



natural

Natural for SQL/DS

Version 4.1.2

Natural for SQL/DS

This document applies to Natural for SQL/DS Version 4.1.2 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

© Copyright Software AG 1979 - 2003.
All rights reserved.

The name Software AG and/or all Software AG product names are either trademarks or registered trademarks of Software AG. Other company and product names mentioned herein may be trademarks of their respective owners.

Table of Contents

Natural for SQL/DS - Overview	1
Natural for SQL/DS - Overview	1
General Information	2
General Information	2
Accessing an SQL/DS Table	2
Integration with Predict	2
Natural System Messages Related to SQL/DS	3
Installing Natural for SQL/DS	4
Installing Natural for SQL/DS	4
Installation Jobs	4
Using System Maintenance Aid	4
Prerequisites	4
Installation under CMS	4
Installation Tape	4
Preparing the Installation	5
Installation Procedure	5
Installation under VSE/ESA	7
Installation Tape	8
Copying the Tape Contents to Disk	8
Installation Procedure	9
Installation Verification	11
Prepare your SQL/DS Environment	11
Online Verification Methods	11
Sample Batch Verification Job - VSE/ESA only	12
Natural Parameter Modification for SQL/DS	12
DB2SIZE Parameter	13
NTDB Macro	13
Performance Considerations for the DB2SIZE Parameter	13
Parameter Module NDBPARM	14
BTIGN - Ignore Error after late BACKOUT TRANSACTION	15
CONVERS - Allows Conversational Mode under CICS	15
DELIMID	15
MAXLOOP - Maximum Number of Nested Database Loops	16
REFRESH	16
RWRDONL	16
STATDYN - Allow Static to Dynamic Switch	17
Database Management	18
Database Management	18
SYSSQL Utility	18
Fixed Mode	19
Free Mode	30
Natural System Commands for SQL/DS	33
LISTSQL Command	33
SQLERR Command	37
LISTDBRM Command	38
DDM Generation	40
DDM Generation	40
Natural Data Definition Module - DDM	40
SQL Services	40
Select SQL Table from a List	40
Generate DDM from an SQL Table	41
List Columns of an SQL Table	44

Dynamic and Static SQL Support	45
Dynamic and Static SQL Support	45
General Information	46
Internal Handling of Dynamic Statements	47
NDBIOMO	47
Statement Table	47
Processing of SQL Statements Issued by Natural	48
Preparing Natural Programs for Static Execution	49
Creating a Static SQL/DS Package under VSE/ESA	49
Generation Procedure - CMD CREATE Command	50
Modification Procedure - CMD MODIFY Command	53
Assembler/Natural Cross-References	54
Execution of Natural in Static Mode	55
Static SQL with Natural Security	55
Mixed Dynamic/Static Mode	55
Messages and Codes	56
Statements and System Variables	59
Statements and System Variables	59
Natural DML Statements	59
BACKOUT TRANSACTION	60
DELETE	60
END TRANSACTION	61
FIND	61
GET	63
HISTOGRAM	63
READ	64
STORE	65
UPDATE	66
Natural SQL Statements	69
Common Syntactical Items	69
COMMIT	72
DELETE	72
INSERT	72
PROCESS SQL	73
ROLLBACK	73
SELECT	74
UPDATE	74
Natural System Variables	75
*ISN	75
*NUMBER	75
Error Handling	75
Interface Subprograms	77
Interface Subprograms	77
Natural Subprograms	77
The NDBDBRM Subprogram	78
The NDBDBR2 Subprogram	79
The NDBERR Subprogram	80
The NDBISQL Subprogram	80
The NDBNOERR Subprogram	82
The NDBNROW Subprogram	83
The NDBSTMP Subprogram	84
The DB2SERV Interface	85
Function "D"	85
Function "U"	87
Environment-Specific Considerations	88
Environment-Specific Considerations	88

Natural for SQL/DS under CICS 88
Natural for SQL/DS in VSE/ESA Batch Mode 88

Natural for SQL/DS - Overview

This documentation describes the various aspects of Natural when used in an SQL/DS environment.

Note:

IBM refers to SQL/DS as DB2 Server for VSE & VM.

Related Documentation

For information on logging SQL statements contained in a Natural program, refer to the DBLOG Utility documentation in the section Debugging and Monitoring.

- General Information Information on how to access SQL/DS tables, on the integration with Software AG's Data Dictionary Predict, and on error messages related to SQL/DS.
- Installing Natural for SQL/DS Installation of the Natural interface to SQL/DS and description of the Natural for SQL/DS parameter module.
- Database Management Maintenance of SQL/DS tables and other SQL/DS objects with the SYSSQL utility; Natural system commands for SQL/DS.
- DDM Generation Generation of Natural data definition modules (DDMs) by using the SQL Services function of the Natural utility SYSDDM.
- Dynamic and Static SQL Support Internal handling of dynamic statements, creation and execution of static DB access modules (SQL/DS packages) in the various supported environments, mixed dynamic/static mode.
- Statements and System Variables Special considerations on Natural DML statements, Natural SQL statements, Natural system variables, and Natural for SQL/DS error handling.
- Interface Subprograms Several Natural and non-Natural subprograms to be used for various purposes.
- Environment-Specific Considerations Special considerations on the various environments supported by Natural for SQL/DS.

General Information

With the Natural interface to SQL/DS, a Natural user can access data in an SQL/DS database. Natural for SQL/DS is supported in CICS and batch environments under VSE/ESA.

In general, there is no difference between using Natural with SQL/DS and using it with Adabas, DB2 or DL/I. The Natural interface to SQL/DS allows Natural programs to access SQL/DS data by using the same Natural DML statements that are available for Adabas, DB2 and DL/I. Therefore, programs written for SQL/DS tables can also be used to access Adabas, DB2 or DL/I databases. In addition, Natural SQL statements are available.

All operations requiring interaction with SQL/DS are performed by the Natural interface module.

This section covers the following topics:

- Accessing an SQL/DS Table
 - Integration with Predict
 - Natural System Messages Related to SQL/DS
-

Accessing an SQL/DS Table

To be able to access an SQL/DS table with a Natural program, the following steps must be taken:

1. Use the SYSSQL utility to define an SQL/DS table.
2. Use Predict or the SQL Services function of the Natural utility SYSDDM to create a Natural DDM of the defined SQL/DS table.
3. Once you have defined a DDM for an SQL/DS table, you can access the data stored in this table by using a Natural program.

The Natural interface to SQL/DS translates the statements of a Natural program into SQL statements.

Natural automatically provides for the preparation and execution of each statement. In dynamic mode, a statement is only prepared once (if possible) and can then be executed several times. For this purpose, Natural internally maintains a table of all prepared statements.

Almost the full range of possibilities offered by the Natural programming language can be used for the development of Natural applications which access SQL/DS tables. For a number of Natural DML statements, however, there are certain restrictions and differences as far as their use with SQL/DS is concerned; see Natural DML Statements. In the Natural Statements documentation, you can find notes on Natural usage with SQL/DS in the descriptions of the statements concerned.

As there is no SQL/DS equivalent to Adabas ISNs (Internal Sequence Numbers), any Natural features which use ISNs are not available when accessing SQL/DS tables with Natural.

For SQL-eligible databases, in addition to the Natural DML statements, Natural provides SQL statements; see Natural SQL Statements. In the Natural Statements documentation you can find a detailed description of these statements.

Integration with Predict

As Predict supports SQL/DS, direct access to the SQL/DS catalog is possible via Predict, and information from the SQL/DS catalog can be transferred to the Predict dictionary to be integrated with data definitions for other environments.

SQL/DS databases, tables and views can be incorporated and compared, new SQL/DS tables and views can be generated and Natural DDMs can be generated and compared. All SQL/DS-specific data types and the referential integrity of SQL/DS are supported. See the relevant Predict documentation for details.

In addition, Predict active references support static SQL for SQL/DS.

Natural System Messages Related to SQL/DS

The message number ranges of Natural system messages related to SQL/DS are 3700-3749 4750 - 4799, 6700 - 6799 and 7386-7395.

Installing Natural for SQL/DS

This section describes step by step how to install the Natural interface to SQL/DS (in the remainder of this section also referred to as NSQ).

This section covers the following topics:

- Installation Jobs
 - Using System Maintenance Aid
 - Prerequisites
 - Installation under CMS
 - Installation under VSE/ESA
 - Installation Verification
 - Natural Parameter Modification for SQL/DS
 - Parameter Module NDBPARM
-

Installation Jobs

The installation of Software AG products is performed by installation "jobs". These jobs are either created "manually" or generated by System Maintenance Aid (SMA).

For each step of the installation procedures described later in this section, the job number of a job performing the respective task is indicated. This job number refers to an installation job generated by SMA. If you are not using SMA, an example job of the same number is provided in the job library on the NSQ installation tape; you must adapt this example job to your requirements. That the job numbers on the tape are preceded by a product code (for example, NSQI070).

Using System Maintenance Aid

For information on using Software AG's System Maintenance Aid for the installation process, refer to the System Maintenance Aid documentation.

Prerequisites

- Base Natural must be installed first; you cannot install Natural and Natural for SQL/DS at the same time.
- The Software AG Editor must be installed (as described in the Natural Installation Guide for Mainframes).

Further product/version dependencies are specified under Natural and Other Software AG Products and Operating/Teleprocessing Systems Required in the current Natural Release Notes for Mainframes.

Installation under CMS

This section only applies to the installation of NSQ under CMS.

Installation Tape

The installation tape was created under OS/390; it has standard OS/390 labels and headers. It contains the datasets listed in the table below. The sequence of the datasets is shown in the Report of Tape Creation which accompanies the installation tape.

Dataset Name	Contents
NSQ nnn .TAPE	NSQ source modules, load modules and installation EXECs. This dataset is in TAPE DUMP format and must be loaded onto the installation minidisk.
NSQ nnn .INPL	NSQ utility programs in INPL format.
NSQ nnn .ERRN	NSQ error messages.

The notation nnn in dataset names represents the version number of the product.

Copying the Tape Contents to Disk

The tape file NSQ nnn .TAPE was created with the CMS TAPE DUMP facility. Load the contents of the tape to your A-disk. The free space should be at least 450 4-KB blocks; for example, 3 cylinders on 3350 or 3380 disks.

Ask the system operator to attach a tape drive to your virtual machine at address X'181' and mount the NSQ installation tape.

To position the tape for the TAPE LOAD command, calculate the number of tape marks as follows: If the sequence number of NSQ nnn .TAPE - as shown by the Report of Tape Creation - is n , you must position over $3n-2$ tape marks; that is, FSF 1 for the first dataset, FSF 4 for the second, etc.

Position the tape by issuing the CMS command:

```
TAPE FSF fsfs
```

where *fsfs* is calculated as described above.

Load the NSQ installation material by issuing the CMS command:

```
TAPE LOAD * * A
```

You may wish to keep the tape drive attached to your virtual machine, because the tape is still needed in Step 7 of the installation procedure.

Preparing the Installation

Perform the following steps to prepare the installation of NSQ:

1. Ensure that the required SQL/DS database machine is activated in multiple-user mode and that the user machine for this installation is properly configured and initialized to access the SQL/DS database machine.
2. All precompilations as well as NSQ itself take advantage of the implicit CONNECT mechanism provided by VM. Therefore, ensure that the VM user ID is authorized for SQL/DS.
3. Ensure that your user machine has access to the following minidisks:
the SQL/DS production minidisk,
the Natural installation minidisk.
4. Ensure that the Adabas environment for your user machine is set up.

Concerning the following installation steps, also refer to the section Installing Natural under VM/CMS in the Natural Installation Guide for Mainframes.

Installation Procedure

Perform the following steps to install NSQ:

Step 1: Generate the NSQ I/O module NDBIOMO

Generate NDBIOMO by using the command:

```
GENIOMO SQL/DS n
```

GENIOMO generates the assembly source for NDBIOMO from the existing source NDBIOTM. It prompts you for the Natural/CMS batch module and invokes the Natural program NDBGENI, which is loaded with INPL during the base Natural installation.

GENIOMO is invoked with the following two parameters:

- the DB-environment parameter, which must be set to SQL/DS,
- the parameter "n" to specify the number of statements for dynamic access; the default value is 10.

NDBIOMO performs the dynamic access to SQL/DS and contains all necessary EXEC SQL statements. In addition, it contains some special SQL statements which cannot be executed in dynamic mode.

An output report is created by this job and should be checked for successful completion. In addition, a condition code of 0 indicates normal completion.

Step 2: Precompile and assemble NDBIOMO

Precompile and assemble NDBIOMO using the command:

```
NDBIOMO
```

Note:

Since no precompiler options are specified, the default SQL/DS isolation level "Repeatable Read" may lead to locking problems, because all SQL/DS locks are held until the end of the transaction. Thus, depending on your application, it may be necessary to specify a different isolation level.

Step 3: Modify and assemble the NSQ parameter module NDBPARAM

Assemble NDBPARAM using the command:

```
NDBPARAM
```

The NSQ parameter module contains the macro NDBPRM, which contains parameters specific to the Natural interface to SQL/DS.

You can generally use the default values for all parameters. Modify only the values of the parameters whose default values do not suit your requirements.

The individual parameters are described in the section Parameter Module NDBPARAM.

Step 4: Modify NATPARAM

Adapt your Natural parameter module NATPARAM by adding parameters specific to Natural for SQL/DS as described in the section Natural Parameter Modification for SQL/DS.

Step 5: Modify NAT\$LOAD LOADLIST

Edit the member NAT\$LOAD EXEC provided on the Natural/CMS installation tape and add the following line to the existing LOADLIST statements:

```
LOADLIST = LOADLIST 'NDBNUC NDBNSQ NDBPARM NDBIOMO ARIRVSTC'
```

Step 6: Generate a Natural module

Generate the Natural/CMS load module using the command:

```
NATBLDM
```

NATBLDM is provided on the Natural CMS/installation tape and prompts you for the name of the Natural nucleus and generates the executable Natural module.

Step 7: Load Natural objects and error messages into system file

In this step, the NSQ system programs, maps and DDMs (dataset NSQ nnn .INPL) and the NSQ error messages file (dataset NSQ nnn .ERRN) are loaded into the Natural system file.

If the tape drive used when copying the contents of the installation tape to disk was detached from your virtual machine, ask the system operator to attach a tape drive to your virtual machine at address X'181' and mount the Natural installation tape.

Issue the following command:

```
NSQINPL
```

You are prompted for the name of the command to invoke Natural. Enter the name of the Natural module generated in Step 6.

NSQINPL then positions the tape and loads the Natural objects and error messages.

The INPL job loads objects into the libraries SYSDDM, SYSTEM and SYSSQL.

The ERRLODUS job loads error messages into the library SYSERR.

The NSQ system programs and error messages **must** be loaded into the Natural 3.1. FNAT system file



Warning:

Ensure that your newly created SYSSQL library contains all necessary Predict interface programs, which are loaded into SYSSQL when installing Predict (see the relevant Predict documentation).

Installation under VSE/ESA

Under VSE/ESA, Natural for SQL/DS basically consists of two parts:

1. An environment-independent nucleus, which can be linked to a shared Natural nucleus and loaded in the shared virtual area (SVA) of the operating system.
2. Environment-dependent components, which must be linked to the appropriate Natural environment-dependent interface.

This section covers the following topics:

- Installation Tape
- Installation Procedure

Installation Tape

The installation tape contains the datasets listed in the table below. The sequence of the datasets is shown in the Report of Tape Creation which accompanies the installation tape.

Dataset Name	Contents
NSQ nnn .LIBR	LIBR backup file.
NSQ nnn .INPL	NSQ utility programs in INPL format.
NSQ nnn .ERRN	NSQ error messages.

The notation nnn in dataset names represents the version number of the product.

Copying the Tape Contents to Disk

If you are using System Maintenance Aid (SMA), refer to the SMA documentation (included on the current edition of the Natural documentation CD).

If you are **not** using SMA, follow the instructions below.

This section explains how to:

- Copy data set COPYTAPE.JOB from tape to library.
- Modify this member to conform with your local naming conventions.

The JCL in this member is then used to copy all data sets from tape to disk.

If the datasets for more than one product are delivered on the tape, the member COPYTAPE.JOB contains the JCL to unload the datasets for all delivered products from the tape to your disk, except the datasets that you can directly install from tape, for example, Natural INPL objects.

