corresponding column name. Assigned –1.
INDEXDEL	None	No corresponding column name. Assigned –1.
SCHEMA	None	No corresponding column name. Assigned –1.

Table 8-35. Description of SYSOBJECTS (page 2 of 2)

Column Name	Column Name in ZNUOBJ, ZNSUS, or TABLES	Description
REFDATE	None	No corresponding column name. Assigned “ ” (blank).
CRDATE	TABLES.CREATETIME	Date and time the object was created.
EXPDATE	None	No corresponding column name. Assigned “ ” (blank).
DELTRIG	None	No corresponding column name. Assigned –1.
INSTRIG	None	No corresponding column name. Assigned –1.
UPDTRIG	None	No corresponding column name. Assigned –1.
SELTRIG	None	No corresponding column name. Assigned –1.
CATEGORY	None	No corresponding column name. Assigned –1
CACHE	None	No corresponding column name. Assigned –1.

When a user creates a view, the view definition text is not visible to a TSQL user unless the mapped VIEWS and mapping tables are queried.

SYSPROTECTS Catalog View (ZVUPROT)

The catalog view SYSPROTECTS is created when the NonStop ODBC Server is installed. It is provided for DBLIB users only. It reports all rights against all objects registered in the local ZNOBJ objects table.

SYSPROTECTS is represented by a join view over ZNSPROT and ZNUOBJ. The view and mapping table reside on each customized subvolume.

Table 8-36. Description of SYSPROTECTS

Column Name	Column Name in ZNSUS	Description
ID	None	No corresponding column name. Assigned –1.
UID	ZNSUS.NOS_UID	User ID used internally by the NonStop ODBC Server.
ACTION	None	An integer corresponding to a Transact-SQL statement: 193 SELECT 195 INSERT 196 DELETE 197 UPDATE 224 EXECUTE
PROTECTTYPE	None	An integer, 205, corresponding to GRANT permission. REVOKE (206) is not supported. When using the NonStop ODBC Server, all users are granted permission for all statements.
COLUMNS	None	Column list to which the permission applies.

SYSTYPES Catalog View (ZVUDT)

The catalog view SYSTYPES contains each system-supplied data type. The NonStop ODBC Server does not support user-defined data types.

SYSTYPES is represented by a view over a data type mapping table. The mapping table resides on each customized subvolume. It contains the same constant values in every catalog.

Table 8-37. Description of SYSTYPES

Column Name	Column Name in ZNUDT	Description
UID	None	No corresponding column name. Assigned 1.
USERTYPE	ZNUDT.T_USERTYPE	ODBC or SQL Server user-defined data type code.
VARIABLE	None	No corresponding column name. Assigned –1.
ALLOWNULLS	ZNUDT.T_NULLABLE	Indicator regarding the use of nulls. The value is based on the value in the TYPE column: 0 TYPE is BIT, SYSNAME, or TINYINT 1 TYPE is any other value
TYPE	ZNUDT.T_DTYPE	ODBC or SQL Server data type code.
LENGTH	ZNUDT.N_PRECHI	High end of the precision range. (0 if not applicable).
TDEFAULT	None	No corresponding column name. Assigned 0.
DOMAIN	None	No corresponding column name. Assigned 0.
NAME	ZNUDT.T_NAME	ODBC or SQL Server data type name, such as INT, FLOAT, or DATETIME.
PRINTFMT	None	No corresponding column name. Assigned “ ” (blank).

SYSUSERS Catalog View (ZVUUS)

The catalog view SYSUSERS contains usernames used only to qualify object names. SYSUSERS is represented by a view over the mapping table ZNSUS.

The view and mapping table reside on each customized subvolume.

SYSUSERS is created when a catalog is customized. You can add users with the NOSUTIL ADD USER statement. For information on this statement, see Running the Catalog Utility Statements on page 7-13.

SYSUSERS is defined for each database, but it has the same values in all user databases for a specific node. SQL Server, on the other hand, has distinct user lists for each user database, and the special username DBO is reserved for the user who creates the database.

Table 8-38. Description of SYSUSERS

Column Name	Column Name in ZNSUS	Description
SUID	None	No corresponding column name. Assigned –1.
UID	ZNSUS.NOS_UID	User ID used internally by the NonStop ODBC Server.
GID	None	No corresponding column name. Assigned –1.
NAME	ZNSUS./NOS_USERNAME	Logical username.
ENVIRON	None	No corresponding column name. Assigned " " (blank).

SYSUSERS is loaded at the time of catalog customization with a row containing the following values:

Column	Value
SUID	–1
UID	1
GID	–1
NAME	DBO
ENVIRON	" " (blank)

Actions That Affect the Mapping Tables

When you use the NonStop ODBC Server, some of the actions you take or statements you execute cause the NonStop ODBC Server to create, drop, query, or modify the mapping tables.

Table 8-39. Actions That Cause the NonStop ODBC Server to Create or Drop Mapping Tables

Action	Mapping Tables Created or Dropped	Description
Install the NonStop ODBC Server	Mapping tables in the system catalog	The tables are created.
Deinstall the NonStop ODBC Server	Mapping tables in the system catalog	The tables are dropped.
Create a database using the NonStop ODBC Server	Mapping tables in a user catalog	The tables are created.
Customize a NonStop SQL/MP catalog	Mapping tables in a user catalog	The tables are created.
Drop a database using the NonStop ODBC Server	Mapping tables in a user catalog	The tables are dropped.
Decustomize a NonStop SQL/MP catalog	Mapping tables in a user catalog	The tables are dropped.
Refresh a customized NonStop SQL/MP catalog	Mapping tables in a user catalog	Individual tables are re-created.

Table 8-40 describes actions that cause the NonStop ODBC Server to query the mapping tables.

Table 8-40. Actions That Cause the NonStop ODBC Server to Query the Mapping Tables

Action	Mapping Tables Queried	Description
Select from a table or view	ZNUOBJ in the subvolume of the catalog in which the table or view is registered	NonStop ODBC server searches the object mapping table, ZNUOBJ, and finds the NonStop SQL/MP name corresponding to the specified table or view name.
Modify, delete, or add data	ZNUOBJ in the subvolume of the catalog in which the table or view is registered	NonStop ODBC server searches the object mapping table, ZNUOBJ, and finds the NonStop SQL/MP name corresponding to the specified table or view name.
Select from a system table	ZNUOBJ in the subvolume of the current catalog and the view corresponding to the system table	NonStop ODBC server searches the object mapping table, ZNUOBJ, and finds the view name that corresponds to the specified system table.
Change the current database context (with the USE statement)	ZNSDB in the subvolume of the system catalog	NonStop ODBC server searches the database mapping table, ZNSDB, and finds the NonStop SQL/MP catalog name that corresponds to the specified database name.

Table 8-41. Actions That Cause the NonStop ODBC Server to Modify the Mapping Tables

Action	Mapping Tables Affected	Description
Create a database using the NonStop ODBC Server or customize a NonStop SQL/MP catalog	ZNSDB in the subvolume of the system catalog	NonStop ODBC server adds a row for the database in ZNSDB.
Drop a database using the NonStop ODBC Server or decustomize a NonStop SQL/MP catalog	ZNUOBJ and ZNSDB in the subvolume of the system catalog	NonStop ODBC server deletes the row for the database from both ZNUOBJ and ZNSDB.
Create an object using the NonStop ODBC Server	ZNUOBJ—or ZNUIX, if the object is an index—in the subvolume of the current catalog	NonStop ODBC server creates a row for the object.
Drop an object using the NonStop ODBC Server	ZNUOBJ—or ZNUIX, if the object is an index—in the subvolume of the current catalog	NonStop ODBC server drops the row for the object.

9 UNIX Notes

This section contains the environment-specific information on using DB-LIBRARY in the UNIX environment.

The Microsoft SQL Server *Programmer's Reference* describes DB-LIBRARY and lists DB-LIBRARY commands and routines for DOS-based and OS/2-based SQL Server. This section contains more information for UNIX users. It covers the following topics:

- Building an executable
- DB-LIBRARY functions and macros

Note. If you are using SQL Server documentation from Sybase, you can use the Sybase manuals—you do not need the Microsoft manuals or the information in this section.

Building an Executable

The syntax of the command to compile and link a DB-LIBRARY application is:

```
cc program.c -I$SYBASE/include $SYBASE/lib/libsybdb.a -lm -o
program
```

The preceding command assumes that the DB-LIBRARY include files and the Sybase library *libsybdb.a* reside in the directories in which they were initially installed. If these files have been moved to standard UNIX library and include directories, the command is simpler:

```
cc program.c -lsybdb -lm -o program
```

Note that DB-LIBRARY applications must link with the math library, *libm.a*, which is invoked with the *-lm* flag.

DB-LIBRARY Functions and Macros

The Microsoft SQL Server *Programmer's Reference* lists DB-LIBRARY functions and macros for DOS-based and OS/2-based SQL Server. UNIX users can use the additional functions and macros described in Table 9-1.

Table 9-1. DB-LIBRARY Functions and Macros for the UNIX Environment

Function or Macro	Description
DBIORDESC	Provides program access to the UNIX file descriptor used by a DBPROCESS to read data coming from SQL Server.
DBIOWDESC	Provides program access to the UNIX file descriptor used by a DBPROCESS to write data to SQL Server.
DBRBUF	Determines whether the DB-LIBRARY network buffer contains any unread bytes.

DBIORDESC

DBIORDESC gives access to the UNIX file descriptor used by a DBPROCESS to read data coming from SQL Server. DBIORDESC has the following syntax:

```
int DBIORDESC (dbproc)

DBPROCESS *dbproc;
```

`dbproc`

a pointer to the DBPROCESS structure that provides the connection for a particular front-end process or SQL Server process. It contains all the information that DB-LIBRARY uses to manage communications and data between the front end and SQL Server.

Comments

The following comments apply to DBIORDESC:

- This routine provides a way for an application to respond effectively to multiple input streams. Depending on the nature of your application, the time between a request for information from SQL Server (usually made using a call to *`dbsqlsend()`*) and SQL Server's response (read by calling *`dbsqlok()`*, *`dbresults()`*, or *`dbnextrow()`*) can be significant. You can use this time to service other parts of your application.

  The *`DBIORDESC()`* routine provides a way to obtain the I/O descriptor that a DBPROCESS uses to read the data stream from SQL Server. This information can then be used with various operating system facilities (such as the UNIX *`select()`* call) to allow the application to respond effectively to multiple input streams.

- The file descriptor returned by this routine can be used only with operating system facilities that do not read data from the incoming data stream. If data is read from this stream by any means other than through a DB-LIBRARY routine, communications between the front end and SQL Server become scrambled.

- An application can use the DB-LIBRARY *`DBRBUF()`* routine, in addition to the UNIX *`select()`* function, to help determine whether any more data from SQL Server is available for reading.
- A companion routine, *`DBIOWDESC()`*, provides access to the file descriptor used to write data to SQL Server.

Returns

DBIORDESC returns an integer file descriptor used by the specified DBPROCESS to read data coming from SQL Server.

See Also

For further information, see DBIOWDESC or DBRBUF on page 9-4, or see *`dbcmd`*, *`dbnextrow`*, *`dbresults`*, *`dbsqlok`*, or *`dbsqlsend`* in the Microsoft SQL Server *Programmer's Reference*.

DBIOWDESC

DBIOWDESC provides program access to the UNIX file descriptor used by a DBPROCESS to write data to SQL Server. DBIOWDESC has the following syntax:

```
int DBIOWDESC(dbproc)

DBPROCESS *dbproc;
```

`dbproc`

a pointer to the DBPROCESS structure that provides the connection for a particular front-end process or SQL Server process. It contains all the information that DB-LIBRARY uses to manage communications and data between the front end and SQL Server.

Comments

The following comments apply to DBIOWDESC:

- This routine provides a way for an application to effectively use multiple input and output streams. Depending on the nature of your application, the time interval between the initiation of an attempt to write information to SQL Server (usually made using a call to *`dbsqlsend()`*) and the completion of that attempt can be significant. You can use this time to service other parts of your application.

  The *`DBIOWDESC()`* routine provides a way to obtain the I/O descriptor that a DBPROCESS uses to write the data stream to SQL Server. This information can be used with various operating system facilities (such as the UNIX *`select()`* function) to allow the application to effectively use multiple input and output streams.

- The file descriptor returned by this routine can be used only with operating system facilities that do not read data from the incoming data stream. If data is read from this stream by any means other than through a DB-LIBRARY routine, communications between the front end and SQL Server will become scrambled.
- A companion routine, *`DBIORDESC()`*, provides access to the file descriptor used to read data coming from SQL Server.

Returns

DBIOWDESC returns an integer file descriptor used by the specified DBPROCESS to write data to SQL Server.

See Also

For further information, see DBIORDESC on page 9-2 or see *`dbcmd`*, *`dbnextrow`*, *`dbresults`*, *`dbsqlok`*, or *`dbsqlsend`* in the Microsoft SQL Server *Programmer's Reference*.

DBRBUF

DBRBUF determines whether the DB-LIBRARY network buffer contains any unread bytes. DBRBUF has the following syntax:

```
DBBOOL DBRBUF (dbproc)

DBPROCESS *dbproc;
```

`dbproc`

a pointer to the DBPROCESS structure that provides the connection for a particular front-end process or SQL Server process. It contains all the information that DB-LIBRARY uses to manage communications and data between the front end and SQL Server.