After that, you will have to perform the individual install procedure for each component.

Step 1 - Copy data set COPYTAPE.JOB from tape to disk

The data set COPYTAPE.JOB (file 5) contains the JCL to unload all other existing data sets from tape to disk. To unload COPYTAPE.JOB, use the following sample JCL:

```
* $$ JOB JNM=LIBRCAT,CLASS=0,                                     +
* $$ DISP=D,LDEST=(*,UID),SYSID=1
* $$ LST CLASS=A,DISP=D
// JOB LIBRCAT
* *****
*       CATALOG COPYTAPE.JOB TO LIBRARY
* *****
// ASSGN SYS004, $nnn$                                            <----- tape address
// MTC REW,SYS004
// MTC FSF,SYS004,4
ASSGN SYSIPT,SYS004
// TLBL IJSYSIN,'COPYTAPE.JOB'
// EXEC LIBR,PARM='MSHP; ACC S=lib.sublib'                       <----- for catalog
/*
// MTC REW,SYS004
ASSGN SYSIPT,FEC
/*
/&
* $$ EOJ
```

Where:

NNN is the tape address

lib.sublib is the library and sublibrary of the catalog

Step 2 - Modify COPYTAPE.JOB

Modify COPYTAPE.JOB to conform with your local naming conventions and set the disk space parameters before submitting this job:

Step 3 - Submit COPYTAPE.JOB

Submit COPYTAPE.JOB to unload all other data sets from the tape to your disk.

Installation Procedure

The following steps describe the procedure for installing the components of NSQ.

Step 1: Generate the NSQ I/O Module NDBIOMO - Job I055, Step 1600

By executing a standard Natural batch job, this step generates the assembly source for NDBIOMO from the member NDBIOTM.

This batch job invokes the Natural program NDBGENI, which is loaded INPL during the base Natural installation. NDBGENI contains the following two parameters, which can be modified to meet your specific requirements:

- the DB-environment parameter, which must be set to SQL/DS,
- the parameter to specify the number of statements for dynamic access.

NDBIOMO performs the dynamic access to SQL/DS and contains all necessary EXEC SQL statements (see further information on NDBIOMO in the section Internal Handling of Dynamic Statements). In addition, it contains some special SQL statements which cannot be executed in dynamic mode.

An output report is created by this job and should be checked for successful completion. In addition, a condition code of 0 indicates normal completion.

Step 2: Precompile and Assemble NDBIOMO - Job I055, Steps 1610 and 1620

Precompile (using the SQL precompiler) and assemble NDBIOMO. Ensure that an appropriate SQL/DS user ID and password is specified for precompiling.

Note:

Since no precompiler options are specified, the default SQL/DS isolation level "Repeatable Read" may lead to locking problems, because all SQL/DS locks are held until the end of the transaction. Thus, depending on your application, it may be necessary to specify a different isolation level.

Step 3: Modify and Assemble the NSQ Parameter Module NDBPRM - Job I055, Step 1640

The NSQ parameter module contains the macro NDBPRM with parameters specific to the Natural interface to SQL/DS.

You can generally use the default values for all parameters. Modify only the values of the parameters whose default values do not suit your requirements.

The individual parameters are described in the section Parameter Module NDBPARAM.

Step 4: Modify and Reassemble NATPARAM

Adapt your Natural parameter module NATPARAM by adding parameters specific to Natural for SQL/DS and reassemble NATPARAM.

Step 5: Relink your Natural Nucleus

Modify the JCL used to link your Natural nucleus by adding the following INCLUDE cards and the corresponding DLBL statements:

INCLUDE NDBNUC	Environment-independent NSQ nucleus
INCLUDE NDBNSQ	Environment-independent SQL/DS interface
INCLUDE NDBPARAM	NSQ parameter module created in Step 3
INCLUDE NDBIOMO	NSQ I/O module created in Step 1
INCLUDE xxxxxx	Environment-dependent SQL/DS interface (see below)

Depending on your environment(s), INCLUDE the appropriate environment-specific language interface "xxxxxxx" as shown in the following table:

Interface	Environment
ARIPRDID	In batch mode
ARIRRTED	Under CICS

Note:

If you want to use NSQ in both environments, repeat this step for each of these environments.

Instead of link-editing your Natural nucleus in the way described above, you have the following alternatives:

1. If you use a shared Natural nucleus, only include NDBNUC and NDBNSQ in the link-edit of this nucleus. All other modules must be included in the link-edit of your Natural environment-dependent nucleus.
2. Remove NDBNUC and NDBNSQ from the link-edit of the Natural nucleus and link-edit them as a separate module with the mandatory *entry* name NATGWDB2. The *name* of the resulting phase is arbitrary. However, if you use a name different from NATGWDB2, this name must be specified as an alias name in an NTALIAS macro entry of the Natural parameter module. This way of link-editing only applies if the Natural Resolve CSTATIC Addresses feature (RCA) is used.
3. Include all modules in the link-edit job of a separate Natural parameter module with the mandatory *entry* name CMPRMTB. The *name* of the resulting phase is arbitrary. This way of link-editing only applies if an alternative parameter module (PARM profile parameter) is used.
4. If link-editing is done in this way, you can install NDB without having to modify your Natural nucleus or driver.

If link-editing is done according to number 2. or 3., the following applies:

Under CICS:

the resulting module must be defined via a PPT entry or RDO:

```
DFHPPT TYPE=ENTRY , PROGRAM=module-name , PGMLANG=ASSEMBLER
```

Step 6: Load Natural Objects Into System File - Job I061, Step 1600

In this step, the NSQ system programs, maps and DDMs are loaded into the Natural system files. The INPL job loads objects into the libraries SYSDDM, SYSTEM and SYSSQL.

The NSQ system programs **must** be loaded into the Natural 3.1 FNAT system file.



Warning:

Ensure that your newly created SYSSQL library contains all necessary Predict interface programs, which are loaded into SYSSQL when installing Predict (see the relevant Predict documentation).

Step 7: Load Natural Error Messages into System File - Job I061, Step 1620

This step executes a batch Natural job that runs an error load program using the NSQ nmn .ERRN dataset as input. The ERRLODUS job loads error messages into the library SYSERR on the FNAT system file.

The NSQ error messages **must** be loaded into the Natural FNAT system file.

Installation Verification

This section covers the following topics:

- Prepare your SQL/DS Environment
- Online Verification Methods
- Sample Batch Verification Job - VSE/ESA only

Prepare your SQL/DS Environment

As all dynamic access to SQL/DS is performed by NDBIOMO, all NSQ users must have RUN privilege on NDBIOMO.

Online Verification Methods

To verify the installation of the Natural interface to SQL/DS online, you can use either of the following methods:

- SQL Services
- DEM* Sample Programs

SQL Services

Perform the following steps to verify and check the installation of NSQ using the SQL Services of the Natural utility SYSDDM.

1. Invoke Natural.
2. Invoke SYSDDM.
On the SYSDDM main menu enter function code "B" to invoke the SQL Services function.
Enter function code "S" and specify SQL system "SQL/DS" to select all SQL/DS tables.
The communication between Natural and SQL/DS works if all existing SQL/DS tables are displayed.
3. For one of the tables, generate a Natural DDM as described in the section Generate DDM from an SQL Table.
To enable SYSDDM to generate a DDM, the Natural administrator requires access to the following SQL/DS tables:

SYSTEM.SYSCATALOG
SYSTEM.SYSCOLUMNS
SYSTEM.SYSINDEXES
SYSTEM.SYSVIEWS
SYSTEM.SYSSYNONYMS
SYSTEM.SYSUSAGE

4. After you have generated a DDM, access the corresponding SQL/DS table with a simple Natural program:

Example:

```
FIND view-name WITH field = value
  DISPLAY field
  LOOP
  END
```

5. If you receive the message SYSFUL 3700, enter the command SQLERR to display the corresponding SQL return code. See the description of the SQLERR command.

DEM2* Sample Programs

To verify and test your installation you can also use the sample programs DEM2* in the library SYSSQL provided on the installation tape.

Using these sample programs, you can create an SQL/DS table using DEM2CREA and create the corresponding DDM via SYSDDM. You can then store data in the created table using DEM2STOR and retrieve data from the table using DEM2FIND or DEM2SEL. You can also drop the table using program DEM2DROP.

Sample Batch Verification Job - VSE/ESA only

To verify the installation of the Natural interface to SQL/DS, a sample batch verification job (Job I065) is provided. This step contains sample JCL and sample programs to test Natural with NSQ in batch mode.

The sample program DEM2CONN performs the connection to the database, which is required before you can run a Natural program that accesses SQL/DS. DEM2CONN calls the DB2SERV module with function code "U" which in turn calls the database connect services.

Sample program DEM2JOIN performs a JOIN combining information from SQL/DS SYSTEM.SYSDSPACE and SYSTEM.SYSCATALOG.

Natural Parameter Modification for SQL/DS

This section covers the following topics:

- DB2SIZE Parameter
- NTDB Macro
- Performance Considerations for the DB2SIZE Parameter

DB2SIZE Parameter

Add the following Natural profile parameter to your NATPARM module:

DB2SIZE=*nn*

The DB2SIZE parameter can also be specified dynamically. It indicates the size of the SQL/DS buffer area, which should be set to at least 6 KB.

The setting of DB2SIZE can be calculated according to the following formula:

$$((808 + n1 * 40 + n2 * 100) + 1023) / 1024 \text{ KB}$$

The variables *n1* and *n2* correspond to:

<i>n1</i>	the number of statements for dynamic access as specified as the second parameter in job I055, step 1600 (under VSE/ESA).
<i>n2</i>	the maximum number of nested database loops as specified with the MAXLOOP parameter in NDBPARM.

Note:

Ensure that you have also added the Natural parameters required for the Software AG Editor (see Installing the Software AG Editor in the Natural Installation Guide for Mainframes).

Since DB2SIZE applies to Natural for SQL/DS and Natural for DB2, it should be set to the maximum value if you run more than one of these environments.

NTDB Macro

Add an NTDB macro for database type SQL specifying the list of logical database numbers that relate to SQL/DS tables. All Natural DDMs that refer to an SQL/DS table must be cataloged with a DBID from this list.

DBIDs can be any number from 1 to 254; a maximum of 254 entries can be specified. For most user environments, one entry is sufficient.

Note:

Ensure that all NSQ DDMs used when cataloging a given program have a valid SQL/DS DBID. Also ensure that the DBIDs selected in the NTDB macro for SQL/DS do not conflict with DBIDs selected for other database systems.

The DBID for SQL/DS used when cataloging a Natural program does not have to be in the NTDB list of DBIDs used when executing this program. Therefore, when executing existing Natural programs, DBID 250 is not mandatory.

Two sample NTDB macros follow:

```
NTDB SQL,250
NTDB SQL,(200,250,251)
```

Performance Considerations for the DB2SIZE Parameter

During execution of an SQL statement, storage is allocated dynamically to build the SQLDA for passing the host variables to SQL/DS.

In previous Natural for SQL/DS versions, this storage was always obtained from the TP monitor or operating system. For performance reasons, it is now first attempted to meet the storage requirements by free space in the Natural for SQL/DS buffer (DB2SIZE). Only if there is not enough space available in this buffer, the TP monitor or operating system is invoked.

To take advantage of this performance enhancement, you must specify your DB2SIZE larger than calculated according to the formula. The additional storage requirements (in bytes) can be calculated as follows:

- With sending fields:
 $64 + n * 56$
 where "n" is the number of sending fields in an SQL statement.
 The storage is freed immediately after the execution of the SQL statement.
- With receiving fields (that is, with variables of the INTO list of a SELECT statement):
 $64 + n * 56 + 24 + n * 2$
 where "n" is the number of receiving fields in an SQL statement.
 The storage remains allocated until the loop is terminated.

Example:

If you use the default value 10 for both variables (*n1* and *n2*), the calculated DB2SIZE will be 2200 bytes. However, if you specify a DB2SIZE of 20 KB, the available space for dynamically allocated storage will be 18272 bytes, which means enough space for up to either 325 sending fields or 313 receiving fields.

As space for receiving fields remains allocated until a database loop is terminated, the number of fields that can be used inside such a loop is reduced accordingly: for example, if you retrieve 200 fields, you can update about 110 fields inside the loop.

Note:

When using VARCHAR fields (that is, fields with either an accompanying L@ field in the Natural view or an explicit LINDICATOR clause), additional storage is allocated dynamically if the L@ or LINDICATOR field is not specified directly in front of the corresponding base field. Therefore, always specify these fields in front of their base fields.

Parameter Module NDBPARM

The NDBPARM source module contains Natural parameters specific to an SQL/DS environment. The parameter default values can be modified to meet site-specific requirements (see Step 3 of the installation procedures).

The individual parameters are described below. Their values cannot be dynamically overwritten.

NDBPARM contains the following parameters:

Parameter	Function
BTIGN	Ignores errors which result from BACKOUT TRANSACTION statements that are issued too late.
CONVERS	Allows conversational mode under CICS.
DELIMID	Escape character for delimited identifiers.
MAXLOOP	Specifies the maximum number of nested program loops.
REFRESH	Refresh connection to DB2 Server.
RWRDONL	Generate delimited identifiers for reserved words only.
STATDYN	Allows dynamic execution of statically generated SQL statements if the static execution returns an error.

BTIGN - Ignore Error after late BACKOUT TRANSACTION

This parameter is relevant in CICS environments only.

This parameter is used to ignore the error which occurs after a BACKOUT TRANSACTION statement that came too late to backout the current transaction, because of an implicit Syncpoint issued by the TP monitor.

Possible Value	Default Value	Explanation
ON	ON	The error after a late BACKOUT TRANSACTION is ignored.
OFF		The error after a late BACKOUT TRANSACTION is not ignored.

CONVERS - Allows Conversational Mode under CICS

This parameter is used to allow conversational mode in CICS environments.

Possible value	Default value	Explanation
ON	ON	Conversational mode is allowed.
OFF		Conversational mode is not allowed.

If this parameter is set to OFF, you cannot continue database loops across terminal I/Os; if so, the SQL codes -501, 504, 507, 514 or 518 may occur.

If you use the SYSDDM SQL services in a CICS environment, specify CONVERS=ON, otherwise the above mentioned errors could occur. See also the section SQL Services.

DELIMID

Possible Values	Explanation	Default Value
"	Double quotation mark	
'	Single quotation mark	
	No value: Delimited identifiers are not enabled.	No value

This parameter determines the escape character to be used for generating delimited SQL identifiers for the column names and table names in SQL statements. A delimited identifier is a sequences of one or more characters enclosed in escape characters. You must specify a delimited identifier if you use SQL-reserved words for column names and tables names, as demonstrated in the Example of DELIMID below.

To enable generation of delimited identifiers, DELIMID must be set to double quotation mark (") or single quotation mark (').

The escape character specified for DELIMID and the SQL STRING DELIMITER are mutually exclusive. This implies that the mark (double or single quotation) used to enclose alphanumeric strings in SQL statements must be different from the value specified for DELIMID. If you enable delimited identifiers, ensure that the value specified for DELIMID also complies with the SQL STRING DELIMITER value of your DB2 installation.

See also the RWRDONL parameter to determine which delimited identifiers are generated in the SQL string.

Example of DELIMID:

In the following example, a double quotation mark (") has been specified as the escape character for the delimited identifier:

Natural statement:

```
SELECT FUNCTION INTO #FUNCTION FROM XYZ-T1000
```

Generated SQL string:

```
SELECT "FUNCTION" FROM XYZ.T1000
```

MAXLOOP - Maximum Number of Nested Database Loops

This parameter specifies the maximum possible number of nested database loops.

Possible values	Default value
1 - 99	10

REFRESH

Possible Values	Default Value
ON/OFF	OFF

This parameter is used to automatically connect to the DB2 server that was active when the last transaction was executed.

To refresh the server connection, use the following SQL statement of DB2:

```
CONNECT ? IDENTIFIED BY ? TO ?
```

Value	Explanation
OFF	No automatic refresh is performed.
ON	An automatic refresh is performed every time before a database transaction starts.

RWRDONL

Possible Values	Default Value
ON/OFF	ON

This parameter determines which identifiers are generated as delimited identifier in an SQL string. RWRDONL only takes effect if the setting of the DELIMID parameter allows delimited identifiers.