Comments

The following comments apply to DBRBUF:

- This routine lets the application know if the DB-LIBRARY network buffer contains any bytes yet unread.
- *`DBRBUF()`* is ordinarily used in conjunction with *`dbsqlok()`* and *`DBIORDESC()`* to manage multiple input data streams. To manage these input streams efficiently, an application that uses *`dbsqlok()`* should check whether any bytes remain either in the network buffer (by calling *`DBRBUF()`*) or in the network itself (by calling the UNIX *`select()`*) before it calls *`dbresults()`*.

Returns

DBRBUF returns TRUE or FALSE:

TRUE	Bytes remain in the buffer
FALSE	No bytes remain in the buffer

See Also

For further information, see DBIORDESC on page 9-2 or see *`dbsqlok`*, or *`dbresults`* in the Microsoft SQL Server *Programmer's Reference*.

A Summary of Support for ODBC Features

Not all ODBC features are supported by the HP NonStop ODBC Server. This appendix summarizes the following features of ODBC and, for each feature, indicates whether the NonStop ODBC Server supports the feature:

- Aggregates
- Data types
- Expressions and operators
- Functions
- Identifiers
- Joins
- NULL values
- Search conditions
- Statements
- Stored procedures
- Wild-card characters

Detailed information on each of these topics can be found under the appropriate headings in Section 3, CORE SQL Language.

Aggregates

The NonStop ODBC Server supports all of the ODBC aggregate functions:

- AVG
- COUNT
- MAX
- MIN
- SUM

Data Types

The NonStop ODBC Server supports all ODBC data types, at both the Core level and the Extended level. When you create objects and data, however, the NonStop ODBC Server maps the data to a HP NonStop SQL/MP data type.

Table A-1 summarizes the ODBC data types and the corresponding NonStop SQL/MP data types.

Table A-1. Conversion of ODBC Data Types to NonStop SQL/MP Data Types

ODBC Data Type	ODBC Language Level	Corresponding NonStop SQL/MP Data Type
BIGINT	E	LARGEINT
BINARY	E	CHAR
BIT	E	SMALLINT
CHAR	C	CHAR
DATE	E	DATETIME year to day
DECIMAL	C	DECIMAL
DOUBLE PRECISION	C	DOUBLE PRECISION
FLOAT	C	FLOAT
INTEGER	C	INTEGER
NUMERIC	C	NUMERIC
LONG VARBINARY	E	VARCHAR
LONG VARCHAR	E	VARCHAR
REAL	C	REAL
SMALLINT	C	SMALLINT
TIME	E	DATETIME hour to second
TIMESTAMP	E	DATETIME hour to fraction(6)
TINYINT	E	SMALLINT
VARBINARY	E	VARCHAR
VARCHAR	C	VARCHAR

C Indicates a Core-level data type
E Indicates an Extended-level data type

Expressions and Operators

The NonStop ODBC Server supports all of the operators used in ODBC expression syntax. Table A-2 summarizes the ODBC operators.

Table A-2. NonStop ODBC Server Support of ODBC Operators

ODBC Operator Type	Symbol	NonStop ODBC Server Support
Arithmetic	+	x
	–	x
	*	x
	/	x
Comparison	=	x
	<>	x
	>	x
	>=	x
	<	x
	>=	x

x Indicates a supported function

Functions

The NonStop ODBC Server supports most ODBC functions.

Table A-3 summarizes NonStop ODBC Server support of ODBC functions.

Table A-3. ODBC Functions (page 1 of 3)

Function	NonStop ODBC Server Support	Comments
Date Functions		
CURDATE	x	Fully supported.
CURTIME	x	Fully supported.
DAYOFMONTH	x	Fully supported. Date and time expressions used within the function, however, have some restrictions.
DAYOFWEEK	x	Fully supported. Date and time expressions used within the function, however, have some restrictions.
DAYOFYEAR	–	–

x Indicates a supported function
– Indicates an unsupported function

Table A-3. ODBC Functions (page 2 of 3)

Function	NonStop ODBC Server Support	Comments
HOUR	x	Fully supported. Date and time expressions used within the function, however, have some restrictions.
MINUTE	x	Fully supported. Date and time expressions used within the function, however, have some restrictions.
MONTH	x	Fully supported. Date and time expressions used within the function, however, have some restrictions.
NOW	x	Fully supported.
QUARTER	–	–
SECOND	x	Fully supported. Date and time expressions used within the function, however, have some restrictions.
WEEK	–	–
YEAR	x	Fully supported. Date and time expressions used within the function, however, have some restrictions.
Mathematical Functions		
ABS	–	–
ACOS	–	–
ASIN	–	–
ATAN	–	–
ATAN2	–	–
CEILING	–	–
COS	–	–
COT	–	–
EXP	x	Fully supported.
FLOOR	–	–
LOG	–	–
MOD	x	Fully supported.
PI	x	Fully supported.
RAND	–	–
SIGN	–	–
SIN	–	–

x Indicates a supported function
– Indicates an unsupported function

Table A-3. ODBC Functions (page 3 of 3)

Function	NonStop ODBC Server Support	Comments
SQRT	–	–
TAN	–	–
String Functions		
ASCII	–	–
CHAR	–	–
CONCAT	x	Fully supported.
INSERT	–	–
LEFT	–	–
LTRIM	x	Fully supported.
LENGTH	x	Fully supported.
LOCATE	–	Fully supported.
LCASE	–	–
REPEAT	–	–
REPLACE	–	–
RIGHT	–	–
RTRIM	x	Fully supported.
SUBSTRING	x	Fully supported.
UCASE	x	Fully supported.
System Functions		
DATABASE	x	Fully supported.
IFNULL	–	–
USER	x	Fully supported.
Type Conversion Functions		
CONVERT	x	There are restrictions when converting to or from a datetime value.

x Indicates a supported function
– Indicates an unsupported function

Identifiers

The NonStop ODBC Server supports ODBC identifiers for naming all objects except databases and owners.

Database Names

The format for a database name is:

node_volume_subvolume

You must separate the parts of the name with underscores (_).

Each portion of the name begins with a letter and consists of letters and numerals. The maximum number of characters for each portion is as follows:

Portion	Characters
node	7
volume	6
subvolume	8

The NonStop ODBC Server maps the database name to a Guardian identifier as follows:

\node.$*volume*.*subvolume*

When specifying a database name in NonStop ODBC Server applications, do not specify the backslash (\) and dollar sign ($) as part of the name.

Owner Names

Each SQL object belongs to an owner, identified by the logical username associated with a Guardian logon ID by an ADDNAME USER statement.

The format for an owner name is:

logical-username

Case Sensitivity

Identifiers used with the NonStop ODBC Server are not case sensitive (My_Table is the same as my_table).

NULL Values

The NonStop ODBC Server supports NULL values in the same way as ODBC.

Search Conditions

The NonStop ODBC Server supports all ODBC search condition syntax.

Statements

The NonStop ODBC Server supports all of the ODBC CORE SQL statements, although some have restrictions.

Table A-4 lists the CORE SQL statements and summarizes how each supported statement differs when used with the NonStop ODBC Server.

Table A-4. NonStop ODBC Server Support of CORE SQL Statements (page 1 of 2)

CORE SQL Statement	NonStop ODBC Server Support	Comments
ALTER TABLE	x	Fully supported.
CALL	x	Extension to CORE SQL—must appear in an escape clause.
CREATE INDEX	x	The fully expanded index name must be unique in the network.
CREATE TABLE	x	Of the column constraint definitions, only NOT NULL is supported Of the table constraint definitions, only PRIMARY KEY is supported The fully expanded table name must be unique in the network.
CREATE VIEW	x	The fully expanded view name must be unique in the network.
DELETE	x	The NonStop ODBC Server supports both the positioned DELETE and the searched DELETE.
DROP INDEX	x	Fully supported.
DROP TABLE	x	Fully supported.
DROP VIEW	x	Dependent views are automatically dropped.
GRANT	x*	The NonStop ODBC Server recognizes the syntax, but does not execute the statement.
INSERT	x	Fully supported.

x Indicates that the statement is supported
* Indicates that the statement has limitations

Table A-4. NonStop ODBC Server Support of CORE SQL Statements (page 2 of 2)

CORE SQL Statement	NonStop ODBC Server Support	Comments
REVOKE	x*	The NonStop ODBC Server recognizes the syntax, but does not execute the statement.
SELECT	x	Fully supported.
UPDATE	x	The NonStop ODBC Server supports both the positioned UPDATE and the searched UPDATE.

x Indicates that the statement is supported
* Indicates that the statement has limitations

Stored Procedures

The NonStop ODBC Server supports execution of stored procedures using the CALL statement. Stored procedures cannot be created using CORE SQL, however; they must be created in the Pathway environment.

Wild-Card Characters

The NonStop ODBC Server supports both of the CORE SQL wild-card characters. Table A-5 summarizes the wild-card characters.

Table A-5. NonStop ODBC Server Support of Wild-Card Characters

ODBC Wild-Card Character	NonStop ODBC Server Support
%	x
_	x

x Indicates a supported function

B Summary of Support for SQL Server Features

Not all SQL Server features are supported by the HP NonStop ODBC Server. This appendix summarizes the following features of SQL Server and, for each feature, indicates whether the NonStop ODBC Server supports the feature:

- Aggregates
- Batch queries
- Browse mode
- Comments
- COMPUTE BY clause
- Data types
- Expressions and operators
- Functions
- Identifiers
- Joins
- NULL values
- Parameters
- Search conditions
- Statements
- Stored procedures
- System procedures
- System tables
- Variables
- Wild-card characters

Aggregates

The NonStop ODBC Server supports all of the SQL Server aggregate functions:

- AVG
- COUNT
- MAX
- MIN
- SUM

The primary differences between aggregate functions in SQL Server and in the NonStop ODBC Server are the following:

Feature	In SQL Server	In the NonStop ODBC Server
Data type of the results of AVG, COUNT, and SUM	Depends on the expression being evaluated.	Always floating point
Data type of the results of MAX and MIN	Depends on the expression being evaluated.	Depends on the expression being evaluated. An INT expression returns a floating point value.
What is the default if the DISTINCT clause is omitted from the COUNT function?	All rows are counted	Only distinct rows are counted.
Are aggregates and row aggregates distinguished?	Yes	No, because the row aggregates are part of the SQL Server COMPUTE BY clause, which is not supported by the NonStop ODBC Server.

For detailed information on aggregates, see Aggregates on page 4-31.

The NonStop ODBC Server supports batch queries.

The following table summarizes how batches differ in SQL Server and the NonStop ODBC Server.

Feature	In SQL Server	In the NonStop ODBC Server
Executing USE DATABASE inside a batch	Does not take effect until the batch is executed	Takes effect immediately
Are Control-of-Flow statements supported?	Yes	No

Browse Mode

The NonStop ODBC Server does not support browse mode. The FOR BROWSE clause is an unsupported clause of the SELECT statement.

For detailed information on the SELECT statement, see SELECT on page 4-82.

Comments

The NonStop ODBC Server supports user-written comments. The syntax is the same as in SQL Server. Nested comments, however, are not supported.

For detailed information on comments, see Comments on page 4-35.

COMPUTE BY Clause

The NonStop ODBC Server does not support the COMPUTE BY clause, which is a clause of the SELECT statement.

For detailed information on the SELECT statement, see SELECT on page 4-82.

Data Types

The NonStop ODBC Server supports most SQL Server data types. When you create objects and data, however, the NonStop ODBC Server maps the data to a HP NonStop SQL/MP data type.

Table B-1 summarizes the SQL Server data types and the corresponding NonStop SQL/MP data types.

Table B-1. Conversion of SQL Server Data Types to NonStop SQL/MP Data Types (page 1 of 2)

SQL Server Data Type	NonStop ODBC Server Support	Corresponding NonStop SQL/MP Data Type
BINARY	–	–
BIT	x	SMALLINT
CHAR	x	CHAR
DATETIME	x	DATETIME year to fraction (3)
FLOAT	x	DOUBLE PRECISION
IMAGE	–	–
INT	x	INTEGER
MONEY	x	DOUBLE PRECISION

x Indicates a supported data type
– Indicates an unsupported data type

Table B-1. Conversion of SQL Server Data Types to NonStop SQL/MP Data Types (page 2 of 2)

SQL Server Data Type	NonStop ODBC Server Support	Corresponding NonStop SQL/MP Data Type
SMALLINT	x	SMALLINT
SYSNAME	x	VARCHAR (30)
TEXT	x	VARCHAR
TINYINT	x	SMALLINT
USER_TYPE	–	–
VARBINARY	–	–
VARCHAR	x	VARCHAR

x Indicates a supported data type
– Indicates an unsupported data type

For detailed information on data types, see Data Types on page 4-12.

Expressions and Operators

The NonStop ODBC Server supports SQL Server expression syntax except for some operators. Table B-2 shows NonStop ODBC Server support of SQL Server operators.

Table B-2. NonStop ODBC Server Support of SQL Server Operators (page 1 of 2)

SQL Server Operator Type	Symbol	NonStop ODBC Server Support
Arithmetic	+	x
	-	x
	*	x
	/	x
	%	x
Bitwise	&	–
	|	–
	^	–
	~	–
Comparison	=	x
	>	x
	<	x
	>=	x
	<=	x

xIndicates a supported operator
–Indicates an unsupported operator

Table B-2. NonStop ODBC Server Support of SQL Server Operators (page 2 of 2)

SQL Server Operator Type	Symbol	NonStop ODBC Server Support
	!=	x
	!>	x
	!<	x
	*=	x
	=*	x
Other	ALL	x
	AND	x
	ANY	x
	BETWEEN	x
	EXISTS	x
	IN	x
	IS NULL	x
	LIKE	x
	NOT	x
	OR	x

xIndicates a supported operator
–Indicates an unsupported operator

For detailed information on expressions and operators, see Expressions and Operators on page 4-29.

Functions

The NonStop ODBC Server does not support most SQL Server functions.

Table B-3 summarizes NonStop ODBC Server support of SQL Server functions.

Table B-3. SQL Server Functions (page 1 of 4)

Function	NonStop ODBC Server Support	Comments
Date Functions		
DATEADD	x	Fully supported. Dateparts used within the function, however, have some restrictions.
DATEDIFF	x	Fully supported. Dateparts used within the function, however, have some restrictions.
DATENAME	–	–

x Indicates a supported function
– Indicates an unsupported function

Table B-3. SQL Server Functions (page 2 of 4)

Function	NonStop ODBC Server Support	Comments
DATEPART	x	Fully supported. Dateparts used within the function, however, have some restrictions.
GETDATE	x	Fully supported. Dateparts used within the function, however, have some restrictions.
Mathetical Functions		
ABS	–	–
ACOS	–	–
ASIN	–	–
ATAN	–	–
ATN2	–	–
CEILING	–	–
COS	–	–
COT	–	–
DEGREES	–	–
EXP	x	Fully supported.
FLOOR	–	–
LOG	–	–
LOG10	–	–
PI	x	Fully supported.
POWER	x	Fully supported.
RADIANS	–	–
RAND	–	–
ROUND	–	–
SIGN	–	–
SIN	–	–
SQRT	–	–
TAN	–	–
String Functions		
ASCII	–	–
CHAR	–	–
CHARINDEX	–	–
DIFFERENCE	–	–
LOWER	–	–

x Indicates a supported function
– Indicates an unsupported function

Table B-3. SQL Server Functions (page 3 of 4)

Function	NonStop ODBC Server Support	Comments
LTRIM	–	–
REPLICATE	–	–
RIGHT	–	–
RTRIM	–	–
SOUNDEX	–	–
SPACE	–	–
STR	–	–
STUFF	–	–
SUBSTRING	–	–
UPPER	x	Fully supported.
+	–	–
System Functions		
COL_LENGTH	–	–
COL_NAME	–	–
DATALENGTH	–	–
DB_ID	–	–
DB_NAME	x	All parameters are ignored, and the function always returns the current database name.
HOST_ID	–	–
HOST_NAME	–	–
INDEX_COL	–	–
ISNULL	–	–
OBJECT_ID	–	–
OBJECT_NAME	–	–
SUSER_ID	–	–
SUSER_NAME	x	All parameters are ignored, and the function always returns the current username.
USER_ID	x	All parameters are ignored, and the function always returns an integer used internally to identify a user.
Text Functions		
PATINDEX	–	–
TEXTPTR	–	–

x Indicates a supported function
– Indicates an unsupported function

Table B-3. SQL Server Functions (page 4 of 4)

Function	NonStop ODBC Server Support	Comments
TEXTVALID	–	–
Type Conversion Function		
CONVERT	x	The style parameter is ignored, and there are restrictions when converting to or from a datetime value.

x Indicates a supported function
– Indicates an unsupported function

SQL Server date functions contain arguments called dateparts. Table B-4 summarizes the dateparts supported by the NonStop ODBC Server.

Table B-4. NonStop ODBC Server Support of SQL Server Dateparts

Datepart	Abbreviation	NonStop ODBC Server Support
day	dd	x
dayofyear	dy	–
hour	hh	x
millisecond	ms	x
minute	mi	x
month	mm	x
quarter	qq	–
second	ss	x
week	wk	–
weekday	wd	–
year	yy	x

xIndicates a supported datepart
–Indicates an unsupported datepart

For detailed information on functions, see Functions on page 4-17.

Identifiers

The NonStop ODBC Server supports SQL Server identifiers for naming all objects except databases and owners.

Database Names

The format for a database name is:

```
node_volume_subvolume
```

You must separate the parts of the name with underscores (_).

Each portion of the name begins with a letter and consists of letters and numerals. The maximum number of characters for each portion is as follows:

Portion	Characters
node	7
volume	6
subvolume	8

The NonStop ODBC Server maps the database name to a Guardian identifier as follows:

```
\node.$volume.subvolume
```

When specifying a database name in NonStop ODBC Server applications, do not specify the backslash (\) and dollar sign ($) as part of the name.

Owner Names

Each SQL object belongs to an owner, identified by the logical username associated with a Guardian logon ID by an ADDNAME USER statement.

The format for an owner name is:

```
logical-username
```

Case Sensitivity

Identifiers used with the NonStop ODBC Server are not case sensitive (My_Table is the same as my_table). Identifiers used with SQL Server are case sensitive only if SQL Server is configured to have case-sensitive identifiers.

For detailed information on identifiers, see Names on page 4-5.

Joins

The NonStop ODBC Server supports all joins, including outer joins.

The following table summarizes NonStop ODBC Server support of the SQL Server join operators.

Operator	NonStop ODBC Server Support
=	x
>	x
<	x
>=	x
<=	x
!=	x
!>	x
!<	x
*=	x
=*	x

x Indicates a supported operator

NULL Values

The NonStop ODBC Server supports NULL values. The only difference between NULL values in SQL Server and NULL values in the NonStop ODBC Server is the order in which data is displayed.

The following table summarizes how NULL values affect the order in which data is displayed in SQL Server and in statements executed using the NonStop ODBC Server.

Clause	In SQL Server	When Executed Using the NonStop ODBC Server
GROUP BY	NULL values form their own group	Same as SQL Server
ORDER BY	NULL values come before all others.	NULL values come after all others.
SELECT clause with the DISTINCT keyword	NULL values are considered duplicates of each other. Only one NULL is selected, no matter how many are encountered.	Same as SQL Server

Parameters

The NonStop ODBC Server supports both input parameters and input/output parameters, values supplied to stored procedures at the time of invocation.

Y2K Implicit Century Conformance

The NonStop ODBC Server supports an implicit century DATE value with the same semantics as found in NonStop SQL Server. When NonStop ODBC/MP can determine by context that an input buffer value is a two-digit year value, it prefixes the year value with a century value of either “00” or “20” before the statement is executed. The prefix “19” is used if the year value is greater than 50; the prefix “20” is used if the year value is less than or equal to 50. For example, an input value of “00/01/01” would be interpreted as “2000/01/01,” while an input value of “99/01/01” would be interpreted as “1999/01/01.”

Search Conditions

The NonStop ODBC Server supports all search condition syntax except the following:

```
WHERE [ NOT ] column-name join-operator column-name
```

For detailed information on search conditions, see Search Conditions on page 4-28.

Statements

The SQL statements you can use with the NonStop ODBC Server are a subset of the SQL Server Transact-SQL statements.

Table B-5 lists the Transact-SQL statements and summarizes how each supported statement differs when used with the NonStop ODBC Server. If you use an unsupported statement with the NonStop ODBC Server, an error message is generated.

Table B-5. NonStop ODBC Server Support of Transact-SQL Statements (page 1 of 7)

Transact-SQL Statement	NonStop ODBC Server Support	Comments
ALTER DATABASE	–	–
ALTER TABLE	x	Fully supported.
BEGIN...END	x	These statements are fully supported; however, in SQL Server, BEGIN and END are used primarily to enclose control-of-flow statements (such as IF, THEN, ELSE, and WHILE), and the control-of-flow statements are not supported by the NonStop ODBC Server.

x Indicates that the statement is supported
– Indicates that the statement is not supported

Table B-5. NonStop ODBC Server Support of Transact-SQL Statements (page 2 of 7)

Transact-SQL Statement	NonStop ODBC Server Support	Comments
BEGIN TRANSACTION	x	Nested transactions are not allowed. Transaction names can be included but are not meaningful. DDL operations are allowed within a transaction.
BREAK	–	–
CHECKPOINT	–	–
COMMIT TRANSACTION	x	Transaction savepoints are ignored. Transaction names can be included but are ignored.
CONTINUE	–	–
CREATE DATABASE	x	The ON DEFAULT clause is not supported. You cannot specify the amount of space to allocate for the database. NonStop SQL/MP allocates disk space for tables and other objects as you create them and enter data.
CREATE DEFAULT	–	Use the DEFAULT keyword on the NonStop SQL/MP CREATE TABLE or ALTER TABLE statement in pass-through mode.
CREATE INDEX	x	The following clauses are not supported: ● CLUSTERED ● WITH index-option You cannot create a unique index on a column that allows null values. You can index BIT type columns; however, the NonStop ODBC Server maps the BIT data type to the NonStop SQL/MP SMALLINT data type. You can index TEXT type columns; however, the NonStop ODBC Server maps the TEXT data type to the NonStop SQL/MP VARCHAR data type. The fully expanded index name must be unique in the network.
CREATE PROCEDURE	–	–

x Indicates that the statement is supported
– Indicates that the statement is not supported

Table B-5. NonStop ODBC Server Support of Transact-SQL Statements (page 3 of 7)

Transact-SQL Statement	NonStop ODBC Server Support	Comments
CREATE RULE	–	Use the NonStop SQL/MP CREATE CONSTRAINT statement in pass-through mode.
CREATE TABLE	x	Some data types are unsupported, and the NonStop ODBC Server maps some data types to NonStop SQL/MP data types. See Data Types earlier in this appendix. The fully expanded table name must be unique in the network.
CREATE TRIGGER	–	–
CREATE VIEW	x	You can modify data in a view only if the view is derived from one table. The fully expanded view name must be unique in the network.
DBCC	–	–
DECLARE	x	Some data types are unsupported, and the NonStop ODBC Server maps some data types to NonStop SQL/MP data types. See Data Types earlier in this appendix. For information on usage differences, see Variables later in this appendix.
DELETE	x	The FROM clause is not supported. You cannot specify more than one table name—you cannot delete rows based on data stored in other tables. DELETE must be inside a transaction if the table is audited.
DISK INIT	–	–
DISK REFIT	–	–
DISK REINIT	–	–
DROP DATABASE	x	Fully supported.
DROP DEFAULT	–	You cannot drop a default in NonStop SQL/MP.

x Indicates that the statement is supported
– Indicates that the statement is not supported

Table B-5. NonStop ODBC Server Support of Transact-SQL Statements (page 4 of 7)

Transact-SQL Statement	NonStop ODBC Server Support	Comments
DROP INDEX	x	The following associated objects must be accessible: ● The underlying table ● All object program files using the underlying table (if any) ● Catalogs containing the description of the index
DROP PROCEDURE	–	–
DROP RULE	–	Use the NonStop SQL/MP DROP CONSTRAINT statement in pass-through mode.
DROP TABLE	x	Dependent views are automatically dropped, but you should drop them first because they are not dropped from the mapping tables.
DROP TRIGGER	–	–
DROP VIEW	x	Dependent views are automatically dropped.
DUMP DATABASE	–	–
DUMP TRANSACTION	–	–
EXECUTE	x	The WITH RECOMPILE clause is not supported.
GOTO	–	–
GRANT	Accepted, but not performed	–
IF...ELSE	–	–

x Indicates that the statement is supported
– Indicates that the statement is not supported

Table B-5. NonStop ODBC Server Support of Transact-SQL Statements (page 5 of 7)

Transact-SQL Statement	NonStop ODBC Server Support	Comments
INSERT	x	You cannot insert rows into a view that references more than one table, even if the columns being inserted belong to only one table. INSERT must be inside a transaction if the table is audited. The select-list must follow these NonStop SQL/MP rules: ● The *`select-list`* must contain an element for each column that you specify in the *`column-name`* list. ● A subquery cannot refer to a table, view, or underlying table of the view into which rows are being inserted.
KILL	–	–
LOAD DATABASE	–	–
LOAD TRANSACTION	–	–
PRINT	x	Fully supported.
RAISERROR	–	–
READTEXT	–	–
RECONFIGURE	–	–
RETURN	–	–
REVOKE	Accepted, but not performed	–
ROLLBACK TRANSACTION	x	Transaction savepoints are ignored.
SAVE TRANSACTION	x	If used in the NonStop ODBC Server, the SAVE TRANSACTION statement is ignored. No error or warning messages are generated.

x Indicates that the statement is supported
– Indicates that the statement is not supported

Table B-5. NonStop ODBC Server Support of Transact-SQL Statements (page 6 of 7)

Transact-SQL Statement	NonStop ODBC Server Support	Comments
SELECT	x	The following clauses are not supported: ● INTO ● GROUP BY ALL ● COMPUTE BY ● FOR BROWSE You cannot include expressions in an aggregate-free expression. SELECT must be inside a transaction if it queries audited objects. Vector aggregates are not supported. You cannot mix aggregate and nonaggregate expressions in the select-list.
SET	x	Only two parameters are supported: ● PARSEONLY ● TEXTSIZE.
SETUSER	–	–
SHUTDOWN	–	–
TRUNCATE TABLE	x	TRUNCATE TABLE works at the same speed as DELETE. Changes are written to the audit log. TRUNCATE TABLE must be inside a transaction if the table is audited.
UPDATE	x	The FROM clause is not supported. You can update data in a view only if the view is derived from one table. UPDATE must be inside a transaction if the table is audited. Aggregate functions are not allowed in the expression.
UPDATE STATISTICS	x	You cannot specify the index for which to update statistics.
USE	x	The new database takes effect as soon as the statement is executed, not when the current batch finishes.

x Indicates that the statement is supported
– Indicates that the statement is not supported

Table B-5. NonStop ODBC Server Support of Transact-SQL Statements (page 7 of 7)

Transact-SQL Statement	NonStop ODBC Server Support	Comments
WAITFOR	–	–
WHILE	–	–
WRITETEXT	–	–

x Indicates that the statement is supported
– Indicates that the statement is not supported

For detailed information on supported statements, see Section 4, Transact-SQL Language.