If RWRDONL is set to ON, only identifiers that are reserved words are generated as delimited identifiers. The list of reserved words is contained in the NDBPARM macro. This list has been merged from the lists of reserved words for DB2 for z/OS, DB2 for VSE/VM, DB2 UDB for LINUX, OS/2, Windows and UNIX, and ISO/ANSI SQL99.

If RWRDONL is set to OFF, all identifiers are generated as delimited identifiers.

STATDYN - Allow Static to Dynamic Switch

This parameter is used to allow dynamic execution of statically generated SQL statements if static execution returns an error.

PossibleValue	Default value	Explanation
NEVER	NEVER	Dynamic execution is never allowed.
ALWAYS		Dynamic execution is always allowed after an error.
SPECIAL		Dynamic execution is allowed after special errors only. These special errors are: NAT3706 Load module not found. SQL -805 SQL/DS package does not exist. SQL -818 Mismatch of timestamps.

Database Management

This section covers the following topics:

- SYSSQL Utility
 - Natural System Commands for SQL/DS
-

SYSSQL Utility

The Natural interactive catalog utility SYSSQL allows you to do SQL/DS database management without leaving your development environment.

With SYSSQL you can maintain SQL/DS tables and other SQL/DS objects.

The SYSSQL utility incorporates an SQL generator that automatically generates from your input the SQL code required to maintain the desired SQL/DS object. You can display, modify, save and retrieve the generated SQL code.

The DDL/DCL definitions are stored in the library SYSSQL on the FDIC system file.

The SYSSQL utility offers two modes of operation: Fixed Mode and Free Mode. To switch between the two modes, you press PF4.

- Fixed Mode
- Free Mode

Fixed Mode

In fixed mode, input screens with syntax graphs help you to specify correct SQL code. You simply enter the required data on input screens, and the data are automatically checked to ensure that they comply with the SQL syntax of SQL/DS. Then, SQL members are generated from the entered data. The members can be executed directly by pressing PF5. But you can also switch to free mode, where the generated SQL code can be modified.

For each field where a window can be invoked, you can specify an "S". When you press ENTER, the window appears and you can select or enter the necessary information. If such a selection is required, an "S" is already preset when the corresponding screen is invoked.

When you press ENTER again, the window closes and if data have been entered, the field is marked with "X" instead of "S". If not, the field is left blank or marked with "S" again.

This continues each time you press ENTER until no "S" remains. To redisplay a window where data have been entered, you change its "X" mark back to "S".

If another letter or character is used, an appropriate error message appears on the screen. The wrong character is automatically replaced by an "S" and if you press ENTER again, the corresponding window appears.

In fields where keywords are to be entered, you have to enter one of the keywords displayed beneath the field. Default keywords are highlighted.

Creating an SQL/DS Table

The following example illustrates how to use the SYSSQL utility to create an SQL/DS table in fixed mode.

When you log on to library SYSSQL and issue the command MENU, the SYSSQL Main Menu appears:

```

14:41:38                **** SYSSQL Utility ****                1999-09-29
                        - Main Menu -

+----- Maintenance -----+   +----- Authorizations -----+
!   x CREATE                !   !   _ GRANT                        !
!   _ ACQUIRE DBSPACE     !   !   _ REVOKE                       !
!   _ ALTER                !   !   _ LOCK TABLE                   !
!   _ DROP                 !   !   _ CONNECT                      !
!   _ UPDATE STATISTICS    !   !                                     !
+-----+-----+-----+-----+
                        +----- Descriptions -----+
!   _ EXPLAIN              !
!   _ COMMENT ON          !
+-----+-----+-----+

+----- Comments -----+
!   Enter ? for HELP or press PF1                !
!   Enter . to QUIT or press PF12                !
!   Press PF4 to enter Free-Mode                 !
+-----+-----+-----+

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help                               Free                               Exit
SYSSQL4776 Please mark your choice.

```

When you select the CREATE function, a window is invoked which shows you a list of all available objects, and you are prompted for the type of object to be created, in this case a table:

```

14:41:39                **** SYSSQL Utility ****                1999-09-29
                        - Main Menu -

+----- M +-----+ +----- Authorizations -----+
!   x CREATE !   _ INDEX   !   !   _ GRANT           !
!   _ ACQUIRE !   _ SYNONYM !   !   _ REVOKE          !
!   _ ALTER   !   x TABLE  !   !   _ LOCK TABLE     !
!   _ DROP    !   _ VIEW    !   !   _ CONNECT         !
!   _ UPDATE  !           !   !                   !
+-----+ +-----+ +-----+
                        +----- Descriptions -----+
!   _ EXPLAIN           !
!   _ COMMENT ON       !
+-----+

+----- Comments -----+
! Enter ? for HELP or press PF1           !
! Enter . to QUIT or press PF12          !
! Press PF4 to enter Free-Mode           !
+-----+

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10---PF11---PF12---
      Help                               Free                               Exit
SYSSQL4776 Please mark your choice.
    
```

The first Create Table syntax input screen is displayed. You can enter the creator and table names on this screen, as well as the individual column names, formats and lengths, as shown below:

```

11:41:52          **** SYSSQL/DS Utility ****          1999-08-28
                  - Create Table -                      Page: 01

>>----- CREATE TABLE ----- SAG_____ . PERSONNEL_____ ----->
                  <creator.>table-name

>- PERS-NO_____ DECIMAL_____ ( 8_____ ) NN -- _ -- _ -- _ -( S_ - A +
+- NAME_____ CHAR_____ ( 25_____ ) NN -- _ -- _ -- _ -- _ - _ +
+- FIRST-NAME_____ CHAR_____ ( 25_____ ) NN -- _ -- _ -- _ -- _ - _ +
+- AGE_____ DECIMAL_____ ( 2_____ ) NN -- _ -- _ -- _ -- _ - _ +
+- SALARY_____ DECIMAL_____ ( 5,2_____ ) ___ -- _ -- _ -- _ -- _ - _ +
+- FUNCTION_____ INTEGER_____ ( _____ ) ___ -- _ -- _ -- _ -- _ - _ +
+- EMPL_SINCE_____ DATE_____ ( _____ ) NN -- _ -- _ -- _ -- _ - _ +
+- _____ ( _____ ) ___ -- _ -- _ -- _ -- _ - _ )
      column-name          format          length  NN    S  field CCS  PRIMARY !
                               NU    M  proc ID  KEY A/D !
                               NP    B  +-----+
                                     +- PCTFREE= ___ ->
                                               0-99

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10---PF11--PF12---
      Help  Next          Free  Exec  Top   Bwd   Fwd   Bot           Error Menu

```

Note:

Since the specification of any special characters as part of a Natural field or DDM name does not comply with Natural naming conventions, any special characters allowed within SQL/DS should be avoided. The same applies to SQL/DS delimited identifiers, which are not supported by Natural.

In addition, various attributes can be specified for each column.

- In the NN/NU/NP field you can specify:
 - NN (NOT NULL) if the column may not contain null values,
 - NP (NOT NULL PRIMARY KEY) if the column is the primary key
 - NU (NOT NULL UNIQUE) if the column is a unique key
- In the S/M/B field you can specify the following for character columns:
 - S (FOR SBCS DATA)
 - B (FOR BIT DATA)
 - M (FOR MIXED DATA)
- You can mark the field "fieldproc" to display a window where you can specify a field procedure which has to be executed for that column.
- For character and graphic columns you can mark the CCSID to display a window where you can specify a CCSID to be used for that column.

You can also specify which columns are to be part of a primary key if the primary key is comprised of multiple columns. To do so enter an "S" or the positional number in the first column of the field PRIMARY KEY.

A primary key is a set of column values that enforce referential integrity. Only one primary key definition is allowed per table. Primary key values must be unique and must be defined as NOT NULL.

If a column is to be part of a primary key, you also have to specify whether the values from this column are to be arranged in ascending ("A") or descending order ("D"), where "A" (Asc) is the default value. In addition, you can specify the percentage of space within each index page for later insertions and updates of the primary key (the default value is 10%).

If a letter or character other than those mentioned above is used, an appropriate error message appears on the screen and the wrong character is automatically replaced by the appropriate one.

Windows like the one below may help you in making a valid selection. They are invoked by entering the help character "?" in the appropriate field on the screen:

```

16:50:09          **** SYSSQL Utility ****          1999-09-29
                  - Create Table -                  Page: 01

>>--- CREATE TABLE ----- SAG_____ . PERSONNEL_____ ----->
                        <creator.>table-na +-----+
                                ! Please mark your choice: !
>-( PERS-NO_____ - DECIMAL_____ ( 8_ ! _ INTEGER !
>-- NAME_____ - CHAR_____ ( 25 ! _ SMALLINT !
>-- FIRST-NAME_____ - CHAR_____ ( 25 ! _ FLOAT(integer,integer) !
>-- AGE_____ - DECIMAL_____ ( 25 ! _ DECIMAL(integer,integer) !
>-- SALARY_____ - DECIMAL_____ ( 2_ ! _ CHAR(integer) !
>-- FUNCTION_____ - INTEGER_____ ( 5, ! _ VARCHAR(integer) !
>-- EMPL-SINCE_____ - DATE_____ ( __ ! _ LONG VARCHAR !
>-- _____ - ?_____ ( __ ! _ GRAPHIC(integer) !
      column-name          format          ( __ ! _ VARGRAPHIC(integer) !
                                ( __ ! _ LONG VARGRAPHIC !
                                ! _ DATE !
                                ! _ TIME !
                                ! _ TIMESTAMP !
                                ! Valid abbreviations: !
                                ! I,S,F,DE,C,VARC,L VARC,G, !
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7 ! VARG,L VARG,DA,TIME,TIMES !
      Help Next          Free Exec Top Bwd !
                                +-----+
    
```

Press ENTER to close the window again.

As you can see on the above screen, the beginning of the syntax specification for an SQL statement is always indicated by ">>".

In the case of complex SQL statements, more than one input screen may be required. If so, you can switch to the following screen by pressing PF2 (Next).

You can specify up to 16 constraint blocks. In each block you can define a foreign key and a unique key. In the top right-hand corner of the screen, the index of the currently displayed referential constraint block (1) is displayed. You can page forward and backward through the constraint blocks by pressing PF7 and PF8.

When you have entered all information, you can press either PF3 (Prev) to return to the previous screen, or PF2 (Next) to go to the last screen as shown below:

```

17:05:38          **** SYSSQL/DS Utility ****          1999-08-28
                  - Create Table -                      Page: 01

>-----+-----+-----+-----+-----+-----+-----+-----+-----><
          !                                     !
          +----- IN -- SAG_____ . DEMO_____ +-----+
                                <owner.>dbspace-name

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10---PF11---PF12---
      Help       Prev  Free  Exec                               Error Menu

```

On this screen, you can specify the dbspace where the table is to be created.

As you can see on the above screen, the end of the syntax specification for an SQL statement is always indicated by "><".

If you press PF2 (Prev) on this screen, you return to the previous screen.

When all information has been entered, you can either switch to free mode (PF4) or submit the created member directly to SQL/DS for execution (PF5). If execution is successful, you receive the message:

```
Statement(s) successful, SQLCODE = 0
```

If not, an error code is returned.

Once a table has been created, the data type of its columns cannot be changed and columns cannot be deleted. However, new columns can be added using the ALTER TABLE function as described in the following section.

Altering an SQL/DS Table

With the ALTER TABLE function you can add single columns to an existing table. You can also add, drop, activate or deactivate primary and foreign keys.

The following example illustrates how to use the SYSSQL utility to alter an SQL/DS table in fixed mode.

When you mark the ALTER function in the SYSSQL Main Menu and press ENTER, a window prompts you for the type of object to be altered - in this case a TABLE:

```

17:05:33                **** SYSSQL Utility ****                1999-09-29
                        - Main Menu -

+----- Maintenance -----+ +----- Authorizations -----+
!  _ CREATE                  !  !  _ GRANT                    !
!  _ ACQUIRE +-----+    !  !  _ REVOKE                   !
!  x ALTER  !  _ DBSPACE    !  !  _ LOCK TABLE             !
!  _ DROP    !  x TABLE    !  !  _ CONNECT                 !
!  _ UPDATE  !              !  !                                  !
+-----+ +-----+ +-----+
                        +----- Descriptions -----+
                        !  _ EXPLAIN                      !
                        !  _ COMMENT ON                   !
                        +-----+

+----- Comments -----+
!  Enter ? for HELP or press PF1                !
!  Enter . to QUIT or press PF12               !
!  Press PF4 to enter Free-Mode                !
+-----+

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help                      Free                      Exit
SYSSQL4776 Please mark your choice.

```

When you press ENTER again, the first Alter Table input screen is displayed:

```

17:07:04          **** SYSSQL/DS Utility *          1999-08-28
                    - Alter Table -

>>--- ALTER TABLE ----- _____ . _____ ----->
                                <creator.>table-name

>+--- ADD -- _____ ( _____ ) -- _ -- _ -- _-+>
!           column-name      format      length      S field CCS !
!                                           M proc ID  !
!                                           B           !
!                                           !           !
+---+-----+--- PRIMARY KEY --- ( --- _ --- ) ----- PCTFREE= -- ___ -----+
! +- ADD -+                column-names                0-99          !
!                                           !           !
+--- DROP ---+--- PRIMARY KEY --- _ -----+
!                                           !           !
+--- FOREIGN KEY --- _____ -----+
!                constraint-name                !
+--- UNIQUE KEY --- _____ -----+
!                constraint-name                !

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10---PF11--PF12---
      Help Next          Free Exec                                Error Menu
    
```

You can enter the creator and table names on this screen, as well as the name, format and length of an additional column.

In addition, you can define a primary key as described in the section Creating an SQL/DS Table. You can also drop an already existing primary key, thereby removing all referential constraints in which the current table is a parent table.

You can also drop any already existing foreign key or unique key by specifying its constraint name. If a foreign key is dropped the corresponding referential constraint is removed.

Once you have entered all necessary information, press PF2 (Next) to display the next Alter Table input screen, where you can add or drop foreign keys and unique keys.

```

17:07:56          **** SYSSQL/DS Utility *          1999-08-28
                    - Alter Table -

+>>-----+-----+- FOREIGN KEY --- _____ --- ( --- _ --- ) ----->
      +- ADD -+                constraint-name      column-names

>---- REFERENCES ----- _____ . _____ ----->
                        <creator.> table-name

>---- ON DELETE +- S - RESTRICT -+-+-----><
      +- _ - CASCADE --+
      +- _ - SET NULL -+

+>>-----+-----+- UNIQUE KEY ----- _____ --- ( --- _ --- ) ----->
      +- ADD -+                constraint-name      column-names

>----- PCTFREE= ----- _____ -----><
                        0-99

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Next  Prev  Free  Exec                               Error Menu

```

A foreign key or unique key is added as described in the section Creating an SQL/DS Table.

When you have entered all information you can press either PF3 (Prev) to return to the previous screen, or PF2 (Next) to go to the last screen as shown below:

```

17:08:40          **** SYSSQL/DS Utility ****          1999-08-28
                    - Alter Table -

>--- ACTIVATE  ---+----- _ --- ALL -----><
      !
      +----- _ --- PRIMARY KEY -----+
      !
      +----- FOREIGN KEY -- _____ +
      !                constraint-name   !
      +----- UNIQUE KEY --- _____ +
                        constraint-name

>--- DEACTIVATE -+----- _ --- ALL -----><
      !
      +----- _ --- PRIMARY KEY-----+
      !
      +----- FOREIGN KEY -- _____ +
      !                constraint-name   !
      +----- UNIQUE KEY --- _____ +
                        constraint-name

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help      Prev  Free  Exec                               Error Menu

```

In the ACTIVATE part you have three options available. You can activate:

- ALL, which automatically enforces all the referential constraints defined for a primary key.
- PRIMARY KEY, which automatically enforces the primary key.
- FOREIGN KEY *constraint-name*, which automatically enforces the specified referential constraint.

In the DEACTIVATE part you have three options available. You can deactivate:

- ALL, which deactivates the primary key and all active foreign keys in the table.
- PRIMARY KEY, which drops the primary key index from the table and implicitly deactivates all active dependent foreign keys.
- FOREIGN KEY *constraint-name*, which deactivates the specified referential constraint.