Stored Procedures

The NonStop ODBC Server supports the execution of stored procedures, using the EXECUTE statement, including the use of parameters supplied to the stored procedures at the time of invocation. Stored procedures cannot be created using Transact-SQL, however, but must be created in the Pathway environment.

System Procedures

The NonStop ODBC Server does not support SQL Server system procedures. Table B-6 lists the system procedures and, where possible, offers alternatives to using system procedures. Note that you can execute a NonStop SQL/MP statement by using pass-through mode with the NonStop ODBC Server.

Table B-6. SQL Server System Procedures (page 1 of 3)

System Procedure	Alternatives Available Using NonStop SQL/MP or the HP NonStop Kernel Operating System
sp_addalias	You can add alias usernames with the ADDNAME ALIAS statement.
sp_addgroup	None. Groups are not supported and are not available in NonStop SQL/MP.
sp_addlogin	Must be done by a Guardian administrator. Anyone with a Guardian login can access the NonStop ODBC Server and NonStop SQL/MP.
sp_addtype	None
sp_addumpdevice	None
sp_adduser	You can add logical usernames with the ADDNAME USER statement. A logical username must correspond to a Guardian logon ID.
sp_bindefault	Use the DEFAULT keyword on CREATE TABLE or ALTER TABLE.
sp_bindrule	Use CREATE CONSTRAINT.

Table B-6. SQL Server System Procedures (page 2 of 3)

System Procedure	Alternatives Available Using NonStop SQL/MP or the HP NonStop Kernel Operating System
sp_changedbowner	Must be done with Guardian permissions. You can alter file permissions in NonStop SQL/MP using the ALTER TABLE or SECURE statements.
sp_commonkey	None
sp_configure	You can configure all tables or specific tables in NonStop SQL/MP using the CONTROL TABLE statement.
sp_dboption	None
sp_defaultdb	You can establish a default database by using the ADDNAME USER statement; otherwise, the default database is always MASTER.
sp_depends	You can select against the USAGES table in NonStop SQL/MP.
sp_diskdefault	None
sp_dropalias	You can drop alias usernames with the DELETENAME ALIAS statement.
sp_dropdevice	None
sp_dropgroup	None
sp_dropkey	None. You cannot remove a key.
sp_droplogin	Must be done by a Guardian administrator
sp_droptype	None
sp_dropuser	You can drop usernames with the DELETENAME USER statement.
sp_foreignkey	None
sp_help	Use INVOKE to view column names. Use Guardian statements to view file information such as owner, permissions, and last modification. To determine the uses of an object, select against the NonStop SQL/MP catalog tables using the NonStop SQL/MP DISPLAY USE OF utility statement.
sp_helpdb	To determine the uses of a database, select against the NonStop SQL/MP catalog tables using the NonStop SQL/MP DISPLAY USE OF utility statement.
sp_helpdevice	None. Devices are not supported.
sp_helpgroup	None. Groups are not supported and are not available in NonStop SQL/MP.
sp_helpindex	To determine the uses of an index, select against the NonStop SQL/MP catalog tables using the NonStop SQL/MP DISPLAY USE OF utility statement.
sp_helpjoins	None
sp_helpkey	None

Table B-6. SQL Server System Procedures (page 3 of 3)

System Procedure	Alternatives Available Using NonStop SQL/MP or the HP NonStop Kernel Operating System
sp_helpprotect	Use Guardian commands to view file security.
sp_helpsql	You can use SQLCI to get help on NonStop SQL/MP statements. You must use SQLCI interactively, however; you cannot get help using the NonStop ODBC Server.
sp_helptext	You can see the text of a view by selecting against the NonStop SQL/MP catalog tables.
sp_helpuser	None
sp_lock	You can display locks on an object or all objects in a subvolume using FUP, but you cannot do it using NonStop SQL/MP.
sp_logdevice	None
sp_monitor	None
sp_password	Must be done with Guardian commands.
sp_primarykey	Use the NonStop SQL/MP CREATE TABLE statement.
sp_rename	Use the NonStop SQL/MP ALTER TABLE, ALTER VIEW, or ALTER INDEX statements. You must refresh the NonStop ODBC Server mapping tables after renaming an object. You cannot change a column name.
sp_renamedb	None. You cannot rename a NonStop SQL/MP catalog.
sp_spaceused	You can show the space used with the TACL FILEINFO statement. You can find the number of rows in a table by using the SELECT statement.
sp_unbindefault	None. You cannot drop a default.
sp_unbindrule	Use the NonStop SQL/MP DROP CONSTRAINT statement.
sp_who	None

For information on pass-through mode, see Section 6, Using Pass-Through Mode.

System Tables

The NonStop ODBC Server supports some of the SQL Server system tables. The following tables summarize NonStop ODBC Server support of the system tables.

For detailed information on how the NonStop ODBC Server supports the system tables, see Section 8, HP NonStop ODBC Server Mapping Tables.

System Tables in All Databases

Table B-7. NonStop ODBC Server Support of System Tables in All Databases (page 1 of 5)

		Columns			
Table Name	**NonStop ODBC Server Support**	**Column Name**	**NonStop ODBC Server Support**	**Data Type**	**Comments or Value Assigned**
sysalternates	–	–	–	–	–
syscolumns	x	id	x	INT (4)	An object ID used internally by the NonStop ODBC Server.
		number	–	–	Assigned 0
		colid	x	INT (2)	A number indicating the position of the column in the row of the table (the first column is 0).
		status	x	INT (2)	A number indicating whether the SQL Server data type allows null values.
		type	x	INT (2)	The SQL Server data type code.
		length	x	INT (2)	The byte length of the data in the column.
		offset	x	INT (2)	Reserved for internal use by HP software.
		usertype	x	INT (2)	An SQL Server user-defined data type code.
		cdefault	–	–	Assigned -1
		domain	–	–	Assigned -1
		name	x	CHAR (30)	The column name, which is a NonStop SQL/MP identifier.

xIndicates a supported table or column
–Indicates an unsupported table or column

Table B-7. NonStop ODBC Server Support of System Tables in All Databases (page 2 of 5)

		Columns			
Table Name	**NonStop ODBC Server Support**	**Column Name**	**NonStop ODBC Server Support**	**Data Type**	**Comments or Value Assigned**
		printfmt	–	–	Assigned “ ” (blank)
syscomments	–	–	–	–	–
sysdepends	–	–	–	–	–
sysindexes	x	name	x	VARCHAR (60)	The fully qualified SQL Server index name in uppercase letters.
		id	x	INT (4)	The object ID of the table.
		indid	x	INT (2)	Primary key indicator.
		dpages	–	–	Assigned -1
		reserved	–	–	Assigned -1
		used	–	–	Assigned -1
		rows	–	–	Assigned -1
		first	–	–	Assigned -1
		root	–	–	Assigned -1
		distribution	–	–	Assigned -1
		usagecnt	–	–	Assigned -1
		segment	–	–	Assigned -1
		status	x	INT (2)	Assigned -1
		rowpage	–	–	Assigned -1
		minlen	x	INT (2)	The length of the packed index record.
		maxlen	–	–	Assigned -1
		maxirow	–	–	Assigned -1
		keycnt	x	INT (2)	The number of columns in the index.
		keys1	–	–	Assigned “ ” (blank)
		keys2	–	–	Assigned “ ” (blank)
syskeys	–	–	–	–	–

xIndicates a supported table or column
–Indicates an unsupported table or column

Table B-7. NonStop ODBC Server Support of System Tables in All Databases (page 3 of 5)

		Columns			
Table Name	NonStop ODBC Server Support	Column Name	NonStop ODBC Server Support	Data Type	Comments or Value Assigned
syslogs	–	–	–	–	–
sysobjects	x	name	x	VARCHAR (60)	The fully qualified SQL Server object name in uppercase letters.
		id	x	INT (4)	An object ID used internally by the NonStop ODBC Server.
		uid	x	INT (2)	A user ID used internally by the NonStop ODBC Server.
		type	x	CHAR (2)	A code indicating the object type: N NonStop SQL/MP catalog table or NonStop ODBC Server mapping table S System table view T Temporary table U User table V View
		userstat	–	–	Assigned -1
		sysstat	–	–	Assigned -1
		indexdel	–	–	Assigned -1
		schema	–	–	Assigned -1
		refdate	–	–	Assigned “ ” (blank)
		crdate	x	DATETIME	Date and time the object was created.
		expdate	–	–	Assigned “ ” (blank)
		deltrig	–	–	Assigned -1
		instrig	–	–	Assigned -1
		updtrig	–	–	Assigned -1
		seltrig	–	–	Assigned -1
		category	–	–	Assigned -1

xIndicates a supported table or column
–Indicates an unsupported table or column

Table B-7. NonStop ODBC Server Support of System Tables in All Databases (page 4 of 5)

		Columns			
Table Name	**NonStop ODBC Server Support**	**Column Name**	**NonStop ODBC Server Support**	**Data Type**	**Comments or Value Assigned**
		cache	–	–	Assigned -1
sysprocedures	–	–	–	–	–
sysprotects	x	id	x	INT (4)	An object ID used internally by the NonStop ODBC Server.
		uid	x	INT (2) UNSIGNED	A user ID used internally by the NonStop ODBC Server.
		action	x	INT (2)	An integer corresponding to a Transact-SQL statement, for example: 193 SELECT 195 INSERT 196 DELETE 197 UPDATE
		protecttype	x	INT (2)	An integer, 205, corresponding to GRANT permission. REVOKE (206) is not supported. When using the NonStop ODBC Server, all users are granted permission for all statements.
		columns	–	–	Assigned “ ” (blank)
syssegments	–	–	–	–	–
systypes	x	uid	–	–	Assigned 1
		usertype	x	INT (2)	An SQL Server user-defined data type code.
		variable	–	–	Assigned -1
		allownulls	–	–	Assigned -1
		type	x	INT (2)	The SQL Server data type code.
		length	–	–	Assigned -1
		tdefault	–	–	Assigned 0

xIndicates a supported table or column
–Indicates an unsupported table or column

Table B-7. NonStop ODBC Server Support of System Tables in All Databases (page 5 of 5)

		Columns			
Table Name	**NonStop ODBC Server Support**	**Column Name**	**NonStop ODBC Server Support**	**Data Type**	**Comments or Value Assigned**
		domain	–	–	Assigned 0
		name	x	VARCHAR (30)	The SQL Server data type name.
		printfmt	–	–	Assigned “ ” (blank)
sysusers	x	suid	–	–	Assigned 1
		uid	x	INT (2) UNSIGNED	A user ID used internally by the NonStop ODBC Server.
		gid	–	–	Assigned -1
		name	x	VARCHAR (60)	The fully qualified owner name in uppercase letters.
		environ	–	–	Assigned “ ” (blank)

xIndicates a supported table or column
–Indicates an unsupported table or column

System Tables in the Master Database

Table B-8. NonStop ODBC Server Support of System Tables in the Master Database Only (page 1 of 2)

		Columns			
Table Name	**NonStop ODBC Server Support**	**Column Name**	**NonStop ODBC Server Support**	**Data Type**	**Comments or Value Assigned**
sysconfigures	–	–	–	–	–
syscurconfigs	–	–	–	–	–
sysdatabases	x	name	x	VARCHAR (60)	The fully qualified SQL Server database name (database.owner) in uppercase letters.
		dbid	x	INT (2) UNSIGNED	An internal ID used by the NonStop ODBC Server.
		suid	x	INT (2) UNSIGNED	The SQL Server user ID of the database creator. If the database is created in NonStop SQL/MP and then customized, the creator is the user ID of the user who customizes the catalog.
		mode	–	–	Assigned -1
		status	–	–	Assigned -1
		version	x	INT (2)	NonStop ODBC Server version number.
		crdate	x	DATETIME	The date and time the catalog was created or customized.
sysdevices	–	–	–	–	–
syslocks	–	–	–	–	–
syslogins	–	–	–	–	–

x Indicates a supported table or column
– Indicates an unsupported table or column

Table B-8. NonStop ODBC Server Support of System Tables in the Master Database Only (page 2 of 2)

		Columns			
Table Name	NonStop ODBC Server Support	Column Name	NonStop ODBC Server Support	Data Type	Comments or Value Assigned
sysmessages	x	error	x	INT (4)	The SQL Server error code that corresponds to this message. If there is no corresponding SQL Server error code, the code is 18001.
		severity	x	INT (2)	The SQL Server error severity level: 10 Warning 11-16 User-generated 17-18 Nonfatal error 19+ Fatal error
		dlevel	x	INT (2)	The SQL Server error detail level. Detail level is not supported and is always -1.
		description	x	VARCHAR (255)	The error text.; it can be SQL Server text, NonStop SQL/MP text, or NonStop ODBC Server text.
sysprocesses	–	–	–	–	–
sysusages	–	–	–	–	–

x Indicates a supported table or column
– Indicates an unsupported table or column

Variables

The NonStop ODBC Server supports local and global variables.

There are some restrictions for using local variables:

- A variable cannot be null when it is referenced.
- Variable expressions are not allowed in date functions.
- You cannot mix nonaggregate and aggregate assignments to variables.
- Variable expressions are not allowed in some constructs.

Not all global variables are supported. Table B-9 summarizes NonStop ODBC Server support of global variables.

Table B-9. Global Variables

Global Variable	NonStop ODBC Server Support	Comments or Value Assigned
@@connections	x	The value is always 1.
@@cpu_busy	–	-1
@@error	x	No differences.
@@idle	–	-1
@@io_busy	–	-1
@@max_ connections	x	Depends on the system resources.
@@next_level	–	-1
@@pack_received	–	-1
@@pack_sent	–	-1
@@packet_errors	–	-1
@@procid	–	-1
@@rowcount	–	0
@@textsize	x	The default is 512 bytes. (In SQL Server, the default is 4096 bytes.)
@@timeticks	–	-1
@@total_errors	–	-1
@@total_read	–	-1
@@total_write	–	-1
@@trancount	x	The value will be either 0 or 1 because nested transactions are not supported. The default is 0.
@@version	–	NonStop ODBC Server T8666D20

x Indicates a supported global variable
– Indicates an unsupported global variable

For detailed information on variables, see Variables on page 4-26.

Wild-Card Characters

The NonStop ODBC Server supports two of the SQL Server wild-card characters.

Table B-10. NonStop ODBC Server Support of Wild-Card Characters

SQL Server Wild-Card Character	**NonStop ODBC Server Support**
%	x
_	x
[*character-range*]	–
[*character-set*]	–
[^*character-range*]	–
[^*character-set*]	–

x Indicates a supported wild-card character
– Indicates an unsupported wild-card character

When used with the NonStop ODBC Server, the % wild-card character works differently than it does in SQL Server:

- In the NonStop ODBC Server, character comparisons are not case sensitive; in SQL Server, they are case sensitive.
- Because NonStop SQL/MP recognizes trailing blanks as part of a string, you must place the % wild-card character at the end of a comparison; this is not necessary in SQL Server.

For detailed information on wild-card characters, see Wild-Card Characters on page 4-33.

C

Summary of Support for ODBC 2.10 Functions

Table C-1 summarizes the ODBC 2.10 functions and shows which are supported by the HP NonStop ODBC Server.

Table C-1. NonStop ODBC Server Support of ODBC 2.10 Functions (page 1 of 3)

Function Name	Description	Support
Core-Level Functions		
SQLAllocConnect	Allocates a connection	Fully supported
SQLAllocEnv	Allocates an ODBC environment	Fully supported
SQLAllocStmt	Allocates a statement	Fully supported
SQLBindCol	Binds a column to a statement	Fully supported
SQLCancel	Cancels an asynchronously running statement	Fully supported
SQLColAttributes	Returns one item of information about a column in a result set	Fully supported
SQLConnect	Connects an allocated connection to a data source	Fully supported
SQLDescribeCol	Returns several items of information about a column in a result set	Fully supported
SQLDisconnect	Closes a connection without freeing the connection handle	Fully supported
SQLError	Returns detailed status information for the ODBC function most recently called	Fully supported
SQLExecDirect	Prepares and executes a statement	Fully supported
SQLExecute	Executes a prepared statement	Fully supported
SQLFetch	Gets a row of data from a result set	Fully supported
SQLFreeConnect	Deallocates a connection handle	Fully supported
SQLFreeEnv	Deallocates an ODBC environment	Fully supported
SQLFreeStmt	Deallocates a statement handle or releases objects associated with a statement	Fully supported
SQLGetCursorName	Returns the cursor name for a statement	Fully supported
SQLNumResultCols	Returns the number of columns in a result set	Fully supported
SQLPrepare	Prepares a statement for execution by SQLExecute	Fully supported

Table C-1. NonStop ODBC Server Support of ODBC 2.10 Functions (page 2 of 3)

Function Name	Description	Support
SQLRowCount	Returns the number of rows affected by an INSERT, UPDATE, or DELETE operation	Fully supported
SQLSetCursorName	Names the cursor that belongs to a statement	Fully supported
SQLTransact	Commits or rolls back all transactions on a connection or all connections in an ODBC environment	Fully supported
Extension Level 1 Functions		
SQLBindParameter	Binds a parameter buffer	Fully supported*
SQLColumns	Returns a result set of columns in one or more tables	Fully supported
SQLDriverConnect	Connects an allocated connection to a data source, providing additional information	Fully supported
SQLGetConnectOption	Returns a connect option setting	Fully supported
SQLGetData	Returns data for an unbound column	Fully supported
SQLGetFunctions	Indicates whether a driver supports a specific ODBC function	Fully supported
SQLGetInfo	Returns one of many items of information about an ODBC driver	Fully supported
SQLGetStmtOption	Returns a statement option setting	Fully supported
SQLGetTypeInfo	Returns a result set of data types supported by a data source	Fully supported
SQLParamData	Part of the data-at-execution process	Fully supported
SQLPutData	Part of the data-at-execution process	Fully supported
SQLSetConnectOption	Sets a connection option	Fully supported
SQLSetStmtOption	Sets a statement option	Partially supported**
SQLSpecial Columns	Retrieves information about uniquely identifying and auto-update columns in a table	Fully supported
SQLStatistics	Returns a result set of statistics and index information about a table	Fully supported
SQLTables	Returns a result set of tables in a data source	Fully supported

Table C-1. NonStop ODBC Server Support of ODBC 2.10 Functions (page 3 of 3)

Function Name	**Description**	**Support**
Extension Level 2 Functions		
SQLBrowseConnect	Connects to a data source step by step	Fully supported
SQLColumnPrivileges	Returns a result set of column privileges for a table	Not supported
SQLDataSources	Lists the names of available data sources	Fully supported
SQLDescribeParam	Returns various items of information about a statement parameter	Not supported
SQLDrivers	Lists information about available drivers	Fully supported
SQLExtendedFetch	Retrieves several rows from a result set	Not supported
SQLForeignKeys	Returns a list of foreign keys for a table or its related tables	Not supported
SQLMoreResults	Indicates whether an additional result set has been returned	Fully supported
SQLNativeSQL	Accepts a SQL statement in ODBC syntax and returns it in a driver's syntax	Fully supported
SQLNumParams	Returns the number of parameters for a statement	Fully supported
SQLParamOptions	Part of the data-at-execution process	Not supported
SQLPrimaryKeys	Returns a result set of the columns in a table's primary key	Fully supported
SQLProcedureColumns	Returns a result set of input, output, and result set columns for a database or stored procedure	Fully supported
SQLProcedures	Returns the list of procedure names stored in a specific data source.	Not supported
SQLSetPos	Positions a cursor and, optionally, performs a positioned operation	Not supported
SQLSetScrollOptions	Sets options that control the behavior of cursors.	Not supported
SQLTablePrivileges	Returns a result set of privileges for one or more tables	Not supported

* Y2K implicit century conformance: If the *`ValueType`* is CHAR and if the date value in the buffer pointed to by *`ParameterValuePtr`* can be determined to have a two-digit year, the value is prefixed with the string “00” prior to using this parameter in the execution of the statement. (This translation occurs only for input parameters to ODBC. Also, if the input buffer contains one of the C DATE data type structures, SQL_C_TYPE_DATE, the SMALLINT year value will have already been translated into a four-digit value.) For example, an input value of “00/01/01” would be interpreted as “0000/01/01,” and an input value of “00/01/01” would be interpreted as “0099/01/01.”

** SQLSetStmtOption does not support the SQL_CURSOR_TYPE option set to SQL_CURSOR_KEYSET_DRIVE.

D

Summary of System Installation Defaults

Table D-1 lists the values stored in the ZNSSCFG table during system installation. Each value in ZNSSCFG is stored in ASCII VARCHAR format in the VALUE column. When a system default value is used in other tables (such as ZNSPROF, ZNSSER), the value is stored as the data type shown in the Table Data Type column.

For more information about default values, see Default Values for NonStop ODBC Server Attributes on page 8-4 and ZNSSCFG (For System Configuration Values) on page 8-28.

Table D-1. System Default Values Installed in ZNSSCFG (page 1 of 5)

Column Name or Attribute (Stored in the ITEM Column)	**Table Data Type**	**System Default (Stored in the VALUE Column)**
ACC_LEVEL	VARCHAR (60)	SESSION
ACC_LOGTABLE_NAME	VARCHAR (182)	" " (no default is assigned)
ACC_MODE_ON	CHAR (1)	N
AVAILABLE_SERVERS	INT (2) UNSIGNED	0
CACHE_STATISTICS	CHAR (1)	N
CHECK_INTERVAL_SECS	CHAR (20)	–1 (indicates no polling takes place regarding the validity of the attribute name in the ITEM column of the ZNSSCFG table)
COLLECT_SQL_ERROR_ INFO	CHAR(1)	This item is not available by default; it is only present if added using the ADD SCFG command.
CON_MODE_ON	CHAR (1)	N
CON_NAME	VARCHAR (60)	" " (no default is assigned)
CON_TEXT	VARCHAR (3900)	None
CPU_BACKUP	INT (2)	99 (indicates that a backup SCS process is not started)
CPU_LIST	VARCHAR (37)	–1 (indicates that PROCESS_ CREATE_ assigns the CPU)
CPU_PRIMARY	INT (2)	0 (indicates that PROCESS_ CREATE_ assigns the CPU)
DATAPAGES	INT (2)	0
DEFAULT_DATABASE	VARCHAR (60)	MASTER
DEFAULT_LOCATION	CHAR (16)	" " (no default is assigned)

Table D-1. System Default Values Installed in ZNSSCFG (page 2 of 5)

Column Name or Attribute (Stored in the ITEM Column)	Table Data Type	System Default (Stored in the VALUE Column)
DEFAULT_SECURITY	CHAR (4)	" " (no default is assigned)
DEFAULT_SCHEMA	VARCHAR (60)	" " (no default is assigned)
DEFAULT_VOLUME	CHAR (25)	" " (no default is assigned)
DEFINE_ATTRIBUTE	CHAR (16)	" " (no default is assigned)
DEFINE_CLASS	CHAR (16)	" " (no default is assigned)
DEFINE_NAME	CHAR (24)	" " (no default is assigned)
DEFINE_VALUE	VARCHAR (512)	" " (no default is assigned)
END_TIME	TIMESTAMP	None
EMIT_EVENTS	CHAR (1)	Y
ERR_FILE	CHAR (31)	" " (no default is assigned)
EXT_SWAPFILE	VARCHAR (35)	" " (no default is assigned)
G_USERNAME	CHAR (17)	" " (no default is assigned)
GOV_ACTION	VARCHAR (60)	None
GOV_ATTRIBUTE	VARCHAR (60)	None
GOV_MODE_ON	CHAR (1)	N
GOV_NAME	VARCHAR (60)	" " (no default is assigned)
HOMETERM	CHAR (24)	" " (no default is assigned)
IDLE_DELETE_DELAY_SEC	INT (4)	0
IN_BUFFER_SIZE_B	INT (2)	2448
IN_FILE	CHAR (35)	" " (no default is assigned)
INIT_HEAP_SIZE_KB	INT (2)	750
INPUT_STREAM	CHAR (1)	N
JOB_ID	INT (2)	1
LAST_UPDATE_SYSTEM_ CONFIG	CHAR (27)	Timestamp at time of installation
LIMIT_VALUE	INT (8)	None
LOG_TO_HOMETERM	CHAR (1)	N
MAX_HEAP_SIZE_KB	INT (2)	750
MAX_SERVERS	INT (2) UNSIGNED	1
MEMORY_CHECK	CHAR (1)	N
MLAN_ADAPTOR	INT (2)	0
MLAN_DOMAIN	VARCHAR (60)	None
MLAN_GATEWAY	VARCHAR (60)	" " (no default is assigned)

Table D-1. System Default Values Installed in ZNSSCFG (page 3 of 5)

Column Name or Attribute (Stored in the ITEM Column)	Table Data Type	System Default (Stored in the VALUE Column)
NET_NAME	VARCHAR (60)	None
NET_PROTOCOL	VARCHAR (60)	None
NET_QUALIFIER	VARCHAR (128)	" " (no default is assigned)
NOS_ALIASNAME	VARCHAR (60)	None
NOS_CREATE_OPTIONS	INT (2)	0
NOS_DEBUG_OPTIONS	INT (2)	0
NOS_LIBRARY_FILE	CHAR (35)	" " (no default is assigned)
NOS_OBJECT	CHAR (34)	$SYSTEM.SYSTEM.NOS
NOS_RUN_OPTIONS	VARCHAR (240)	" " (no default is assigned)
NOS_USERNAME	VARCHAR (60)	None
NOSUTIL_CREATE_OPTIONS	INT (2)	22
NOSUTIL_DEBUG_OPTIONS	INT (2)	0
NOSUTIL_OBJECT	CHAR (34)	$SYSTEM.SYSTEM.NOSUTIL
NOSUTIL_RUN_OPTIONS	VARCHAR (240)	" " (no default is assigned)
NSSQL	CHAR (1)	N
OBJ_CACHE_LEVEL	INT (4)	1
OUT_BUFFER_SIZE_B	INT (2)	2448
OUT_FILE	CHAR (35)	" " (no default is assigned)
OUTPUT_STREAM	CHAR (1)	N
PIPE_TEST	CHAR (1)	Obsolete; not used.
PRIORITY	INT (2)	–1 (indicates that the caller's priority is used)
PROFILE_NAME	VARCHAR (60)	DEFAULT
QST_MODE_ON	CHAR (1)	N
QST_LOGTABLE_NAME	VARCHAR (182)	" " (no default is assigned)
SCS_CREATE_OPTIONS	INT (2)	0
SCS_DEBUG_OPTIONS	INT (2)	0
SCS_NAME	CHAR (16)	None
SCS_LIBRARY_FILE	CHAR (35)	" " (no default is assigned)
SCS_OBJECT	CHAR (35)	$SYSTEM.SYSTEM.SCSOBJ
SCS_PRIORITY	INT (2)	–1 (indicates that the caller's priority is used)
SCS_RUN_OPTIONS	VARCHAR (240)	" " (no default is assigned)
SER_NAME	CHAR (32)	None