By specifying any of these options, the restrictions imposed by the referential constraints are suspended and the parent and dependent tables involved in a referential constraint are made unavailable to users other than the DBA and the owner of the table.

Press PF2 (Prev) to return to the previous screen.

Free Mode

When free mode is invoked from fixed mode, the data that were entered in fixed mode are shown as generated SQL code, which can be saved for later use or modification. The editor provided is an adapted version of the Natural program editor.

If you modify an SQL member in free mode, this has no effect on the fixed-mode version of the member. You can save your modified code in free mode, but when you return to fixed mode, the original data appear again. Thus, both original and modified data are available.

In free mode you can execute the member currently in the source area by pressing PF5 (as in fixed mode).

If you switch to free mode after you have created an SQL/DS table in fixed mode as described in the section [Creating an SQL/DS Table](#), the free-mode editor displays the generated SQL code as in the following sample screen:

```

15:13:39                **** SYSSQL Utility ****                1999-09-29
                        - Free Mode -                            Member :

Command:
+-----+
! CREATE TABLE SAG.PERSONNEL                                     !
!   (PERS-NO           DECIMAL(8)           NOT NULL,          !
!   NAME              CHAR(25)             NOT NULL,          !
!   FIRST-NAME        CHAR(25)             NOT NULL,          !
!   AGE               DECIMAL(2)           NOT NULL,          !
!   SALARY            DECIMAL(5,2),        !
!   FUNCTION          INTEGER,             !
!   EMPL-SINCE        DATE                 NOT NULL,          !
!   PRIMARY KEY (PERS-NO),                  !
!   FOREIGN KEY  AUTO-NAME (NAME)          !
!     REFERENCES SAG.AUTOMOBILES          !
!     ON DELETE SET NULL                  !
!   )                                     !
!   IN SAG.DEMO                           !
+-----+

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help           Fix   Exec   Top   Bwd   Fwd   Bot           Error Menu

```

The Free-Mode Editor

The free-mode editor available is almost identical to the Natural program editor and allows you to edit the generated SQL code. All program editor line commands and the following editor commands are available:

Command	Function
ADD <i>dnf</i>	Adds <i>n</i> empty lines.
CHANGE	Scans for the value entered as <i>scandata</i> and replaces each such value found with the value entered as <i>replacedata</i> . The syntax for this command is: CHANGE ' <i>scandata</i> ' ' <i>replacedata</i> '
CLEAR	Clears the editor source area (including the line markers "X" and "Y").
DX, DY, DX-Y	Deletes the X-marked line or the Y-marked line or the block of lines delimited by "X" and "Y".
EX, EY, EX-Y	Deletes source lines from the top of the source area to - but not including - the X-marked line, or from the source line following the Y-marked line to the bottom of the source area, or all source lines in the source area excluding the block of lines delimited by "X" and "Y".
LET	Undoes all modifications made to the current screen since the last time ENTER was pressed, including all line commands already entered but not yet executed.
POINT	Positions the line in which the line command ".N" was entered to the top of the current screen.
RESET	Deletes the current X and/or Y line markers and any marker previously set with the line command ".N".
SCAN [<i>'scan-value'</i>]	Scans for the string <i>scan-value</i> in the source area.
SCAN = [<i>+ -</i>]	Scans forwards (+) or backwards (-) for the next occurrence of the scan value.
SHIFT [<i>- +nn</i>]	Shifts the block of source lines delimited by the X and Y markers to the left (-) or right (+). " <i>nn</i> " represents the number of characters the source line is to be shifted.

For further details, refer to Program Editor in the Natural Editors documentation.

In addition, the following SQL code maintenance commands are available:

Command	Function
INSERT <i>member-name</i>	Saves the code in the source area as a member. If you press PF5, the code in the source area can also be executed as in fixed mode.
SELECT <i>member-name</i>	Reads the specified member into the source area.
DELETE <i>member-name</i>	Deletes the specified member.
LIST QUERY <i>member-name</i>	Displays a list of members on the screen using asterisk notation (*). For example, "L Q A*" would display a list of all SQL code members beginning with "A".

Member names must correspond to the naming conventions for Natural objects, which means they can be up to eight characters long and must start with a letter.

You can also always refer to the SYSSQL help system, which is invoked via PF1.

Natural System Commands for SQL/DS

This section describes special Natural system commands for the use with SQL/DS. There are three Natural system commands which perform SQL/DS-specific functions:

- **LISTSQL Command**
Lists Natural DML statements and their corresponding SQL statements.
- **SQLERR Command**
Provides diagnostic information about an SQL/DS error
- **LISTDBRM Command**
Displays either a list of packages for a particular Natural program or a list of Natural programs that reference a particular package.

Note:

LISTDBRM has to be issued from library SYSSQL, which means you have to LOGON to SYSSQL first and then enter the command LISTDBRM.

LISTSQL Command

LISTSQL [*object-name*]

The LISTSQL command lists the Natural statements in the source code of a programming object that are associated with a database access, and the corresponding SQL statements into which they have been translated. LISTSQL is issued from the Natural NEXT prompt.

Thus, before executing a Natural program which accesses an SQL/DS table, you can view the generated SQL code by using the command LISTSQL.

If a valid object name is specified, the object to be displayed must be stored in the library to which you are currently logged on.

If no object name is specified, LISTSQL refers to the object currently in the Natural source area.

The generated SQL statements contained in the specified object are listed one per page.

Sample LISTSQL Screen:

```

13:55:18          * * * NATURAL Tools for SQL * * *          1999-08-29
Member N2PIGDDM          LISTSQL          Library SYSSQL

NATURAL statement at line 3820          Stmt 4 / 4

  FIND SYSTEM-SYSCOLUMNS WITH TNAME EQ TABLE-NAME AND
    CREATOR = ICREATOR SORTED BY COLNO
  IF NO RECORDS FOUND DO

Generated SQL statement  Mode : dynamic  DBRM :          Line 1 / 5

  SELECT COLNO, CNAME, COLTYPE, SYSLLENGTH, NULLS, REMARKS, REMARKS,
         CLABEL, LENGTH
  FROM   SYSTEM.SYSCOLUMNS
  WHERE  TNAME = ? AND CREATOR = ?
  ORDER BY COLNO

Command ==>          Queryno for EXPLAIN 1_____
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10---PF11---PF12---
          Error Exit  Expl          Parms - +          Prev Next Canc

```

Within the listed results, you can go from one listed SQL statement to another by pressing PF10 (Prev) or PF11 (Next). If a single SQL statement does not fit on the screen, you can scroll backwards or forwards by pressing PF7 or PF8, respectively.

If a static DBRM has been generated, the name of this DBRM is displayed in the DBRM field of the LISTSQL screen; otherwise, the DBRM field remains empty.

If an error occurs, PF2 (Error), which executes the SQLERR command, can be used to provide information about SQL/DS errors.

With PF4 (Expl), a SQL/DS EXPLAIN command can be executed for the SQL statement currently listed. The query number (Queryno) for the EXPLAIN command is set to "1" by default, but you can overwrite this default.

With PF6 (Parms), a further screen is displayed which lists all parameters from the SQLDA for the currently displayed SQL statement:

```

14:04:25          * * * NATURAL Tools for SQL * * *          1999-08-29
Member N2PIGDDM          LISTSQL          Library SYSSQL

      Mode : dynamic   DBRM :          Contoken :

      static parms : (1st)
                    (2nd)

      SQLDA

                                DBID : 250  FNR :   3  CMD : S2 3820 08

Nr  Type   Length
1.  SMALLINT  2          0F5C C0C2 0002 01F5 0000 0000 0901 0000
2.  CHAR      18          0F5E 0012 0012 01C5 0000 0000 0D01 0000
3.  CHAR       8          0F70 0008 0008 01C5 0000 0000 0901 0000
4.  SMALLINT  2          0F78 C0C2 0002 01F5 0000 0000 0901 0000
5.  CHAR       1          0F7A 0001 0001 01C5 0000 0000 0901 0000
6.  VARCHAR   127         002A 00FE 007F 01C1 0000 0000 0901 0000
7.  VARCHAR   127         0038 00FE 007F 01C1 0000 0000 0901 0000
8.  CHAR      30          1079 001E 001E 01C5 0000 0000 0901 0000
9.  CHAR       7          1097 0007 0007 01C5 0000 0000 0801 0000

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
                                Exit          -      +                                Canc
    
```

In static mode, static information is also displayed, which includes the static DBRM name, the SQL/DS consistency token and some internal static parameters.

EXPLAIN Command

LISTSQL enables you to use the SQL/DS command EXPLAIN, which provides information on the SQL/DS optimizer's choice of strategy for executing SQL statements.

Natural executes the EXPLAIN command for the SQL statement that is displayed on the LISTSQL screen.

The information determined by the SQL/DS optimizer is written into your PLAN_TABLE. Natural then reads the table and displays the contents.

```

14:05:42          * * * NATURAL Tools for SQL * * *          1999-08-29
Queryno 1          EXPLAIN Result                            Row 1 / 1

          Estimated cost :   16.3 timerons

          Qblockno          Table
          Planno Method Tabno creator          Tablename
          ---
          1      1          1      SYSTEM      SYSCOLUMNS

          Access Access
          type creator          Accessname          sort_new sort_comp
          ---
          I      SYSTEM      ICOL          Y          N

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
          Exit          Del          -          +          Canc

```

The Query Number is set to "1" by default, but you can overwrite this default.

SQLERR Command

The SQLERR command is used to obtain diagnostic information about an SQL/DS error.

When an SQL/DS error occurs, Natural issues an appropriate error message. When you enter the SQLERR command, the following information on the most recent SQL/DS error is displayed:

- the Natural error message number;
- the corresponding reason code (if applicable);
- the variable SQLCODE returned by SQL/DS;
- the SQL/DS error message.

The SQLERR command can be issued either from the Natural NEXT prompt or from within a Natural program (by using the FETCH statement).

Sample SQLERR Diagnostic Information Screen:

```

*** SQLERR Diagnostic Information ***
----- NATURAL SQL Interface Codes -----
Return Code: 3700          Reason Code: 0          SQL code : -204
----- SQLCA -----
SQLERRP (Adabas SQL Subroutine where error occurred)      :  ARIXOCA
SQLERRD (Adabas SQL Internal State)
  RDS Return Code          :          100
  DBSS Return Code        :          0
  Number of Rows Processed :          0
  Estimated Cost           :          1.0
  Syntax error on PREPARE or EXECUTE IMMEDIATE           :          0
  Buffer Manager ERROR Code :          0
SQLWARN (Warning Flags)
  Data truncated
  Null Values ignored(AVG,SUM,MAX,MIN)                   :
  No. of columns greater than no. of host variables     :
  UPDATE/DELETE without WHERE clause                     :
  SQL statement causes a performance degradation        :
  Adjustment to DATE/TIMESTAMP Variable made           :
SQL/DS Error Message :
SAG.SYSTABLES not found in system catalog

```

LISTDBRM Command

The LISTDBRM command is used to display either existing packages of Natural programs or Natural programs referencing a given package.

Since LISTDBRM has to be issued from the library SYSSQL, first LOGON to SYSSQL and then enter the command LISTDBRM. The following menu is displayed:

```

16:53:20                * * * LISDBRM Command * * *                1999-09-29

                                Code Function
                                -----
                                D  Display DBRMs of Programs
                                R  List Programs Referencing DBRM
                                ?  Help
                                .  Exit
                                -----
Code .. _  Library .. EXAMPLE_
           Member .. _____
           DBRM  .... _____

Command ==>
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help           Exit                               Canc

```

The following functions are available:

Code	Description
D	This function displays programs with SQL/DS access and their corresponding package (DBRM). If no package name is shown, the corresponding program uses dynamic SQL.
R	This function lists all programs that use a given package (DBRM). If no package name is specified, all programs that use dynamic SQL are listed.

The following parameters apply:

Parameter	Description
Library	Specifies the name of a Natural library. Library names beginning with "SYS" are not permitted. This parameter must be specified.
Member	Specifies the name of the Natural program (member) to be displayed. This parameter is optional and can be used to limit the output. If a value is specified followed by an asterisk (*), all members in the specified library with names beginning with this value are listed. If this field is left blank, or if an asterisk is specified only, all members in the specified library are listed.
DBRM	Specifies a valid package name. If left blank, programs that run dynamically are referenced. This parameter applies to function code "R" only.

Sample LISTDBRM Result Screen:

16:53:20	* * * LISTDBRM Command * * *					1999-04-26
Library	Name	Type	DBRM	User ID	Date	Time
-----	-----	-----	-----	-----	-----	-----
EXAMPLE	PROG1	Program	PACK1	SAG	93-07-17	11:10:43
EXAMPLE	PROG2	Program	PACK1	SAG	93-07-17	11:10:48
EXAMPLE	PROG3	Program	PACK2	SAG	93-07-17	11:11:04
EXAMPLE	PROG4	Program		SAG	93-07-17	11:11:07

DDM Generation

This section covers the following topics:

- Natural Data Definition Module - DDM
 - SQL Services
-

Natural Data Definition Module - DDM

To enable Natural to access an SQL/DS table, a Natural DDM of the table must be generated. This is done either with Predict (see the relevant Predict documentation for details) or with the Natural utility SYSDDM.

If you do not have Predict installed, use the SYSDDM function "SQL Services" to generate Natural DDMs from SQL/DS tables. This function is invoked from the main menu of SYSDDM and is described below.

SQL Services

If you are using SYSDDM SQL Services in a CICS environment, you should have specified CONVERS=ON in the NDBPARM module.

The SQL Services offer a number of functions related to SQL/DS tables. If you enter function code "B" for "SQL Services" on the main menu of the SYSDDM utility, a menu is displayed, which offers you the following functions:

- Select SQL Table from a List
- Generate DDM from an SQL Table
- List Columns of an SQL Table

The input field "SQL system" contains the name of the SQL system actually in use. If Natural for SQL/DS is the only SQL interface installed, the field is write-protected and contains "SQL/DS"; if not, enter "SQL/DS".

The individual functions are described below.

Select SQL Table from a List

This function is used to select an SQL/DS table from a list for further processing.

To invoke the function, enter function code "S" on the SQL Services Menu. If you enter the function code only, you obtain a list of all tables defined to the SQL/DS catalog.

If you do not want a list of all tables but would like only a certain range of tables to be listed, you can, in addition to the function code, specify a start value in the Table Name and/or Creator fields. You can also use asterisk notation (*) for the start value.

When you invoke the function, the "Select SQL Table from a List" screen is invoked displaying a list of all SQL/DS tables requested.

On the list you can mark an SQL/DS table with either "G" for "Generate DDM from an SQL Table" or "L" for "List Columns of an SQL Table". Then the corresponding function is invoked for the marked table.

Generate DDM from an SQL Table

This function is used to generate a Natural DDM from an SQL/DS table, based on the definitions in the SQL/DS catalog.

To invoke the function, enter function code "G" on the SQL Services Menu along with the name and creator of the table for which you wish a DDM to be generated. If you do not know the table name/creator, you can use the function "Select SQL Table from a List" to choose the table you want.

If you do not want the creator of the table to be part of the DDM name, enter a "N" in the field "DDM Name with Creator" when you invoke the Generate function (default is "Y").

Note:

Since the specification of any special characters as part of a field or DDM name does not comply with Natural naming conventions, any special characters allowed within SQL/DS must be avoided. SQL/DS delimited identifiers must be avoided, too.

If you wish to generate a DDM for a table for which a DDM already exists and you want the existing one to be replaced by the newly generated one, enter a "Y" in the Replace field when you invoke the Generate function.

By default, Replace is set to "N" to prevent an existing DDM from being replaced accidentally. If Replace is "N", you cannot generate another DDM for a table for which a DDM has already been generated.

DBID/FNR Assignment

When the "Generate DDM from an SQL Table" function is invoked for a table for which a DDM is to be generated for the first time, the DBID/FNR Assignment screen is displayed. If a DDM is to be generated for a table for which a DDM already exists, the existing DBID and FNR are used and the DBID/FNR Assignment screen is suppressed.

On the DBID/FNR Assignment screen, enter one of the database IDs (DBIDs) chosen at Natural installation time and the file number (FNR) to be assigned to the SQL/DS table. Natural requires these specifications for identification purposes only.

The range of DBIDs which are reserved for SQL/DS tables is specified in the NTDB macro of the Natural parameter module (see the relevant section in the Natural Parameter Reference documentation) in combination with the NDBID macro of the parameter module NDBPARAM. Any DBID not within this range is not accepted. The FNR can be any valid number between 1 and 255.

After a valid DBID and FNR have been assigned, a DDM is automatically generated from the specified table.

Long Field Redefinition

The maximum field length supported by Natural is 253 bytes. If an SQL/DS table contains a column which is longer than 253 bytes, this column has to be redefined as a one-dimensional array; otherwise the column is truncated and only the first 253 bytes are considered.

When redefined as an array, this array is represented in the DDM as a multiple-value field. Arrays are defined on the Long Field Redefinition screen, which is automatically invoked for each column over 253 bytes in length.

On the Long Field Redefinition screen you specify the element length of the array, which means the length of the occurrences. The number of occurrences depends on the length you specify.

If, for example, an SQL/DS column has a length of 2000 bytes, you can specify an array element length of 200 bytes, and you receive a multiple-value field with 10 occurrences, each occurrence with a length of 200 bytes.

Since redefined long fields are no multiple-value fields in the sense of Natural, the Natural C* notation cannot be applied to those fields.

When such a redefined long field is defined in a Natural view for being referenced by Natural SQL statements (that is, by host variables which represent multiple-value fields), both when defined and when referenced, the specified range of occurrences (index range) must always start with occurrence 1. If not, a Natural syntax error is returned.

Example:

```
UPDATE table SET varchar = #arr(*)
SELECT ... INTO #arr(1:5)
```

Note:

When such a redefined long field is updated with the Natural DML UPDATE statement, care must be taken to update each occurrence appropriately.

Length Indicator for Variable Length Fields - VARCHAR, LONG VARCHAR, VARGRAPHIC, LONG VARGRAPHIC

For each variable length column, an additional length indicator field (format/length I2) is generated in the DDM. The length is always measured in number of characters, not in bytes. To obtain the number of bytes of a VARGRAPHIC or LONG VARGRAPHIC field, the length must be multiplied by 2.

The name of a length indicator field begins with "L@" followed by the name of the corresponding field. The value of the length indicator field can be checked or updated by a Natural program.

If the length indicator field is not part of the Natural view and if the corresponding field is a redefined long field, the length of this field with UPDATE and STORE operations is calculated without trailing blanks.

Null Values

With Natural, it is possible to distinguish between a null value and the actual value "0" (or "blank") in an SQL/DS column.

When a DDM is generated from the SQL/DS catalog, an additional null indicator field is generated for each column which can be NULL; that is, which has neither "NOT NULL" nor "NOT NULL WITH DEFAULT" specified.

The name of the null indicator field begins with "N@" followed by the name of the corresponding field.

When the column is read from the database, the corresponding indicator field contains either "0" (if the column contains a value, including the value "0" or "blank") or "-1" (if the column contains no value).

Example:

The column "NULLCOL CHAR(6)" in an SQL/DS table definition would result in the following DDM fields:

```
NULLCOL      A    6.0
N@NULLCOL    I    2.0
```

When the field NULLCOL is read from the database, the additional field N@NULLCOL contains:

- "0" if NULLCOL contains a value (including the values "0" and "blank");
- "-1" if NULLCOL contains no value.

A null value can be stored in a database field by providing "-1" as input for the corresponding null indicator field.

Note:

If a column is NULL, an implicit RESET is performed on the corresponding Natural field.

List Columns of an SQL Table

This function lists all columns of a specific SQL/DS table.

To invoke this function, enter function code "L" on the SQL Services Menu along with the name and creator of the table whose columns you wish to be listed.

The List Columns screen for this table is invoked, which lists all columns of the specified table and displays the following information for each column:

Variable	Content
Name	The SQL/DS name of the column.
Type	The column type.
Length	The length (or precision if Type is DECIMAL) of the column as defined in the SQL/DS catalog.
Scale	The decimal scale of the column (only applicable if Type is DECIMAL).
Update	Y The column can be updated. N The column cannot be updated.
Nulls	Y The column can contain null values. N The column cannot contain null values.
Not	A column which is of a scale length or type not supported by Natural is marked with an asterisk (*). For such a column, a view field cannot be generated. The maximum scale length supported is 7 bytes. Types supported are: CHAR, VARCHAR, LONG VARCHAR, GRAPHIC, VARGRAPHIC, LONG VARGRAPHIC, DECIMAL, INTEGER, SMALLINT, DATE, TIME, TIMESTAMP and FLOAT.

The data types DATE, TIME, TIMESTAMP and FLOAT are converted into numeric or alphanumeric fields of various lengths: DATE is converted into A10, TIME into A8, TIMESTAMP into A26 and FLOAT into F8.

For SQL/DS, Natural provides an SQL/DS TIMESTAMP column as an alphanumeric field (A26) of the format "YYYY-MM-DD-HH.SS.MMMMMM".

Since Natural does not yet support computations with such fields, a Natural subprogram called NDBSTMP is provided to enable this kind of functionality.

Dynamic and Static SQL Support

This section covers the following topics:

- General Information
 - Internal Handling of Dynamic Statements
 - Preparing Natural Programs for Static Execution
 - Assembler/Natural Cross-References
 - Execution of Natural in Static Mode
 - Static SQL with Natural Security
 - Mixed Dynamic/Static Mode
 - Messages and Codes
-

General Information

The SQL support of Natural combines the flexibility of dynamic SQL support with the high performance of static SQL support.

In contrast to static SQL support, Natural dynamic SQL support does not require any special considerations with regard to the operation of the SQL interface. All SQL statements required to execute an application request are generated automatically and can be executed immediately with the Natural system command RUN. Before executing a program, you can look at the generated SQL code, using the LISTSQL command.

Access to SQL/DS through Natural has the same form whether dynamic or static SQL support is used. Thus, with static SQL support, the same SQL statements in a Natural program can be executed in either dynamic or static mode. An SQL statement can be coded within a Natural program and, for testing purposes, it can be executed using dynamic SQL. If the test is successful, the SQL statement remains unchanged and static SQL for this program can be generated.

Thus, during application development, the programmer works in dynamic mode and all SQL statements are executed dynamically, whereas static SQL is only created for applications that have been transferred to production status.



Internal Handling of Dynamic Statements

Natural automatically provides for the preparation and execution of each SQL statement and handles the opening and closing of cursors used for scanning a table.

This section covers the following topics:

- NDBIOMO
- Statement Table
- Processing of SQL Statements Issued by Natural

NDBIOMO

As each dynamic execution of an SQL statement requires a statically defined DECLARE STATEMENT and DECLARE CURSOR statement, a special I/O module (NDBIOMO) is provided which contains a fixed number of these STATEMENTs and CURSORs. This number is specified during the generation of NDBIOMO.

Statement Table

If possible, an SQL statement is only prepared once and can then be executed several times if required. For this purpose, Natural internally maintains a table of all SQL statements that have been prepared and assigns each of these statements to a DECLARED STATEMENT in NDBIOMO. In addition, this table maintains the cursors used by the SQL statements SELECT, FETCH, UPDATE (positioned) and DELETE (positioned).

Each SQL statement is uniquely identified by:

- the name of the Natural program that contains this SQL statement,
- the line number of the SQL statement in this program,
- the name of the Natural library into which this program was stowed,
- the time stamp when this program was stowed.

Once a statement has been prepared, it can be executed several times with different variable values, using the dynamic SQL statement EXECUTE USING DESCRIPTOR or OPEN CURSOR USING DESCRIPTOR respectively.

When the full capacity of the statement table is reached, the entry for the next prepared statement overwrites the entry for a free statement whose latest execution is the least recent one.

When a new SELECT statement is requested, a free entry in the statement table with the corresponding cursor is assigned to it and all subsequent FETCH, UPDATE and DELETE statements referring to this SELECT statement will use this cursor. Upon completion of the sequential scanning of the table, the cursor is released and free for another assignment. While the cursor is open, the entry in the statement table is marked as used and cannot be reused by another statement.

If the number of nested FIND (SELECT) statements reaches the number of entries available in the statement table, any further SQL statement is rejected at execution time and a Natural error message is returned.

The size of the statement table depends on the size specified for NDBIOMO. Since the statement table is contained in the SQL/DS buffer area, the DB2SIZE parameter may not be sufficient and may need to be increased.

Processing of SQL Statements Issued by Natural

The embedded SQL uses cursor logic to handle SELECT statements. The preparation and execution of a SELECT statement is done as follows:

- The typical SELECT statement is prepared by a program flow which contains the following embedded SQL statements (note that *X* and *SQLOBJ* are SQL variables, not program labels, and that the question marks below are parameter markers which indicate where values are to be inserted at execution time.):

```
DECLARE SQLOBJ STATEMENT

DECLARE X CURSOR FOR SQLOBJ

INCLUDE SQLDA (copy SQL control block)
```

Then, the following statement is moved into *SQLSOURCE*:

```
SELECT PERSONNEL_ID, NAME, AGE
FROM EMPLOYEES
WHERE NAME IN (?, ?)
AND AGE BETWEEN ? AND ?
```

```
PREPARE SQLOBJ FROM SQLSOURCE
```

- Then, the SELECT statement is executed as follows:

```
OPEN X USING DESCRIPTOR SQLDA
FETCH X USING DESCRIPTOR SQLDA
```

The descriptor *SQLDA* is used to indicate a variable list of program areas. When the OPEN statement is executed, it contains the address, length and type of each value which replaces a parameter marker in the WHERE clause of the SELECT statement. When the FETCH statement is executed, it contains the address, length and type of all program areas which receive fields read from the table. When the FETCH statement is executed for the first time, it sets the Natural system variable *NUMBER to a non-zero value if at least one record is found that meets the search criteria. Then, all records satisfying the search criteria are read by repeated execution of the FETCH statement.

- Once all records have been read, the cursor is released by executing the following statement:

```
CLOSE X
```

Preparing Natural Programs for Static Execution

Static SQL is generated in Natural batch mode for one or more Natural applications which can consist of one or more Natural object programs. The number of programs that can be modified for static execution in one run of the generation procedure is limited to 999.

During the generation procedure, the database access statements contained in the specified Natural objects are extracted, written to work files and transformed into a temporary Assembler program. If no Natural program is found that contains an SQL access or if any error occurs during static SQL generation, batch Natural terminates and condition code 40 is returned, which means that all further JCL steps should no longer be executed.

The temporary Assembler program is written to a temporary file (the Natural work file CMWKF06) and precompiled. During precompilation, a static SQL/DS package (access module) is created and after the precompilation, the precompiler output is extracted from the Assembler program and written to the corresponding Natural objects, which means that the Natural objects are modified (prepared) for static execution. The temporary Assembler program is no longer used and deleted.

Note:

Since the Assembler precompiler of SQL/DS does not support GRAPHIC field types, you cannot generate a static Assembler program if your Natural program(s) contain any references to GRAPHIC-type columns.

The Natural subprogram NDBDBRM can be used to check whether a Natural program contains an SQL access and whether it has been modified for static execution.

This section covers the following topics:

- Creating a Static SQL/DS Package under VSE/ESA
- Generation Procedure - CMD CREATE Command
- Modification Procedure - CMD MODIFY Command

Creating a Static SQL/DS Package under VSE/ESA

Under VSE/ESA, a static SQL/DS package is created by using the sample job I075.

The job I075 consists of the following steps:

Step 1: Generate the Static Assembler Program

- Define six Natural work files for output.
- Define as PHASE search library the library that contains the Natural batch module and the library where you installed this Natural for SQL/DS version (since the static generation process uses the Natural modules NDBSTAT and NDBCHNK).
- Define the necessary Natural commands and the Natural input for the static generation procedure.

The output (CMWKF06) consists of a temporary Assembler program which contains all the database access statements of the Natural objects involved and serves as input for the precompilation step below.

Step 2: Precompile the Generated Assembler Program

- Define as PHASE search library the library where you installed SQL/DS.
- Define the necessary precompiler options and specify your SQL user ID and password.

The precompiler output consists of a static SQL/DS package and a precompiled temporary Assembler program (IJSYSPH) which contains all the database access statements transformed from SQL into Assembler statements and serves as input for the modification step below.

Step 3: Modify the Natural Objects

- Define as PHASE search library the library that contains the Natural batch module.
- Define the necessary Natural commands and the Natural input for the object modification procedure.

The output consists of a modified Natural object which contains all required SQL/DS access information.

Generation Procedure - CMD CREATE Command

To generate static SQL for Natural programs, LOGON to library SYSSQL.

Note:

Since a new SYSSQL library has been created when installing Natural for SQL/DS, ensure that it contains all Predict interface programs necessary to run the static SQL generation. These programs are loaded into SYSSQL during Predict installation (see the relevant Predict documentation).

Then specify the CMD CREATE command and the Natural input necessary for the static SQL generation process; the CMD CREATE command has the following syntax:

```
CMD CREATE DBRM static-name USING using-clause
{application-name,object-name,excluded-object}
:
:
```

The generation procedure reads but does not modify the specified Natural objects. If one of the specified programs was not found or had no SQL access, return code 4 is returned at the end of the generation step.

Static Name

If the PREDICT DOCUMENTATION option is to be used, a corresponding Predict static SQL entry must be available and the *static-name* must correspond to the name of this entry. In addition, the *static-name* must correspond to the name of the static SQL/DS package to be created during precompilation. The *static-name* can be up to 8 characters long and must conform to Assembler naming conventions.

USING Clause

The *using-clause* specifies the Natural objects to be contained in the static SQL/DS package. These objects can either be specified explicitly as INPUT DATA in the JCL or obtained as PREDICT DOCUMENTATION from Predict.

$$\left\{ \begin{array}{l} \text{INPUT DATA} \\ \text{PREDICT DOCUMENTATION} \end{array} \right\} \left[\text{WITH XREF} \left\{ \begin{array}{l} \text{YES} \\ \text{NO} \\ \text{FORCE} \end{array} \right\} \right] [\text{LIB } \textit{lib-name}]$$

If the parameters to be specified do not fit in one line, specify the command identifier (CMD) and the various parameters in separate lines and use both the input delimiter (as specified with the ID parameter; default is ",") and the continuation indicator (as specified with the CF parameter; default is "%") as shown in the example below.

Example:

```
CMD
CREATE,DBRM,static,USING,PREDICT,DOCUMENTATION,WITH,XREF,NO,%
LIB,library
```

Alternatively, you can also use abbreviations as shown in the following example:

Example:

```
CMD CRE DBRM static US IN DA W XR Y LIB library
```

The sequence of the parameters USING, WITH and LIB is optional.

INPUT DATA

As input data, the applications and names of the Natural objects to be included in the static SQL/DS package must be specified in the subsequent lines of the job stream (*application-name,object-name*). A subset of these objects can also be excluded again (*excluded-objects*). Objects in libraries whose name begins with "SYS" can be used for static generation, too.

The applications and names of Natural objects must be separated by the input delimiter (as specified with the ID parameter; default is ","). If you wish to specify all objects whose name begins with a specific string of characters, use an *object-name* or *excluded-objects* name that ends with asterisk notation (*). To specify all objects in an application, use asterisk notation only.

<p>Example:</p> <pre>LIB1 , ABC* LIB2 , A* , AB* LIB2 , * . . .</pre>
--

The specification of applications/objects must be terminated by a line that contains a period (.) only.

PREDICT DOCUMENTATION

As Predict supports static SQL for SQL/DS, you can also have Predict supply the input data for creating static SQL by using already existing PREDICT DOCUMENTATION.

WITH XREF Option

As Predict Active References supports static SQL for SQL/DS, the generated static SQL/DS package can be documented in Predict and the documentation can be used and updated with Natural.

WITH XREF is the option which enables you to store cross-reference data for a static SQL entry in Predict each time a static SQL/DS package is created (YES). You can instead specify that no cross-reference data are stored (NO) or that a check is made to determine whether a Predict static SQL entry for this static SQL/DS package already exists (FORCE). If so, cross-reference data are stored; if not, the creation of the static SQL/DS package is not allowed. For more information on Predict Active References, refer to the Predict documentation.

When WITH XREF (YES/FORCE) is specified, XREF data are written for both the Predict static SQL entry (if defined in Predict) and each generated static Natural program. However, static generation with WITH XREF (YES/FORCE) is possible only if the corresponding Natural programs have been cataloged with XREF ON.