Table D-1. System Default Values Installed in ZNSSCFG (page 4 of 5)

Column Name or Attribute (Stored in the ITEM Column)	**Table Data Type**	**System Default (Stored in the VALUE Column)**
SO_KEEPALIVE	INT (4)	0
SO_OOBINLINE	INT (4)	0
SO_LINGER	INT (4)	0
SO_REUSEADDR	INT (4)	0
SP_READ	CHAR (1)	N
SP_WRITE	CHAR (1)	N
SQL_ACCESS_MODE	CHAR (2)	RW
SQL_CURSOR_MODE	CHAR (2)	RW (SQL_CURSOR_ DEFAULT affects cursors not specified as FOR READ ONLY or FOR UPDATE. RW indicates the cursor defaults to FOR UPDATE.)
SQL_DIALECT	CHAR (8)	TDM_CORE
SQL_ERROR_*sqlcode*	INT(2)	This item is not available by default; it is only present if added using the ADD SCFG command. If COLLECT_SQL_ERROR_INFO is Y, then 1 indicates that NOS should abend when *sqlcode* is returned.
SQL_MAX_STATEMENT_ CACHE	INT (8)	0
SQL_TXN_ISOLATION	CHAR (1)	1 indicates stable access.
SQL_UNSUPPORTED	CHAR (1)	E (indicates that any unsupported syntactical token is reported as an error and the statement fails)
SQL_WARNING_*sqlcode*	INT(2)	This item is not available by default; it is only present if added using the ADD SCFG command. If COLLECT_SQL_ERROR_INFO is Y, then 1 indicates that NOS should abend when *sqlcode* is returned.
START_TIME	TIMESTAMP	None
STMT_CACHED	CHAR (1)	None
STMT_NAME_CACHE	CHAR (1)	Y
STMT_ORIGIN	CHAR (1)	None
STMT_STATUS	CHAR (1)	None
STMT_TYPE	CHAR (1)	None

Table D-1. System Default Values Installed in ZNSSCFG (page 5 of 5)

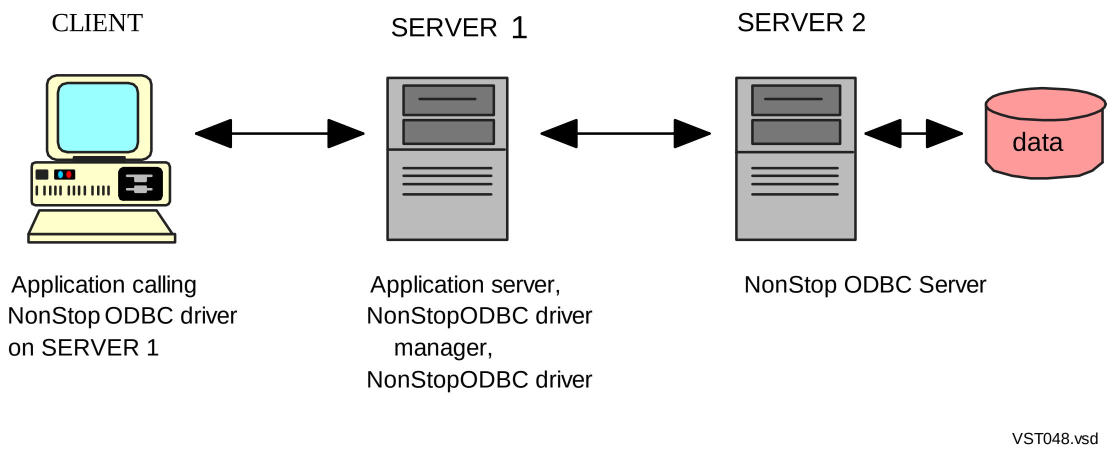
Column Name or Attribute (Stored in the ITEM Column)	**Table Data Type**	**System Default (Stored in the VALUE Column)**
SWAPVOL	CHAR (25)	" " (no default is assigned)
TRA_ERROR	CHAR (1)	N
TRA_LEVEL	VARCHAR (60)	" " (no default is assigned)
TRA_MODE_ON	CHAR (1)	N
TRA_NAME	VARCHAR (60)	" " (no default is assigned)

E

Changing Passwords in a Three-Tier Environment

A three-tier environment is a configuration of ODBC where the HP NonStop ODBC/MP driver is invoked by an application server on behalf of a user application. Figure E-1 shows a typical three-tier configuration.

Figure E-1. A Typical Three-Tier Configuration


CLIENT
SERVER 1
SERVER 2
data
Application calling NonStop ODBC driver on SERVER 1
Application server, NonStopODBC driver manager, NonStopODBC driver
NonStop ODBC Server
VST048.vsd

In the configuration shown, the HP NonStop ODBC Server cannot open a window on the CLIENT platform to prompt for a new password when SERVER 2 determines that the current user ID or password has changed.

The NonStop ODBC/MP driver has a facility that supports user change of password in three-tier environments by allowing an application server to do one of the following:

- Report detection of an expired password on the same platform where the NonStop ODBC/MP driver is being executed and allow the user to change the password.
- Provide its own secure paths by which the client can provide old and new passwords so that a change can be effected to the server.

The user or administrator of the middle tier activates this facility by setting the new “Suppress CHANGEPASSWORD View” attribute found on the NonStop ODBC driver configuration page. This page is displayed whenever the ODBC Datasource Administrator is invoked to add new datasource definitions or modify existing ones on the middle-tier server.

Possible values for the Suppress CHANGEPASSWORD View attribute are:.

NO	Specifies that detection of an expired password by the NonStop ODBC Server is reported to the user on the same platform where the NonStop ODBC/MP driver is being executed. The user is asked if they wish to change the password by using a popup menu.
YES	Specifies that detection of an expired password by the NonStop Server is returned as a SQLSTATE error or warning. The application server in the middle tier may use secure means to report and solicit a new password. The application server then creates a second connection to complete the change of password before proceeding with normal ODBC session activities.

This NonStop ODBC driver attribute is found in the ODBC.INI file, associated with one or more datasource definitions. The attribute name in the Windows NT registry is SQL_CHANGEPASSWORD_NO_GUI, and it takes values of YES and NO.

If a password error is returned, it is necessary for the change password operation to complete successfully before the client can successfully log in and execute SQL. If a password warning is returned, the current session is valid and statements can be executed.

To use this facility, the user configuration must also be configured for change password on the server where NonStop ODBC Server is running. The username alias configuration attribute CHANGE_PASSWORD_OPTION must have a value of 1 or 2. Refer to the description of the ADD ALIAS statement in Section 7, Managing Customized Catalogs, for information about setting this attribute.

F Creating Partitioned Tables

To create partitioned tables, follow these steps:

1. Create a Partition Overlay Specification (POS) template
2. Configure your environment
3. Create tables

Creating POS Templates

A template is composed of one or more NonStop SQL/MP DDL partition specifications. Each template is terminated using a semicolon (;) in the first column of the next line. The syntax is the same as defined for SQL/MP partition specifications, as follows:

```
[PRIMARY] KEY (column-list)
   PARTITION (partition-definition
   [, partition-definition. . .])
;


column-list:
    column-name [,column-name . . .]


partition-definition is defined as:

    partition-template-filename
    [ CATALOG catalog ]
    [ PHYSVOL volume-name ]
    [ EXTENT { size | ( pri-size [, sec-size ] ) } ]
    [ MAXEXTENTS size ]
    FIRST KEY { key-value | ( key-value[, key-value ...] ) }
```

partition-template-filename

specifies the template file name in the format $*volume*.SSSSSSSS.FFFFFFFF.

You must include the S and F placeholder characters after the volume name.

The ODBC partition specification is limited to a maximum of 4059 characters, including those allotted for the SQL statement.

Example Template

```
KEY (C1)
PARTITION ($volume1.SSSSSSSS.FFFFFFFF EXTENT 64 MAXEXTENTS 160 FIRST KEY -1000
     ,$volume2.SSSSSSSS.FFFFFFFF EXTENT 32 MAXEXTENTS 128 FIRST KEY 1000
     ,$volume3.SSSSSSSS.FFFFFFFF EXTENT 64 MAXEXTENTS 160 FIRST KEY 2222)
;
KEY (C1,C2)
PARTITION ($volume1.SSSSSSSS.FFFFFFFF CATALOG \N1.$DATA.SQL2CAT
                                     FIRST KEY (-50,"AAAAA")
,\N2.$volume1.SSSSSSSS.FFFFFFFF CATALOG \N2.$DATA.SQLCAT
                                    FIRST KEY (50, "MMMMM"))
;
```

Configuring for Partitioned Tables

Create a Partition File Segment

The ADDPOS statement adds partition overlay specifications (POS) to an existing partitioned table partition file. Any partition can be used, because the utility finds the primary partition from the file's header. It is most efficient to use the primary partition.

```
nosutil ADDPOS template-file-name
    [ [ $volume.]subvolume.]partition-table1
    [ [ [ $volume.]subvolume.]partition-tablen ... ]
```

The POS is appended to the specified file; one or more partition files can be appended in one statement. If *template-file-name* does not exist, the file is created.

If a POS is a duplicate of a previously added POS, the NonStop ODBC Server uses the first POS that matches the primary key list.

If an error occurs, the statement is aborted and the original *template-file-name* is unchanged.

Configure the ZNSSCFG File

If the ODBC system catalog is in $SYSTEM.SQL, use the following SQLCI statement to configure the ZNSSCFG file.

```
INSERT INTO $SYSTEM.SQL.ZNSSCFG (ITEM,VALUE)
                     VALUES ("PK_MAP_FNM",
                             "\N.$ODBC.POMAP.POMAP")
```

If the ODBC system catalog is in $DATA1.ODBCSYS, use the following SQLCI statement to configure the ZNSSCFG file.

```
INSERT INTO $DATA1.ODBCSYS.ZNSSCFG (ITEM,VALUE)
                     VALUES ("PK_MAP_FNM",
                             "\N.$ODBC.POMAP.POMAP")
```

Note that you must have read access rights to the POS template file, which is $ODBC.POMAP.POMAP in this example.

Example CREATE TABLE Statement

Execute the following CREATE TABLE statement from an ODBC client that is connected to ODBC service configured to use the previous example template.

```
CREATE TABLE db1.sql_odbc. T1  (C1 int, C2 char(5),
                                 C3 char(5),
                               PRIMARY KEY (C1,C3))
```

If TRACE is on, you can see the `CREATE TABLE` translated into the following SQL/MP statement:

```
CREATE TABLE  \N1.$ODBC.SV1.T1
       ( C1 INT NOT NULL HEADING “C1”
       , C2 CHAR(5) NOT NULL HEADING “C2”
       , C3 CHAR(5) NOT NULL HEADING “C3”
       ,PRIMARY KEY (C1,C3) )

    PARTITION ( \N1.$VOLUME1.SV1.T1 EXTENT 64 MAXEXTENTS 160
                          FIRST KEY -1000
              ,\N1.$VOLUME.SV1.T1 EXTENT 32 MAXEXTENTS 128
                         FIRST KEY 1000
              ,\N1.$VOLUME.SV1.T1 EXTENT 64 MAXEXTENTS 160
                        FIRST KEY 2222)

;
```

The partition specification for primary key C1 is used as the best subset match between the original CREATE. . .PRIMARY KEY and the templates.

The subvolume and filename are used to modify the partition specification template according to SQL/MP syntax.

Glossary

This glossary defines terms used in this manual. Because this manual discusses several products, the glossary indicates which of the following products or categories are associated with each term:

- NonStop ODBC Server
- NonStop SQL/MP
- ODBC
- SQL Server
- Both NonStop SQL/MP and ODBC/SQL Server
- Standard SQL
- HP
- Industry-standard term

aggregate function [standard SQL]**.** A function that generates a summary value from a group of values in a specified column. The aggregate functions in both HP NonStop SQL/MP and SQL Server are AVG, SUM, MIN, MAX, and COUNT.

See also row aggregate function [SQL Server], scalar aggregate function [ODBC/SQL Server], and vector aggregate [SQL Server].

alias [SQL Server]**.** A temporary name give to a table (in the FROM clause) when it is joined with itself in a self join. The temporary table names are then used to qualify the column names in the join.

In NonStop SQL/MP, aliases are called correlation names.

See also correlation name [NonStop SQL/MP].

application program interface (API) [industry-standard term]**.** The set of functions or procedures that permits user programs to communicate with a host operating system.

audited table [NonStop SQL/MP]**.** A table flagged for auditing by the Transaction Management Facility (TMF). TMF monitors all transactions against an audited table in preparation for transaction backout, autorollback, or rollforward recovery.

See also nonaudited table [NonStop SQL/MP] and Transaction Management Facility (TMF) subsystem [HP].

batch [SQL Server]**.** One or more TRANSACT-SQL statements terminated by an end-of-batch signal, which submits them to SQL Server for processing.

browse access [NonStop SQL/MP]**.** The access option for transaction consistency that provides immediate access to data, but the data could be inconsistent because it might be changing in an incomplete transaction. With this access, a process does not acquire a lock on the data it accesses and does not test for existing locks before reading data. Browse access can be specified only for reading data. Of the access options, this access provides the lowest consistency but the highest concurrency. In the HP NonStop ODBC Server, browse access is available only in pass-through mode.

See also browse mode [SQL Server], repeatable access [NonStop SQL/MP], and stable access [NonStop SQL/MP].

browse mode [SQL Server]**.** A method of viewing data while performing updates. The FOR BROWSE clause is used in a SELECT statement.

The NonStop ODBC Server does not support browse mode or the FOR BROWSE clause.

See also browse access [NonStop SQL/MP].

call-level interface (CLI) [standard SQL]**.** A library of function calls that support SQL statements. The ODBC interface is a CLI.

catalog [NonStop SQL/MP]**.** A set of NonStop SQL/MP tables containing the descriptions of objects such as tables, columns, indexes, views, files, and partitions.

See also catalog table [NonStop SQL/MP] and database [industry-standard term].

catalog table [NonStop SQL/MP]**.** One of the tables in a catalog; each table describes objects such as tables, columns, views, and indexes. Any authorized user can read a catalog table, but only a NonStop SQL/MP licensed process can update a catalog table.

CATALOGS table [NonStop SQL/MP]**.** A directory table in the system catalog that contains one entry for each catalog in that system. The table name is SQL.CATALOGS.

catalog utilities [NonStop ODBC Server]**.** Statements provided by the NonStop ODBC Server for maintaining customized catalogs. The utility statements can be run using the NonStop ODBC Server or can be entered from a TACL prompt.

client [industry-standard term]**.** The portion of a client/server architecture that provides application and user interface resources. Clients are usually workstations or PCs.

See also client/server architecture [industry-standard term].

client application [industry-standard term]**.** An application running on a client accessing data on a server. A client application is sometimes called a client.

See also client [industry-standard term] and workstation-based tool [industry-standard term].

client/server architecture [industry-standard term]**.** A computer architecture that divides work between a client and a server. The client provides application and user interface resources, and the server stores, retrieves, and protects data. Client/server architecture enables users to access shared data and resources. Clients and servers run on a local area network.

client, DBLIB/TSQL [NonStop ODBC Server]**.** A type of client that uses the Microsoft/Sybase TRANSACT-SQL (TSQL) dialect and makes DBLIB function calls. It is a client for SQL Server and the HP SQL Server Gateway product.

client, ODBC/CORE [NonStop ODBC Server]**.** A type of client that uses the ODBC CORE SQL dialect and makes ODBC CLI function calls. This is the basic ODBC client application intended for maximum interoperability and portability.

client, ODBC/TSQL [NonStop ODBC Server]**.** A type of client that uses the TRANSACT-SQL dialect, but makes ODBC CLI function calls. The expected use for this client is as part of a migration path for applications moving from DBLIB to ODBC.

clustered index [SQL Server]**.** An index in which the physical order and the logical (indexed) order is the same.

See also clustering key [NonStop SQL/MP].

clustering key [NonStop SQL/MP]**.** A group of columns that forms a non unique key to determine the physical row order or partitioning of a key-sequenced table. The columns of the clustering key together with the system-generated SYSKEY make up the physical primary key for the table.

See also clustered index [SQL Server].

column alias [SQL Server]**.** A column heading, specified in the SELECT statement, that replaces the default column heading (the column name).

See also column heading [NonStop SQL/MP].

column heading [NonStop SQL/MP]**.** A heading, specified in the SELECT statement, that replaces the default column heading (the column name).

See also column alias [SQL Server].

communication driver [industry-standard term]**.** A unit of library code that is needed, along with the HP NonStop ODBC/MP Driver, to use a particular communication transport protocol. A separate communication driver is needed for each vendor’s communication transport. With the NonStop ODBC Server product, HP supplies communication drivers for Microsoft, FTP, Novell, and Winsock TCP/IP.

connection [ODBC/SQL Server]**.** An instance of a specific client and a specific server communicating for a period of time (also called a session). A connection is initiated by the client, but can be terminated by either the client or the server.

constraint [NonStop SQL/MP]**.** A DDL statement that helps protect the integrity of data in a table by specifying rules that all values in particular columns of the table must satisfy.

See also rule [SQL Server].

control-of-flow language [standard SQL]**.** The statements that enable the user to control the flow of execution of statements. Control-of-flow statements include statements such as IF, ELSE, WHILE, and GOTO.

Most SQL Server control-of-flow statements are not supported by the NonStop ODBC Server.

core services [HP]**.** The portion of the operating system that consists of the low-level functions, including interprocess communication, I/O interface procedures, and memory, time, and process management.

correlation name [NonStop SQL/MP]**.** A simple name associated with a table (or view) in a single SQL statement and used to distinguish the table from another table with the same simple name, to qualify an ambiguous column reference, to distinguish separate uses of the same table, or to make the query shorter.

See also alias [SQL Server].

current database [NonStop ODBC Server]**.** The database most recently specified by the USE statement. When a user logs on to the NonStop ODBC Server, the *master* database is the current database.

current owner [NonStop ODBC Server]**.** The Guardian group name and username that the user specified when logging on to the NonStop ODBC Server.

customized catalog [NonStop ODBC Server]**.** A NonStop SQL/MP catalog available through the NonStop ODBC Server. A customized catalog contains NonStop ODBC Server mapping tables. All catalogs created using the NonStop ODBC Server are customized.

See also decustomized catalog [NonStop ODBC Server] and NonStop ODBC Server mapping tables [NonStop ODBC Server].

Data Control Language (DCL) [standard SQL]**.** The statements used to control process resources such as locks and cursors. DCL includes statements such as LOCK TABLE, UNLOCK TABLE, and FREE RESOURCES.

See also Data Definition Language (DDL) [standard SQL] and Data Manipulation Language (DML) [standard SQL].

Data Definition Language (DDL) [standard SQL]**.** The statements used to define, delete, or modify the catalog definition of an object. DDL includes statements such as CREATE DATABASE, CREATE TABLE, and DROP VIEW.

See also Data Control Language (DCL) [standard SQL] and Data Manipulation Language (DML) [standard SQL].

Data Manipulation Language (DML) [standard SQL]**.** The statements used to select, update, insert, or delete rows in tables. DML includes statements such as DELETE, INSERT, SELECT, and UPDATE.

See also Data Control Language (DCL) [standard SQL] and Data Definition Language (DDL) [standard SQL].

database [industry-standard term]**.** A set of data tables and other database objects organized and presented to serve a specific purpose.

See also catalog [NonStop SQL/MP].

database device [SQL Server]**.** A file in which databases and transaction logs are stored. A database device has both a physical name and a logical name.

database object. See object [standard SQL].

datepart [SQL Server]**.** A parameter used with a date function. Dateparts include year, month, day, week, hour, and minute.

The NonStop ODBC Server supports some of the SQL Server dateparts.

DBLIB. See DB-LIBRARY [ODBC/SQL Server].

DB-LIBRARY [ODBC/SQL Server]**.** A set of C routines and macros that enables an application program to interact with the SQL Server. DB-LIBRARY resides on the workstation with the application program.

DBLIB/TSQL client. See client [industry-standard term], DBLIB/TSQL client.

DCL. See Data Control Language (DCL) [standard SQL].

DDL. See Data Definition Language (DDL) [standard SQL].

decision-support tool [industry-standard term]**.** An application that enables users to query (and possibly update) data. The NonStop ODBC Server is a decision-support tool.

See also online transaction processing (OLTP) application [industry-standard term].

decustomized catalog [NonStop ODBC Server]**.** A NonStop SQL/MP catalog that was previously customized but has had the NonStop ODBC Server mapping tables removed and is no longer available through the NonStop ODBC Server.

See also customized catalog [NonStop ODBC Server].

default trace record [NonStop ODBC Server]**.** A set of trace flag values that are assigned to a user if the user has no trace logs set up and if no override trace record exists.

See also override trace record [NonStop ODBC Server] and trace flag [NonStop ODBC Server].

default trace value [NonStop ODBC Server]**.** A trace flag value in a default trace record.

See also default trace record [NonStop ODBC Server].

dependent object [standard SQL]**.** An object whose definition depends on a base or underlying object; for example, an index or protection view is dependent on a table.

device. See database device [SQL Server].

DLL. See dynamic link library (DLL) [ODBC].

DML. See Data Manipulation Language (DML) [standard SQL].

DOS workstation [industry-standard term]**.** An IBM personal computer or a compatible computer running DOS.

dynamic link library (DLL) [ODBC]**.** An executable module that lets Windows applications share code and resources. It contains procedures that applications can call to perform useful tasks.

escape clause [ODBC]**.** A specialized syntax convention that ODBC uses to permit inclusion of an extension to SQL in a CORE SQL statement.

exclusive lock [both NonStop SQL/MP and SQL Server]**.** In NonStop SQL/MP, the lock exclusion mode in which only the holder of the lock can access the locked items.

In SQL Server, a lock that prevents any other transaction from acquiring a lock until the original lock is released at the end of a transaction. SQL Server always applies an exclusive lock for update operations (INSERT, UPDATE, and DELETE).

See also shared lock [both NonStop SQL/MP and SQL Server].

file [NonStop SQL/MP]**.** The physical storage for a table or index. The creation of a table or index implicitly creates a file with the same name as the table or index.

file system [NonStop SQL/MP]**.** The portion of the operating system software that provides a standard interface between application processes and input/output devices or other processes.

global variable [SQL Server]**.** A system-defined variable that SQL Server updates on an ongoing basis. For example, @@ERROR contains the last error number generated by the system.

See also local variable [SQL Server] and variable [SQL Server].

Guardian [HP]**.** The original application program interface (API) to the HP NonStop Kernel operating system.

Guardian environment [HP]**.** The Guardian API and the HP NonStop tools and utilities.

Guardian name [HP]**.** A name formed according to the Guardian rules for naming disk files that can be used to identify a table, view, index, program, file, or partition. Process and device names are also Guardian names.

LAN. See local area network [industry-standard term].

local area network [industry-standard term]**.** A data communications system that enables PCs to have access to common data and peripherals. LANs typically consist of PCs with adapter cards, file servers, a network operating system, printers, and gateways to departmental or corporate computers.

local variable [SQL Server]**.** A variable declared with the DECLARE statement and assigned an initial value with the SELECT statement. Local variables are declared and assigned values within a batch.

See also global variable [SQL Server] and variable [SQL Server].

log table [NonStop ODBC Server]**.** An SQL table containing a trace log.

See also trace feature [NonStop ODBC Server] and trace log [NonStop ODBC Server].

login name [SQL Server]**.** The name a user specifies when logging on to SQL Server.

logon [NonStop ODBC Server]**.** A connection between a specific user and the NonStop ODBC Server. The NonStop ODBC Server must be started for a user to log on.

mapping tables [NonStop ODBC Server]**.** See NonStop ODBC Server mapping tables [NonStop ODBC Server].

master database [SQL Server]**.** The database that controls the user databases and the operation of SQL Server as a whole. The master database keeps track of such things as user accounts, ongoing processes, and system error messages.

NETBIOS (NETwork Basic Input/Output System) [industry-standard term]**.** An application programming interface for data exchange between clients and servers on a network (typically, a local area network).

NETLIB (NETwork LIBrary) [industry-standard term]**.** An interface within DBLIB that provides uniform access to transport protocols such as TCP/IP and NetBIOS. The term is also used to refer to a NETLIB driver.

See also NETLIB driver [industry-standard term].

NETLIB driver [industry-standard term]**.** A unit of library code that is needed along with DBLIB to use a particular transport protocol. A separate NETLIB driver is needed for each vendor’s transport.

node [HP]**.** One of the computer systems in a network.

See also system [HP].

nonaudited table [NonStop SQL/MP]**.** A table that is not audited by TMF and, therefore, is not protected from system failure by TMF autorollback.

See also audited table [NonStop SQL/MP] and Transaction Management Facility (TMF) subsystem [HP].

NonStop Kernel operating system [HP]**.** The operating system, which consists of the core and system services. The operating system does not include any application program interfaces.

NonStop ODBC Server. A HP product that lets programs developed for ODBC or SQL Server access databases using NonStop SQL/MP.

NonStop ODBC Server mapping tables [NonStop ODBC Server]**.** The NonStop SQL/MP tables, views, and indexes that the NonStop ODBC Server creates to customize a catalog (make it available through the NonStop ODBC Server). The mapping table names begin with the letter Z. Mapping tables exist on the subvolume of the NonStop SQL/MP system catalog and on the subvolume of each customized catalog.

See also customized catalog [NonStop ODBC Server].

NonStop ODBC Server object [NonStop ODBC Server]**.** An object created through the NonStop ODBC Server. The NonStop ODBC Server keeps track of whether objects are created through the NonStop ODBC Server or with NonStop SQL/MP. Objects created in pass-through mode are NonStop SQL/MP objects.

See also NonStop SQL/MP object [NonStop ODBC Server].

NonStop ODBC Server system catalog [NonStop ODBC Server]**.** A set of NonStop SQL/MP tables, views, and indexes used by NonStop ODBC Server to support the master database for NonStop ODBC Server clients. The NonStop ODBC Server system catalog is a NonStop ODBC Server user catalog plus additional tables.

NonStop ODBC Server user catalog [NonStop ODBC Server]**.** A set of NonStop SQL/MP tables, views, and indexes used by NonStop ODBC Server to support a database for NonStop ODBC Server clients.