WITH XREF FORCE only applies to the USING INPUT DATA option.

Note:

If you do not use Predict, the XREF option must be omitted or set to NO and the module NATXRF2 need not be linked to the Natural nucleus.

Library Option - LIB

With the LIB option, a Predict library other than the default library (*SYSSTA*) can be specified to contain the Predict static SQL entry and XREF data. The name of the library can be up to eight characters long.

Modification Procedure - CMD MODIFY Command

The modification procedure modifies the Natural objects involved by writing precompiler information into the object and by marking the object header with the *static-name* as specified with the CMD CREATE command.

In addition, any existing copies of these objects in the Natural global buffer pool (if available) are deleted and XREF data are written to Predict (if specified during the generation procedure).

To perform the modification procedure, LOGON to SYSSQL and specify the CMD MODIFY command which has the following syntax:

CMD <u>MODIFY</u> [<u>XREF</u>]
--

The input for the modify step is the precompiler output which must reside on a dataset defined as the Natural work file CMWKF01.

The output consists of precompiler information which is written to the corresponding Natural objects. In addition, a message is returned telling you whether it was the first time an object was modified for static execution ("modified") or whether it had been modified before ("re-modified").

If the XREF option is specified, the Natural work file CMWKF02 must be defined to contain the resulting list of cross-reference information concerning the statically generated SQL statements (see also Assembler/Natural Cross-References).

Assembler/Natural Cross-References

If you specify the XREF option of the MODIFY command, an output listing is created on the work file CMWKF02, which contains the static SQL/DS package name and the Assembler statement number of each statically generated SQL statement together with the corresponding Natural source-code line number, program name, library name, database ID and file number.

Example:

DBRMNAME	STMTNO	LINE	NATPROG	NATLIB	DB	FNR	COMMENT
DEM2S	000087	0170	DEM2SUPD	HGK	00010	00032	SELECT
	000111	0230					UPD/DEL
DEM2S	000121	0370	DEM2SINS	HGK	00010	00032	INSERT
DEM2S	000131	0150	DEM2SDEL	HGK	00010	00032	SELECT
	000155	0170					UPD/DEL
DEM2S	000165	0040	DEM2SDL2	HGK	00010	00032	UPD/DEL

Column	Explanation
DBRMNAME	Name of the static SQL/DS package which contains the static SQL statement.
STMTNO	Assembler statement number of the static SQL statement.
LINE	Corresponding Natural source code line number.
NATPROG	Name of the Natural program that contains the static SQL statement.
NATLIB	Name of the Natural library that contains the Natural program.
DB / FNR	Natural database ID and file number.
COMMENT	Type of SQL statement.

Execution of Natural in Static Mode

To be able to execute Natural in static mode, all users of Natural must have the SQL/DS RUN privilege for the static SQL/DS package created at precompilation.

To execute static SQL start Natural and execute the corresponding Natural program. Internally, the Natural runtime interface evaluates the precompiler data written to the Natural object and then performs the static accesses.

To the user there is no difference between dynamic and static execution.

Static SQL with Natural Security

Static generation can be disallowed with Natural Security by:

- restricting access to library SYSSQL,
- disallowing the module CMD,
- restricting access to the libraries that contain the relevant Natural objects,
- disallowing one of the commands CATALOG or STOW for a library that contains relevant Natural objects.

If a library is defined in Natural Security and the DBID and FNR of this library are different from the default specifications, the static generation procedure automatically switches to the DBID and FNR specifications defined in Natural Security.

Mixed Dynamic/Static Mode

It is possible to operate Natural in a mixed static and dynamic mode where static SQL is generated for some programs.

The mode in which a program is run is determined by the Natural object program itself. If a static SQL/DS package is referenced in the executing program, all statements in this program are executed in static mode.

Note:

Natural programs which return a runtime error do not automatically execute in dynamic mode. Instead, either the error must be corrected or, as a temporary solution, the Natural program must be recataloged to be able to execute in dynamic mode.

Within the same Natural session, static and dynamic programs can be mixed without any further specifications. The decision which mode to use is made by each individual Natural program.

Messages and Codes

This section lists the error messages that may be issued during static generation.

STAT9001 Object buffer allocation failed. RC = return code

Expl.: Program NDBCHNK has been invoked to allocate space for Natural object load, but the allocation has failed; retry or increase the free storage pool.

STAT9002 Write on object area failed. RC = return code

Expl.: Program NDBCHNK has been invoked to write a Natural object row into the appropriate buffer, but the write has failed; this is probably a NDBCHNK program error.

STAT9003 Statement entry retrieve error. RC = return code

Expl.: Program NDBSTAT has been invoked to retrieve next SQL/DS statement information from the Natural object loaded in main storage, but the retrieval has failed (RC was neither 0 (OK) nor 4 (EOP)); the probable cause is a Natural object inconsistency.

STAT9004 Unsupported Adabas command: command

Expl.: Program NDBSTAT has been invoked to retrieve next SQL/DS statement information from the Natural object loaded in main storage, but the Adabas command code returned was invalid; the probable cause is a Natural object inconsistency.

STAT9005 Freemain failed. RC = return code

Expl.: Program NDBCHNK has been invoked to free the area allocated for Natural object load, but the release has failed; this is probably a program error.

STAT9006 Call for timestamp of program failed. RC = return code

Expl.: Program NDBSTAT has been invoked to know the time stamp associated to the loaded Natural object, but the call has failed; this is probably a program error.

STAT9007 A-List item retrieve failed. RC = return code

Expl.: Program NDBSTAT has been invoked to retrieve the next compilation A-list element, but the retrieval has failed (RC was neither 0 (OK) nor 20 (EOL)); the probable cause is a Natural object inconsistency.

STAT9009 Invalid database field format: format

Expl.: Program NDBSTAT has been invoked to retrieve the next compilation A-list element, but the SQL/DS format code returned is invalid; the probable cause is a Natural object inconsistency.

STAT9011 Mailbox length exceeds maximum.

Expl.: The SQL/DS precompiler parameter (mailbox length) does not fit into a halfword, which means that the SQL statement contains too many variables.

STAT9014 Warning, may indicate a problem: second select table reset.

Expl.: The table for a second selection logs the statement number of all second SELECT statements. The table is reset if there are more than 100 entries, which means with many nested program loops. If the table is reset, no second UPDATE or DELETE statements are generated.

STAT9016 Versions of NDBSTAT and SQLGEN Natural programs do not match.

Expl.: The versions of the Natural programs used for the static generation (library SYSSQL) must be the same as one of the dynamically loaded Assembler program NDBSTAT.

STAT9017 address of program *program* in library *library* not found.

Expl.: A Natural object address was not found and the object cannot be modified. Either the object was not found or the address was wrong.

STAT9019 * Warning: Natural terminates abnormally, run may continue. *****

Expl.: Natural terminates abnormally with RC=4. A Natural member was explicitly entered which does not exist or does not have SQL access. The static generation can continue.

STAT9020 Start run of SQLGEN for DBRM *dbrm*.**STAT9021 Start merging temporary datasets.****STAT9022 Precompile input *input* written to temporary dataset.**

Expl.: The temporary assembler program for the precompiler input was written to a temporary dataset (Natural work file 5).

STAT9023 * END OF DATA *******STAT9024 No program with SQL access found.**

Expl.: None of the programs processed by the CMD command accessed an SQL system.

STAT9025 Program *program* in library *library* not found.**STAT9026 DB access module names *module* and *module* do not match.**

Expl.: The module name specified with the CMD CREATE command must be the same as the name of the DBRM specified in the DBRMLIB job card of the precompilation step.

STAT9027 Error *error* purging *program, library* in buffer pool. Run continues.**STAT9028 Number of programs to be generated exceeds 999.**

Expl.: The number of programs to be generated statically into one DBRM exceeds the maximum of 999.

STAT9029 Limit of *limit* NULL indicators per SQL statement exceeded.

Expl.: The maximum number of 1500 NULL indicators per SQL statement has been exceeded.

STAT9030 Number of variables to be generated exceeds 9999.

Expl.: The number of variables to be generated statically for one program exceeds the maximum of 9999.

STAT9031 XREF option "NO" and Predict DDA default "YES" do not match.

Expl.: The Predict DDA default setting for static SQL XREF is set to "YES", but the XREF option in the CMD command is set to "NO".

STAT9032 XREF option "FORCE" but no Predict documentation found.

Expl.: With the XREF option FORCE, the static generation continues and writes XREF data only if Predict documentation exists for a given DBRM. If there is no Predict documentation available, static generation is not performed.

STAT9033 No XREF data exist for member *member*.

Expl.: Either the Natural program which is to be statically generated cannot be cataloged with XREF=ON or the XREF data are not on the used Predict file.

STAT9034 XREF option "YES" or "NO" but Predict DDA default "FORCE".

Expl.: The Predict DDA default setting for static SQL XREF is set to "FORCE", but the XREF option in the CMD command is set to "NO" or "YES".

STAT9036 Given DBRM library not defined as 3GL Predict application.

Expl.: The library for the DBRM entered with the LIB option is not defined as 3GL application in Predict. Check the library name in Predict which contains the DBRM.

STAT9039 Library name must not be blank.

STAT9040 CATALOG or STOW not allowed for library *library*.

Expl.: The commands CATALOG or STOW are not allowed in your security environment. However, the CATALOG or STOW privilege is needed for static generation.

STAT9041 Natural Security restriction. Message code: *message code*

STAT9050 No Predict documentation for specified DBRM found.

Expl.: No documentation was found in Predict for the DBRM specified with the CMD command. Either the DBRM is not documented in the used Predict file or a wrong DBRM name has been specified.

STAT9062 No Predict installed.

STAT9063 XREF interface not linked. XREF option reset, run continues.

STAT9064 XREF option not set. Predict DDA default *default* taken.

Expl.: The Predict DDA default setting for static SQL XREF is read, because no XREF option is specified in the CMD command and the XREF interface and Predict are installed.

STAT9065 DBRM name must start with an uppercase character.

STAT9066 No Predict installed.

STAT9072 DBRM name must not be blank.

STAT9073 Invalid syntax for *parameter/option* specified.

STAT9092 Error occurred. XREF data for DBRM will be deleted.

STAT9093 Error *error* occurred in program *program* in line *line*.

STAT9094 Return code *return code* on call of *program*.

STAT9095 Error in *parameter* *parameter* on call of *program*.

Statements and System Variables

This section contains special considerations concerning Natural DML statements, Natural SQL statements and Natural system variables when used with SQL/DS.

It mainly consists of information also contained in the Natural documentation set where each Natural statement and system variable is described in detail.

This section covers the following topics:

- Natural DML Statements
 - Natural SQL Statements
 - Natural System Variables
 - Error Handling
-

Natural DML Statements

Summarized below are particular points you have to consider when using Natural DML statements with SQL/DS.

Any Natural statement not mentioned in this section can be used with SQL/DS without restriction.

- BACKOUT TRANSACTION
- DELETE
- END TRANSACTION
- FIND
- GET
- HISTOGRAM
- READ
- STORE
- UPDATE

BACKOUT TRANSACTION

This statement undoes all database modifications made since the beginning of the last logical transaction. Logical transactions can start either after the beginning of a session or after the last SYNCPOINT, END TRANSACTION or BACKOUT TRANSACTION statement.

Under CICS, the BACKOUT TRANSACTION statement is translated into an EXEC CICS ROLLBACK command. However, in pseudo-conversational mode, only changes made to the database since the last terminal I/O are undone. This is due to CICS-specific transaction processing.

Note:

Be aware that with terminal input in SQL/DS database loops, Natural switches to conversational mode.

As all cursors are closed when a logical unit of work ends, a BACKOUT TRANSACTION statement must not be placed within a database loop; instead, it has to be placed outside such a loop or after the outermost loop of nested loops.

If an external program written in another standard programming language is called from a Natural program, this external program should not contain its own ROLLBACK command if the Natural program issues database calls, too. The calling Natural program should issue the BACKOUT TRANSACTION statement on behalf of the external program.

DELETE

The DELETE statement is used to delete a row from an SQL/DS table which has been read with a preceding FIND, READ or SELECT statement. It corresponds to the SQL statement DELETE WHERE CURRENT OF *cursor-name*, which means that only the row which was read last can be deleted.

Example:

```
FIND EMPLOYEES WITH NAME = 'SMITH'
      AND FIRST_NAME = 'ROGER'
DELETE
```

Natural would translate the above Natural statements into SQL and assign a cursor name (for example, CURSOR1) as follows:

```
DECLARE CURSOR 1 CURSOR FOR
SELECT FROM EMPLOYEES
WHERE NAME = 'SMITH' AND FIRST_NAME = 'ROGER'
DELETE FROM EMPLOYEES
WHERE CURRENT OF CURSOR1
```

Both the SELECT and the DELETE statement refer to the same cursor.

Natural translates a DML DELETE statement into an SQL DELETE statement in the same way it translates a FIND statement into an SQL SELECT statement.

A row read with a FIND SORTED BY cannot be deleted due to SQL/DS restrictions explained with the FIND statement. A row read with a READ LOGICAL cannot be deleted either.

END TRANSACTION

This statement indicates the end of a logical transaction and releases all SQL/DS data locked during the transaction. All data modifications are made permanent.

Under CICS, the END TRANSACTION statement is translated into an EXEC CICS SYNCPOINT command.

As all cursors are closed when a logical unit of work ends, the END TRANSACTION statement must not be placed within a database loop; instead, it has to be placed outside such a loop or after the outermost loop of nested loops.

If an external program written in another standard programming language is called from a Natural program, this external program should not contain its own COMMIT command if the Natural program issues database calls, too. The calling Natural program should issue the END TRANSACTION statement on behalf of the external program.

Note:

With SQL/DS, the END TRANSACTION statement cannot be used to store transaction data.

FIND

The FIND statement corresponds to the SQL SELECT statement.

<p>Example:</p> <p>Natural statements:</p> <pre>FIND EMPLOYEES WITH NAME = 'BLACKMORE' AND AGE EQ 20 THRU 40 OBTAIN PERSONNEL_ID NAME AGE</pre> <p>Equivalent SQL statement:</p> <pre>SELECT PERSONNEL_ID, NAME, AGE FROM EMPLOYEES WHERE NAME = 'BLACKMORE' AND AGE BETWEEN 20 AND 40</pre>

Natural internally translates a FIND statement into an SQL SELECT statement. The SELECT statement is executed by an OPEN CURSOR command followed by a FETCH command. The FETCH command is executed repeatedly until either all records have been read or the program flow exits the FIND processing loop. A CLOSE CURSOR command ends the SELECT processing; See Processing of SQL Statements Issued by Natural.

The WITH clause of a FIND statement is converted to the WHERE clause of the SELECT statement. The basic search criterion for a SQL/DS table can be specified in the same way as for an Adabas file. This implies that only database fields which are defined as descriptors can be used to construct basic search criteria and that descriptors cannot be compared with other fields of the Natural view (that is, database fields) but only with program variables or constants.

Note:

As each database field (column) of a SQL/DS table can be used for searching, any database field can be defined as a descriptor in a Natural DDM.

The WHERE clause of the FIND statement is evaluated by the Natural processor **after** the rows have been selected via the WITH clause. Within the WHERE clause, non-descriptors can be used and database fields can be compared with other database fields.

Note:

SQL/DS does not have sub-, super-, or phonetic descriptors.

A FIND NUMBER statement is translated into a SELECT statement containing a COUNT(*) clause. The number of rows found is returned in the Natural system variable *NUMBER.

The FIND UNIQUE statement can be used to ensure that only one record is selected for processing. If the FIND UNIQUE statement is referenced by an UPDATE statement, a non-cursor ("searched") UPDATE operation is generated instead of a cursor-oriented (positioned) UPDATE operation. Therefore, it can be used if you want to update an SQL/DS primary key. It is, however, recommended to use Natural SQL ("searched" UPDATE statement) to update a primary key.

In static mode, the FIND NUMBER and FIND UNIQUE statements are translated into a SELECT SINGLE statement.

The FIND FIRST statement cannot be used. The PASSWORD, CIPHER, COUPLED and RETAIN clauses cannot be used either.

The SORTED BY clause of a FIND statement is translated into the SQL SELECT ... ORDER BY clause, which follows the search criterion. Because this produces a read-only result table, a row read with a FIND statement that contains a SORTED BY clause cannot be updated or deleted.

A limit on the depth of nested database loops can be specified at installation. If this limit is exceeded, a Natural error message is returned.

GET

This statement is ISN-based and, therefore, cannot be used with SQL/DS tables.

HISTOGRAM

The HISTOGRAM statement returns the number of rows in a table which have the same value in a specific column. The number of rows is returned in the Natural system variable *NUMBER.

Example:

Natural statements:

```
HISTOGRAM EMPLOYEES FOR AGE  
OBTAIN AGE
```

Equivalent SQL statement:

```
SELECT COUNT(*), AGE FROM EMPLOYEES  
WHERE AGE > -999  
GROUP BY AGE  
ORDER BY AGE
```

Natural translates the HISTOGRAM statement into an SQL SELECT statement, which means that the control flow is similar to the flow explained for the FIND statement.

READ

The READ statement can also be used to access SQL/DS tables. Natural translates a READ statement into an SQL SELECT statement.

READ PHYSICAL and READ LOGICAL can be used; READ BY ISN, however, cannot be used, as there is no SQL/DS equivalent to Adabas ISNs. The PASSWORD and CIPHER clauses cannot be used either.

Since a READ LOGICAL statement is translated into a SELECT ... ORDER BY statement - which produces a read-only table -, a row read with a READ LOGICAL statement cannot be updated or deleted (see Example 1 below). The start value can only be a constant or program variable; any other field of the Natural view (that is, any database field) cannot be used.

A READ PHYSICAL statement is translated into a SELECT statement without an ORDER BY clause and can, therefore, be updated or deleted (see Example 2 below).

Example 1:

Natural statements:

```
READ PERSONNEL BY NAME  
OBTAIN NAME FIRSTNAME DATEOFBIRTH
```

Equivalent SQL statement:

```
SELECT NAME, FIRSTNAME, DATEOFBIRTH FROM PERSONNEL  
WHERE NAME >= ' '  
ORDER BY NAME
```

Example 2:

Natural statements:

```
READ PERSONNEL PHYSICAL  
OBTAIN NAME
```

Equivalent SQL statement:

```
SELECT NAME FROM PERSONNEL
```

If the READ statement contains a WHERE clause, this clause is evaluated by the Natural processor **after** the rows have been selected according to the descriptor value(s) specified as search criterion.

STORE

The STORE statement is used to add a row to an SQL/DS table. The STORE statement corresponds to the SQL statement INSERT.

Example:

Natural statement:

```
STORE RECORD IN EMPLOYEES
    WITH PERSONNEL_ID = '2112'
        NAME = 'LIFESON'
        FIRST_NAME = 'ALEX'
```

Equivalent SQL statement:

```
INSERT INTO EMPLOYEES (PERSONNEL_ID, NAME, FIRST_NAME)
    VALUES ('2112', 'LIFESON', 'ALEX')
```

The PASSWORD, CIPHER and USING/GIVING NUMBER clauses cannot be used.

UPDATE

The Natural DML UPDATE statement updates a row in an SQL/DS table which has been read with a preceding FIND, READ or SELECT statement. It corresponds to the SQL statement UPDATE WHERE CURRENT OF *cursor-name* (positioned UPDATE), which means that only the row which was read last can be updated.

UPDATE with FIND/READ

As explained with the FIND statement, Natural translates a FIND statement into an SQL SELECT statement. When a Natural program contains a DML UPDATE statement, this statement is translated into an SQL UPDATE statement and a FOR UPDATE OF clause is added to the SELECT statement.

Example:

```
FIND EMPLOYEES WITH SALARY < 5000
  ASSIGN SALARY = 6000
  UPDATE
```

Natural would translate the above Natural statements into SQL and assign a cursor name (for example, CURSOR1) as follows:

```
DECLARE CURSOR1 CURSOR FOR
SELECT SALARY FROM EMPLOYEES WHERE SALARY < 5000
  FOR UPDATE OF SALARY
UPDATE EMPLOYEES SET SALARY = 6000
  WHERE CURRENT OF CURSOR1
```

Both the SELECT and the UPDATE statement refer to the same cursor.

Due to SQL/DS logic, a column (field) can only be updated if it is contained in the FOR UPDATE OF clause; otherwise updating this column (field) is rejected. Natural includes automatically all columns (fields) into the FOR UPDATE OF clause which have been modified anywhere in the Natural program or which are input fields as part of a Natural map.

However, an SQL/DS column is not updated if the column (field) is marked as "not updateable" in the Natural DDM. Such columns (fields) are removed from the FOR UPDATE OF list without any warning or error message. The columns (fields) contained in the FOR UPDATE OF list can be checked with the LISTSQL command.

The Adabas short name in the Natural DDM determines whether a column (field) can be updated.

The following table shows the ranges that apply:

Short-Name Range	Type of Field
AA - N9	non-key field that can be updated
Aa - Nz	non-key field that can be updated
OA - O9	primary key field
PA - P9	ascending key field that can be updated
QA - Q9	descending key field that can be updated
RA - X9	non-key field that cannot be updated
Ra - Xz	non-key field that cannot be updated
YA - Y9	ascending key field that cannot be updated
ZA - Z9	descending key field that cannot be updated
1A - 9Z	non-key field that cannot be updated
1a - 9z	non-key field that cannot be updated

Be aware that a primary key field is never part of a FOR UPDATE OF list. A primary key field can only be updated by using a non-cursor UPDATE operation.

A row read with a FIND statement that contains a SORTED BY clause cannot be updated (due to SQL/DS limitations as explained with the FIND statement). A row read with a READ LOGICAL cannot be updated either (as explained with the READ statement).

If a column is to be updated which is redefined as an array, it is strongly recommended to update the whole column and not individual occurrences; otherwise, results are not predictable. To do so, in reporting mode you can use the OBTAIN statement (as described in the Natural Statements documentation), which must be applied to all field occurrences in the column to be updated. In structured mode, however, all these occurrences must be defined in the corresponding Natural view.

The data locked by an UPDATE statement are released when an END TRANSACTION (COMMIT WORK) or BACKOUT TRANSACTION (ROLLBACK WORK) statement is executed by the program.

Note:

If a length indicator field or null indicator field is updated in a Natural program without updating the field (column) it refers to, the update of the column is not generated for SQL/DS and thus no updating takes place.

UPDATE with SELECT

In general, the DML UPDATE statement can be used in both structured and reporting mode. However, after a SELECT statement, only the syntax defined for Natural structured mode is allowed:

UPDATE [RECORD] [IN] [STATEMENT] [(r)]

This is due to the fact that in combination with the SELECT statement the DML UPDATE statement is only allowed in the special case of:

```

...
SELECT ...
  INTO VIEW view-name
...

```

Thus, only a whole Natural view can be updated; individual columns (fields) cannot.

Example:

```

DEFINE DATA LOCAL
01 PERS VIEW OF SQL-PERSONNEL
  02 NAME
  02 AGE
END-DEFINE
...
SELECT *
  INTO VIEW PERS
  FROM SQL-PERSONNEL
  WHERE NAME LIKE 'S%'
  ...
  IF NAME = 'SMITH'
    ADD 1 TO AGE
  UPDATE
  END-IF
  ...
END-SELECT
...

```

In combination with the DML UPDATE statement, any other form of the SELECT statement is rejected and an error message is returned.

In all other respects, the DML UPDATE statement can be used with the SELECT statement in the same way as with the Natural FIND statement described in the Natural Statements documentation.

Natural SQL Statements

Summarized in the following section are particular points you have to consider when using Natural SQL statements with SQL/DS. These SQL/DS-specific points partly consist in syntax enhancements which belong to the *Extended Set* of Natural SQL syntax. The Extended Set is provided in addition to the *Common Set* to support database-specific features. It also includes features not supported by SQL/DS.

- Common Syntactical Items
- COMMIT
- DELETE
- INSERT
- PROCESS SQL
- ROLLBACK
- SELECT
- UPDATE

Common Syntactical Items

The following syntactical items are either SQL/DS-specific and do not conform to the standard SQL syntax definitions (that is, to the Common Set of Natural SQL syntax) or impose restrictions when used with SQL/DS (see also SQL Statements in the Natural Statements documentation).

atom

An *atom* can be either a *parameter* (that is, a Natural program variable or host variable) or a *constant*. When running dynamically, however, the use of host variables is restricted by SQL/DS. For further details, refer to the relevant literature on SQL/DS.

comparison

The comparison operators specific to DB2 belong to the Natural Extended Set. For a description, refer to Comparison Predicate in Search Conditions, Natural SQL Statements (Statements Grouped by Functions, Natural Statements documentation).

factor

The following three factors are specific to SQL/DS and belong to the Natural Extended Set:

<i>special-register</i> <i>scalar-function (scalar-expression, ...)</i> <i>scalar-expression unit</i> <i>case-expression</i>

scalar-function

A *scalar-function* is a built-in function that can be used in the construction of scalar computational expressions. Scalar functions are specific to SQL/DS and belong to the Natural Extended Set.

The following scalar functions are supported:

CHAR
DATE
DAY
DAYS
DECIMAL
DIGITS
FLOAT
HEX
HOUR
INTEGER
LENGTH
MICROSECOND
MINUTE
MONTH
SECOND
STRIP
SUBSTR
TIME
TIMESTAMP
TRANSLATE
VALUE
VARGRAPHIC
YEAR

Each scalar function is followed by one or more scalar expressions in parentheses. The number of scalar expressions depends upon the scalar function. Multiple scalar expressions must be separated from one another by commas.

Example:

```
SELECT  
  NAME INTO NAME FROM SQL-PERSONNEL WHERE SUBSTR ( NAME, 1, 3 ) = 'Fri' ...
```

scalar-operator

The concatenation operator (CONCAT or "//") does not conform to standard SQL. It is specific to SQL/DS and belongs to the Natural Extended Set.

special-register

The following special registers do not conform to standard SQL. They are specific to SQL/DS and belong to the Natural Extended Set:

USER
CURRENT TIMEZONE
CURRENT DATE
CURRENT TIME
CURRENT TIMESTAMP

A reference to a special register returns a scalar value.

units

Units, also called "durations", are specific to SQL/DS and belong to the Natural Extended Set.

The following *units* are supported:

YEAR
YEARS
MONTH
MONTHS
DAY
DAYS
HOUR
HOURS
MINUTE
MINUTES
SECOND
SECONDS
MICROSECOND
MICROSECONDS

COMMIT

The SQL COMMIT statement indicates the end of a logical transaction and releases all SQL/DS data locked during the transaction. All data modifications are made permanent.

COMMIT is a synonym for the Natural END TRANSACTION statement.

As all cursors are closed when a logical unit of work ends, the COMMIT statement must not be placed within a database loop; instead, it has to be placed outside such a loop or after the outermost loop of nested loops.

If an external program written in another standard programming language is called from a Natural program, this external program should not contain its own COMMIT command if the Natural program issues database calls, too. The calling Natural program should issue the COMMIT statement on behalf of the external program.

DELETE

Both the "cursor-oriented" or "positioned" and the "non-cursor" or "'searched'" SQL DELETE statement are supported as part of Natural SQL; the functionality of the "positioned" DELETE statement corresponds to that of the Natural DML DELETE statement.

With SQL/DS, a table name in the FROM clause of a "searched" DELETE statement can be assigned a *correlation-name*. This does not correspond to the standard SQL syntax definition and therefore belongs to the Natural Extended Set.

INSERT

The INSERT statement is used to add one or more new rows to a table.

Since the INSERT statement can contain a select expression, all the SQL/DS-specific syntactical items described above apply.

PROCESS SQL

The PROCESS SQL statement is used to issue SQL statements to the underlying database. The statements are specified in a *statement-string*, which can also include constants and parameters.

The set of statements which can be issued is also referred to as Flexible SQL and comprises those statements which can be issued with the SQL statement "EXECUTE".

In addition, Flexible SQL includes the SQL/DS-specific statement CONNECT.

With the PROCESS SQL statement you can also specify the *statement-string* SQLDISCONNECT to release the connection to your SQL/DS application server. SQLDISCONNECT is transformed into the SQL/DS ROLLBACK WORK RELEASE command.

Execution of SQLDISCONNECT is only allowed if no transaction (logical unit of work) is open. Therefore, an explicit COMMIT (END TRANSACTION) or ROLLBACK (BACKOUT TRANSACTION) statement is required before executing SQLDISCONNECT, otherwise an error message is returned.

Note:

To avoid transaction synchronization problems between the Natural environment and SQL/DS, the COMMIT and ROLLBACK statements must not be used within PROCESS SQL.

ROLLBACK

The SQL ROLLBACK statement undoes all database modifications made since the beginning of the last logical transaction. Logical transactions can start either after the beginning of a session or after the last COMMIT/END TRANSACTION or ROLLBACK/BACKOUT TRANSACTION statement. All records held during the transaction are released.

ROLLBACK is a synonym for the Natural BACKOUT TRANSACTION statement.

As all cursors are closed when a logical unit of work ends, a BACKOUT TRANSACTION statement must not be placed within a database loop; instead, it has to be placed outside such a loop or after the outermost loop of nested loops.

If an external program written in another standard programming language is called from a Natural program, this external program should not contain its own ROLLBACK command if the Natural program issues database calls, too. The calling Natural program should issue the ROLLBACK statement on behalf of the external program.

SELECT

Cursor-Oriented Selection

Like the Natural FIND statement, the cursor-oriented SELECT statement is used to select a set of rows (records) from one or more SQL/DS tables, based on a search criterion. Since a database loop is initiated, the loop must be closed by a LOOP (reporting mode) or END-SELECT statement. With this construction, Natural uses the same database loop processing as with the FIND statement.

In addition, no cursor management is required from the application program; it is automatically handled by Natural.

Non-Cursor Selection - SELECT SINGLE

The Natural SELECT SINGLE statement provides the functionality of a non-cursor selection (singleton SELECT); that is, a select expression that retrieves at most one row without using a cursor.

Since SQL/DS supports the singleton SELECT command in static SQL only, in dynamic mode, the Natural SELECT SINGLE statement is executed like a set-level SELECT statement, which results in a cursor operation. However, Natural checks the number of rows returned by SQL/DS. If more than one row is selected, a corresponding error message is returned.

UPDATE

Both the "cursor-oriented" or "positioned" and the "non-cursor" or "'searched'" SQL UPDATE statement are supported as part of Natural SQL. Both of them reference either a table or a Natural view.

With SQL/DS, the name of a table or Natural view to be referenced by a "searched" UPDATE can be assigned a *correlation-name*. This does not correspond to the standard SQL syntax definition and, therefore, belongs to the Natural Extended Set.

The "searched" UPDATE statement must be used, for example, to update a primary key field, since SQL/DS does not allow updating of columns of a primary key by using a positioned UPDATE statement.

Note:

If you use the SET * notation, all fields of the referenced Natural view are added to the FOR UPDATE OF and SET lists. Therefore, ensure that your view contains only fields which can be updated; otherwise a negative SQLCODE is returned by SQL/DS.

Natural System Variables

When used with SQL/DS, the following restrictions apply to the following Natural system variables:

- *ISN
- *NUMBER

*ISN

As there is no SQL/DS equivalent to Adabas ISNs, the system variable *ISN is not applicable to SQL/DS tables.

*NUMBER

When used with a FIND NUMBER or HISTOGRAM statement, *NUMBER contains the number of rows actually found.

When applied to data from an SQL/DS table in any other case, the system variable *NUMBER only indicates whether any rows have been found. If no rows have been found, *NUMBER is "0". Any value other than "0" indicates that at least one row has been found; however, the value contained in *NUMBER has no relation to the number of rows actually found.

The reason is that if *NUMBER was to produce a valid number, Natural would have to translate the corresponding FIND statement into an SQL SELECT statement including the special function COUNT(*); however, a SELECT containing a COUNT function would produce a read-only result table, which would not be available for updating. In other words, the option to update selected data was given priority in Natural over obtaining the number of rows that meet the search criteria.

To obtain the number of rows affected by the Natural SQL statements "searched" UPDATE, "searched" DELETE and INSERT, the Natural subprogram NDBNROW is provided. Or you can use the Natural system variable *ROWCOUNT as described in the Natural System Variables documentation.