NonStop SQL/MP [HP]**.** A relational database management system that runs on a HP NonStop server.

NonStop SQL/MP Conversational Interface [NonStop SQL/MP]**.** See SQLCI (SQL Conversational Interface). [NonStop SQL/MP].

NonStop SQL/MP object [NonStop ODBC Server]**.** An object created with NonStop SQL/MP. The NonStop ODBC Server keeps track of whether objects are created through the NonStop ODBC Server or with NonStop SQL/MP. Objects created in pass-through mode are NonStop SQL/MP objects.

See also NonStop ODBC Server object [NonStop ODBC Server].

NonStop SQL/MP simple name. See simple name [NonStop SQL/MP].

NOSUTIL [NonStop ODBC Server]**.** The NonStop ODBC Server catalog utility programs.

NSODBC Server [NonStop ODBC Server]**.** The server process component of the NonStop ODBC Server product.

object [standard SQL]**.** A database entity created, manipulated, or dropped by means of SQL statements.

In NonStop SQL/MP, objects are described in an SQL catalog and include tables, views, columns, partitions, constraints, indexes, and files. In SQL Server, objects are tables, columns, views, indexes, defaults, rules, triggers, and procedures.

The NonStop ODBC Server does not support these SQL Server objects: defaults, triggers, rules, and procedures.

ODBC (Open Database Connectivity). An API that allows applications to access data in database management systems using SQL as a standard for data access.

ODBC driver [ODBC]**.** A unit of library code that is needed for an ODBC application to communicate with a server. ODBC drivers are DLLs and are loaded on demand. HP supplies an ODBC driver as part of the NonStop ODBC Server product.

ODBC/CORE client. See client [industry-standard term].

ODBC/TSQL client. See client [industry-standard term].

online transaction processing (OLTP) application [industry-standard term]**.** An application in which many users can update data simultaneously, recording the changes in the database as they are entered. NonStop SQL/MP is designed for efficient OLTP.

See also decision-support tool [industry-standard term].

OS/2 workstation [industry-standard term]**.** An IBM personal computer or a compatible computer running MS OS/2, an operating system co-developed by Microsoft and IBM for personal computers based on the Intel 80286 and 80386 microprocessors.

override trace record [NonStop ODBC Server]**.** A set of trace flag values assigned to all users who log on to the NonStop ODBC Server. Override trace values take priority over the user’s own trace flag values.

See also default trace record [NonStop ODBC Server] and trace flag [NonStop ODBC Server].

override trace value [NonStop ODBC Server]**.** A trace flag value in an override trace record.

See also override trace record [NonStop ODBC Server].

owner [ODBC/SQL Server]**.** The login ID of the user who owns a database or object. Owner names are ODBC/SQL Server identifiers. The NonStop ODBC Server supports owner names in the format `group-name_user-name`, in which `group-name` and `user-name` correspond to a Guardian logon ID.

pass-through mode [NonStop ODBC Server]**.** A mode the NonStop ODBC Server provides for executing statements other than TRANSACT-SQL statements. Pass-through mode allows NonStop SQL/MP statements, catalog utility statements, and trace statements.

procedure [SQL Server]**.** See stored procedure [both ODBC and SQL Server] or system procedures [SQL Server].

profile table [NonStop ODBC Server]**.** A NonStop ODBC Server mapping table, ZNSPROF, used with the trace feature. The profile table records all log table names, trace flag values, override trace values, and default trace values.

See also NonStop ODBC Server mapping tables [NonStop ODBC Server] and trace feature [NonStop ODBC Server].

protection view [NonStop SQL/MP]**.** A view defined with the protection attribute. The view can be derived from a single table by taking either a projection of the columns of the table, a selection of the rows in the table, or both. A protection view provides a form of field level security because the view can be secured, updated, and read.

See also shorthand view [NonStop SQL/MP].

repeatable access [NonStop SQL/MP]**.** The access option for transaction consistency that guarantees that the data you access within a transaction cannot be changed by other users until the transaction ends. With this access, a process tests for existing locks on data before acquiring its own lock and holds the lock until the transaction completes. Of the NonStop SQL/MP access options, this access provides the highest consistency but the lowest concurrency.

See also browse access [NonStop SQL/MP] and stable access [NonStop SQL/MP].

row aggregate function [SQL Server]**.** A function that generates a new row for summary data when used with the COMPUTE BY clause in a SELECT statement. The SQL Server row aggregate functions are SUM, AVG, MIN, MAX, and COUNT.

The NonStop ODBC Server does not support the COMPUTE BY clause or row aggregate functions.

See also aggregate function [standard SQL].

rule [SQL Server]**.** A specification that controls what data can be entered into a particular column.

The NonStop ODBC Server does not support rules.

See also constraint [NonStop SQL/MP].

savepoint [SQL Server]**.** A marker that the user puts inside a user-defined transaction. When transactions are rolled back, they are rolled back only to the savepoint.

The NonStop ODBC Server does not support savepoints.

scalar aggregate function [ODBC/SQL Server]**.** An aggregate function that produces a single value from a SELECT statement that does not include a GROUP BY clause. This result is true whether the aggregate function is operating on all the rows in a table or on a subset of rows defined by a WHERE clause.

See also vector aggregate [SQL Server].

SCS (SQL Communications Subsystem) [NonStop ODBC Server]**.** A NonStop ODBC Server software component that runs on a NonStop server and implements the communications protocols for data transfer between a client on a PC or workstation and a NonStop ODBC Server process on a NonStop server.

select list [standard SQL]**.** The columns and expressions specified in the main clause of a SELECT statement.

server [industry-standard term]**.** The server in a client/server architecture. The server stores, retrieves, and protects data. Servers can be minicomputers, microcomputers, mainframes, workstations, or specifically developed machines. When running the NonStop ODBC Server, a HP system becomes a server.

See also client/server architecture [industry-standard term].

session. See connection [ODBC/SQL Server].

shared lock [both NonStop SQL/MP and SQL Server]**.** In NonStop SQL/MP, a lock exclusion mode that allows any number of processes to read the same data but prevents any process from writing to the locked unit or reading it with intent to rewrite.

In SQL Server, a lock created by nonupdate (read) operations. When a shared lock is set, other users can read the data concurrently, but no transaction can acquire an exclusive lock on the data until all the shared locks have been released.

See also exclusive lock [both NonStop SQL/MP and SQL Server].

shorthand view [NonStop SQL/MP]**.** A view that can be derived from one or more tables or views by joining tables or views, by selecting columns, by projecting rows, or by a combination of these; a shorthand view can be read but not updated or secured.

See also protection view [NonStop SQL/MP].

simple name [NonStop SQL/MP]**.** See SQL identifier [NonStop SQL/MP].

SQL (Structured Query Language) [industry-standard term]**.** A relational database language used to define, manipulate, and control databases. SQL is the standard language for relational database management systems.

SQL identifier [NonStop SQL/MP]**.** A name used for a column, constraint, cursor, statement, or correlation. These names are never qualified by subvolume, volume, or system names and cannot be SQL reserved words.

SQL Server. An SQL-based relational database management system. The server maintains the database and handles queries to and responses from the database. Through the server, client applications such as NExpert Object, Paradox, and SQL File can access the database.

SQL Server Gateway [HP]**.** A HP product available with the C30.00 RVU that provides connectivity between DBLIB clients on PCs or UNIX workstations to NonStop SQL/MP. NonStop ODBC Server supports HP SQL Server Gateway clients. HP SQL Server Gateway is superseded by the NonStop ODBC Server product on D20 and subsequent D-Series releases.

SQL Server identifier [SQL Server]**.** The name of a database object. An identifier can be from 1 to 30 characters long. The first character must be a letter, a pound symbol (#), or an underscore (_). The remaining characters can be letters, digits, or the symbols #, $, and _. (A table name beginning with the pound symbol (#) denotes a temporary table.) Embedded spaces are not allowed.

SQLCI (SQL Conversational Interface). [NonStop SQL/MP]**.** A line-oriented terminal interface that enables a user to enter SQL statements, format and run reports, and operate database utilities.

stable access [NonStop SQL/MP]**.** The access option for transaction consistency that guarantees that no other user can change data that is locked or indicated by the current position of a cursor. With this access, a process tests for existing locks before acquiring its own, then holds the locks on queried data only until the next row is read and holds the locks on modified data until the transaction finishes. Of the NonStop SQL/MP access options, this access provides a medium level of consistency and concurrency.

See also browse access [NonStop SQL/MP] and repeatable access [NonStop SQL/MP].

stand-alone mode [NonStop ODBC Server]**.** The mode required to enter a catalog utility statement at a TACL prompt.

See also catalog utilities [NonStop ODBC Server].

stored procedure [both ODBC and SQL Server]**.** A collection of SQL statements stored under a name, executed by invocation of that name. Stored procedures supplied by SQL Server are called system procedures.

system [HP]**.** A HP system that consists of from 2 to 16 processors on a dual communication path (DYNABUS)

A system in a network of systems is called a node.

system catalog [NonStop SQL/MP]**.** The catalog that describes the CATALOGS table. The system catalog resides in an SQL subvolume of each system ($SYSTEM.SQL, by default).

See also catalog [NonStop SQL/MP], CATALOGS table [NonStop SQL/MP], and user catalog [NonStop SQL/MP].

system-defined transaction [standard SQL]**.** A transaction initiated by the system to protect the consistency of the database.

See also transaction [standard SQL] and user-defined transaction [standard SQL].

system procedures [SQL Server]**.** Special stored procedures that SQL Server supplies to retrieve information from system tables or to update information in system tables.

The NonStop ODBC Server does not support system procedures.

See also system tables [SQL Server].

system services [HP]**.** The tasks performed on behalf of the user or user programs by the operating system, including formatting, process control, I/O support, performance measurement, NonStop process-pair support, standard security, and transaction management.

system tables [SQL Server]**.** The tables that compose the data dictionary. The system tables keep track of information about SQL Server as a whole and about each user database. The master database contains some system tables that are not in user databases.

TCP/IP (Transport Communication Protocol/Internet Protocol) [industry-standard term]. A set of data communications protocols at the transport (TCP) and network (IP) layers that conform to U.S. Department of Defense standards.

temporary table [SQL Server]**.** A table that exists only during the current work session. A temporary table disappears either when it is dropped or at the end of the session. You specify a temporary table by beginning the table name with a pound symbol (#).

The NonStop ODBC Server supports temporary tables.

TMF. See Transaction Management Facility (TMF) subsystem [HP].

trace feature [NonStop ODBC Server]**.** A utility the NonStop ODBC Server provides for recording information such as statements executed, NonStop SQL/MP statements generated, and output and error messages returned.

trace flag [NonStop ODBC Server]**.** The flags you set to control the trace information that the NonStop ODBC Server logs. The trace flags are MODE, LOG_TABLE, LOG_TERM, IN_STREAM, OUT_STREAM, NSSQL, TSQL, PROD, and ERROR.

trace log [NonStop ODBC Server]**.** The information the NonStop ODBC Server generates when you set trace flags. Trace logs are stored in SQL tables.

See also log table [NonStop ODBC Server].

trace statements [NonStop ODBC Server]**.** The statements provided with the trace feature. The trace statements are ALTER, CLEAR_LOG, DISPLAY_LOG, DROP_LOG, SAVE, SET, SHOW, and SHOW_LOGS.

transaction [standard SQL]**.** A series of changes to one or more database tables that transforms the data from one consistent state to another. TRANSACT-SQL supports transactions; CORE SQL does not.

See also system-defined transaction [standard SQL] and user-defined transaction [standard SQL].

Transaction Management Facility (TMF) subsystem [HP]**.** A product that helps to maintain database consistency and protects the database against damage due to system or media failures.

Transact-SQL (TSQL) [ODBC/SQL Server]**.** The SQL Server enhanced version of SQL.

The NonStop ODBC Server supports standard SQL; most enhancements are not supported.

TSQL. See Transact-SQL (TSQL) [ODBC/SQL Server].

unaudited table. See nonaudited table [NonStop SQL/MP].

user catalog [NonStop SQL/MP]**.** A catalog that is not a system catalog. A user catalog is created when you execute a CREATE CATALOG statement.

See also catalog [NonStop SQL/MP] and system catalog [NonStop SQL/MP].

user-defined transaction [standard SQL]**.** A transaction initiated with a BEGIN TRANSACTION statement (in NonStop SQL/MP, the statement is BEGIN WORK).

See also transaction [standard SQL] and system-defined transaction [standard SQL].

utility statement. See catalog utilities [NonStop ODBC Server].

variable [SQL Server]**.** An entity that can assume any of a set of values. Local variables are user-defined with the DECLARE statement; global variables are system-defined.

See also global variable [SQL Server] and local variable [SQL Server].

vector aggregate [SQL Server]**.** A value that results from using an aggregate function with a GROUP BY clause.

The NonStop ODBC Server does not support vector aggregates or the GROUP BY clause.

See also scalar aggregate function [ODBC/SQL Server] .

wild-card character [standard SQL]**.** A special character, the underscore (_) or the percent sign (%), used with the LIKE keyword to stand for one or any number of characters in pattern matching.

The NonStop ODBC Server supports both wild-card characters.

workstation [industry-standard term]**.** A microcomputer or minicomputer attached to other computers through a local-area network.

See also DOS workstation [industry-standard term] and OS/2 workstation [industry-standard term].

workstation-based tool [industry-standard term]**.** A client application such as Excel, Paradox, or SQL File.

See also client application [industry-standard term].

Index