Error Handling

In contrast to the normal Natural error handling, where either an ON ERROR statement is used to intercept runtime errors or standard error message processing is performed and program execution is terminated, the enhanced error handling of Natural for SQL/DS provides an application-controlled reaction to the encountered SQL error.

Two Natural subprograms, NDBERR and NDBNOERR, are provided to disable the usual Natural error handling and to check the encountered SQL error for the returned SQL code. This functionality replaces the "E" function of the DB2SERV interface, which is still provided but no longer documented.

See further information on Natural subprograms provided for SQL/DS.

Example:

```

DEFINE DATA LOCAL
01 #SQLCODE           (I4)
01 #SQLSTATE         (A5)
01 #SQLCA             (A136)
01 #DBMS              (B1)
END-DEFINE
*
*           Ignore error from next statement
*
CALLNAT 'NDBNOERR'
*
*           This SQL statement produces an SQL error
*
INSERT INTO SYSIBH-SYSTABLES (CREATOR, NAME, COLCOUNT)
VALUES ('SAG', 'MYTABLE', '3')
*
*           Investigate error
*
CALLNAT 'NDBERR' #SQLCODE #SQLSTATE #SQLCA #DBMS
*
IF #DBMS NE 3                               /* not SQL/DS
MOVE 3700 TO *ERROR-NR
END-IF
*
DECIDE ON FIRST VALUE OF #SQLCODE
VALUE 0, 100                                 /* successful execution
IGNORE
VALUE -803                                   /* duplicate row
/* UPDATE existing record
/*
IGNORE
NONE VALUE
MOVE 3700 TO *ERROR-NR
END-DECIDE
*
END

```

Interface Subprograms

Several Natural and non-Natural subprograms are available to provide you with either internal information from the Natural interface to SQL/DS or specific functions that are not available within the interface itself.

From within a Natural program, Natural subprograms are invoked with the CALLNAT statement and non-Natural subprograms are invoked with the CALL statement.

This section covers the following topics:

- Natural Subprograms
- The DB2SERV Interface

Natural Subprograms

All Natural subprograms are provided in the library SYSSQL and should be copied to the SYSTEM or steplib library, or to any library where they are needed. The corresponding parameters must be defined by using either the DEFINE DATA statement in structured mode or the RESET statement in reporting mode.

The following subprograms are provided:

Subprogram	Function
NDBDBRM	Checks whether a Natural program contains SQL access and whether it has been modified for static execution.
NDBDBR2	Checks whether a Natural program contains SQL access and whether it has been modified for static execution.
NDBERR	Provides diagnostic information on the most recently executed SQL call.
NDBISQL	Executes SQL statements in dynamic mode.
NDBNOERR	Suppresses normal Natural error handling.
NDBNROW	Obtains the number of rows affected by a Natural SQL statement.
NDBSTMP	Provides an SQL/DS TIMESTAMP column as an alphanumeric field and vice versa.

The NDBDBRM Subprogram

The Natural subprogram NDBDBRM is used to check whether a Natural program contains SQL access and whether it has been modified for static execution. It is also used to obtain the corresponding package name from the header of a Natural program generated as static (see also Preparing Natural Programs for Static Execution).

A sample program called CALLDBRM is provided on the installation tape; it demonstrates how to invoke NDBDBRM. A description of the call format and of the parameters is provided in the text member NDBDBRMT.

The calling Natural program must use the following syntax:

```
CALLNAT 'NDBDBRM' #LIB #MEM #DBRM #RESP
```

The various parameters are described in the following table:

Parameter	Format/Length	Explanation
#LIB	A8	Contains the name of the library of the program to be checked.
#MEM	A8	Contains the name of the program (member) to be checked.
#DBRM	A8	Returns the package name.
#RESP	I2	Returns a response code. The possible codes are listed below.

The #RESP parameter can contain the following response codes:

Code	Explanation
0	The member #MEM in library #LIB has SQL access; it is static if #DBRM contains a value.
-1	The member #MEM in library #LIB has no SQL access.
-2	The member #MEM in library #LIB does not exist.
-3	No library name has been specified.
-4	No member name has been specified.
-5	The library name must start with a letter.
<-5	Further negative response codes correspond to error numbers of Natural error messages.
> 0	Positive response codes correspond to error numbers of Natural Security messages.

The NDBDBR2 Subprogram

The Natural subprogram NDBDBR2 is used to check whether a Natural program contains SQL access and whether it has been modified for static execution. It is also used to obtain the corresponding DBRM name from the header of a Natural program generated as static (see also Preparing Natural Programs for Static Execution) and the time stamp generated by the precompiler.

A sample program called CALLDBR2 is provided on the installation tape; it demonstrates how to invoke NDBDBR2. A description of the call format and of the parameters is provided in the text member NDBDBR2T.

The calling Natural program must use the following syntax:

```
CALLNAT 'NDBDBR2' #LIB #MEM #DBR2 #TIMESTAMP #PCUSER #PCRELLEV #ISOLLEVL #DATEFORM #TIMEFORM #RESP
```

The various parameters are described in the following table:

Parameter	Format/Length	Explanation
#LIB	A8	Contains the name of the library of the program to be checked.
#MEM	A8	Contains the name of the program (member) to be checked.
#DBR2	A8	Returns the package name.
#TIMESTAMP	B8	Consistency token generated by precompiler.
#PCUSER	A1	User ID used at precompile.
#PCRELLEV	A1	Release level of precompiler.
#ISOLLEVL	A1	Precompiler isolation level.
#DATEFORM	A1	Date format.
#TIMEFORM	A1	Time format.
#RESP	I2	Returns a response code. The possible codes are listed below.

The #RESP parameter can contain the following response codes:

Code	Explanation
0	The member #MEM in library #LIB has SQL access; it is static if #DBR2 contains a value.
-1	The member #MEM in library #LIB has no SQL access.
-2	The member #MEM in library #LIB does not exist.
-3	No library name has been specified.
-4	No member name has been specified.
-5	The library name must start with a letter.
<-5	Further negative response codes correspond to error numbers of Natural error messages.
> 0	Positive response codes correspond to error numbers of Natural Security messages.

The NDBERR Subprogram

The Natural subprogram NDBERR replaces the "E" function of the DB2SERV interface (which is still provided but no longer documented). It provides diagnostic information on the most recent SQL call. It also returns the database type which returned the error. NDBERR is typically called if a database call returns a non-zero SQL code (which means a NAT3700 error); see also Error Handling.

A sample program called CALLERR is provided on the installation tape; it demonstrates how to invoke NDBERR. A description of the call format and of the parameters is provided in the text member NDBERRT.

The calling Natural program must use the following syntax:

```
CALLNAT 'NDBERR' #SQLCODE #SQLSTATE #SQLCA #DBTYPE
```

The various parameters are described in the following table:

Parameter	Format/Length	Explanation
SQLCODE	I4	Returns the SQL return code.
SQLSTATE	A5	Returns a return code for the output of the most recently executed SQL statement.
SQLCA	A136	Returns the SQL communication area of the most recent SQL/DS access.
DBTYPE	B1	Returns the identifier (in hexadecimal format) for the currently used database (where X'03' identifies SQL/DS).

The NDBISQL Subprogram

The Natural subprogram NDBISQL is used to execute SQL statements in dynamic mode. The SELECT statement and all SQL statements which can be prepared dynamically (according to the Adabas SQL Server documentation) can be passed to NDBISQL.

A sample program called CALLISQL is provided on the installation tape; it demonstrates how to invoke NDBISQL. A description of the call format and of the parameters is provided in the text member NDBISQLT.

The calling Natural program must use the following syntax:

```
CALLNAT 'NDBISQL' #FUNCTION #TEXT-LEN #TEXT (*) #SQLCA #RESPONSE
```

The various parameters are described in the following table:

Parameter	Format/Length	Explanation
#FUNCTION	A8	For valid functions, see below.
#TEXT-LEN	I2	Length of the SQL statement or of the buffer for the return area.
#TEXT	A1(1:V)	Contains the SQL statement or receives the return code.
#SQLCA	A136	Contains the SQLCA.
#RESPONSE	I4	Returns a response code.

Valid functions for the #FUNCTION parameter are:

Function	Parameter	Explanation
CLOSE		Closes the cursor for the SELECT statement.
EXECUTE	#TEXT-LEN #TEXT (*)	Executes the SQL statement.
		Contains the length of the statement.
		Contains the SQL statement. The first two characters must be blank.
FETCH	#TEXT-LEN #TEXT (*)	Returns a record from the SELECT statement.
		Size of #TEXT (in bytes).
		Buffer for the record.
TITLE	#TEXT-LEN #TEXT (*)	Returns the header for the SELECT statement.
		Size of #TEXT (in bytes); receives the length of the header (= length of the record).
		Buffer for the header line.

The #RESPONSE parameter can contain the following response codes:

Code	Function	Explanation
5	EXECUTE	The statement is a SELECT statement.
6	TITLE, FETCH	Data are truncated; only set on first TITLE or FETCH call.
100	FETCH	No record / end of data.
-2		Unsupported data type (for example, GRAPHIC).
-3	TITLE, FETCH	No cursor open; probably invalid call sequence or statement other than SELECT.
-4		Too many columns in result table.
-5		SQL code from call.
-6		Version mismatch.
-7		Invalid function.
-10		Interface not available.
-11	EXECUTE	First two bytes of statement not blank.

Call Sequence

The first call must be an EXECUTE call. If the statement is a SELECT statement (that is, response code 5 is returned), any sequence of TITLE and FETCH calls can be used to retrieve the data. A response code of 100 indicates the end of the data.

The cursor must be closed with a CLOSE call.

Note:

Function code EXECUTE implicitly closes a cursor which has been opened by a previous EXECUTE call for a SELECT statement.

In TP environments, no terminal I/O can be performed between an EXECUTE call and any TITLE, FETCH or CLOSE call that refers to the same statement.

The NDBNOERR Subprogram

The Natural subprogram NDBNOERR is used to suppress Natural NAT3700 errors caused by the next SQL call. This allows a program controlled continuation if an SQL statement produces a non-zero SQL code. After the SQL call has been performed, NDBERR is used to investigate the SQL code; see also Error Handling.

A sample program called CALLNOER is provided on the installation tape; it demonstrates how to invoke NDBNOERR. A description of the call format and of the parameters is provided in the text member NDBNOERT.

The calling Natural program must use the following syntax:

```
CALLNAT 'NDBNOERR'
```

There are no parameters provided with this subprogram.

Note:

Only NAT3700 errors (that is, non-zero SQL response codes) are suppressed, and also only errors caused by the following SQL call.

Restrictions with Database Loops

- If NDBNOERR is called before a statement that initiates a database loop and an initialization error occurs, no processing loop will be initiated, unless the IF NO RECORDS FOUND clause has been specified.
- If NDBNOERR is called within a database loop, it does not apply to the processing loop itself, but only to the SQL statement subsequently executed inside this loop.

The NDBNROW Subprogram

The Natural subprogram NDBNROW is used to obtain the number of rows affected by the Natural SQL statements "searched" UPDATE, "searched" DELETE and INSERT. The number of rows affected is read from the SQL communication area (SQLCA). A positive value represents the number of affected rows, whereas a value of "-1" indicates that all rows of a table in a segmented tablespace have been deleted; see also *NUMBER.

A sample program called CALLNROW is provided on the installation tape; it demonstrates how to invoke NDBNROW. A description of the call format and of the parameters is provided in the text member NDBNROWT.

The calling Natural program must use the following syntax:

```
CALLNAT 'NDBNROW' #NUMBER
```

The parameter #NUMBER (I4) contains the number of rows affected.

The NDBSTMP Subprogram

For SQL/DS, Natural provides a `TIMESTAMP` column as an alphanumeric field (A26) of the format `"YYYY-MM-DD-HH.MM.SS.MMMMMM"`; see also List Columns of an SQL Table.

Since Natural does not yet support computation with such fields, the Natural subprogram `NDBSTMP` is provided to enable this kind of functionality. It converts Natural time variables to SQL/DS time stamps and vice versa, and performs SQL/DS time stamp arithmetics.

A sample program called `CALLSTMP` is provided on the installation tape which demonstrates how to invoke `NDBSTMP`. A description of the call format and of the parameters is provided in the text member `NDBSTMPT`.

The functions available are:

Code	Explanation
ADD	Adds time units (labeled durations) to a given SQL/DS time stamp and returns a Natural time variable and a new SQL/DS time stamp.
CNT2	Converts a Natural time variable (format T) into an SQL/DS time stamp (column type <code>TIMESTAMP</code>) and labeled durations.
C2TN	Converts an SQL/DS time stamp (column type <code>TIMESTAMP</code>) into a Natural time variable (format T) and labeled durations.
DIFF	Builds the difference between two given SQL/DS time stamps and returns labeled durations.
GEN	Generates an SQL/DS time stamp from the current date and time values of the Natural system variable <code>*TIMX</code> and returns a new SQL/DS time stamp.
SUB	Subtracts labeled durations from a given SQL/DS time stamp and returns a Natural time variable and a new SQL/DS time stamp.
TEST	Tests a given SQL/DS time stamp for valid format and returns <code>"TRUE"</code> or <code>"FALSE"</code> .

Note:

Labeled durations are units of year, month, day, hour, minute, second and microsecond.

The DB2SERV Interface

DB2SERV is an Assembler program entry point which can be called from within a Natural program.

DB2SERV performs either of the following functions:

- Function "D", which performs the SQL statement EXECUTE IMMEDIATE;
- Function "U", which calls the database connection services (VSE/ESA batch mode only).

The parameter or variable values returned by each of these functions are checked for their format, length and number.

Function "D"

Function "D" performs the SQL statement EXECUTE IMMEDIATE. This allows SQL statements to be issued from within a Natural program.

The SQL statement string that follows the EXECUTE IMMEDIATE statement must be assigned to the Natural program variable STMT. It must contain valid SQL statements allowed with the EXECUTE IMMEDIATE statement as described in the relevant IBM documentation. Examples can be found below and in the demonstration programs DEM2* in library SYSSQL.

Note:

The conditions that apply to issuing Natural END TRANSACTION or BACKOUT TRANSACTION statements also apply when issuing SQL COMMIT or ROLLBACK statements.

Command Syntax

```
CALL 'DB2SERV' 'D' STMT STMTL SQLCA RETCODE
```

The variables used in this command are described in the following table:

Variable	Format/Length	Explanation
STMT	<i>Annn</i>	Contains a command string which consists of SQL syntax as described above.
STMTL	I2	Contains the length of the string defined in STMT.
SQLCA	A136	Returns the current contents of the SQL communication area.
RETCODE	I2	Returns an interface return code. The following codes are possible: <ul style="list-style-type: none"> 0 No warning or error occurred. 4 SQL statement produced an SQL warning. 8 SQL statement produced an SQL error. 12 Internal error occurred;the corresponding Natural error message number can be displayed with SYSERR.

The current contents of the SQLCA and an interface return code (RETCODE) are returned. The SQLCA is a collection of variables that are used by SQL/DS to provide an application program with information on the execution of its SQL statements.

The following example shows you how to use DB2SERV with function "D":

Example of Function "D" - DEM2CREA:

```
*****
* DEM2CREA - CREATE TABLE NAT.DEMO *
*****
*
DEFINE DATA
LOCAL USING DEMSQLCA
LOCAL
*
*                               Parameters for DB2SERV
1 STMT          (A250)
1 STMTL         (I2)      CONST <250>
1 RETCODE       (I2)
*
END-DEFINE
*
COMPRESS 'CREATE TABLE NAT.DEMO'
' (NAME          CHAR(20)      NOT NULL,'
' ADDRESS        VARCHAR(100) NOT NULL,'
' DATEOFBIRTH   DATE          NOT NULL,'
' SALARY         DECIMAL(6,2),'
' REMARKS        VARCHAR(500))'
INTO STMT
CALL 'DB2SERV' 'D' STMT STMTL SQLCA RETCODE
*
END TRANSACTION
*
IF RETCODE = 0
  WRITE 'Table NAT.DEMO created'
ELSE
  FETCH 'SQLERR'
END-IF
END
*****
```

Note:

The functionality of the DB2SERV function "D" is also provided with the PROCESS SQL statement (see also the section SQL Statements in the Natural Statements documentation).

Function "U"

Function "U" calls the database connection services when running in batch mode under VSE/ESA; see also Sample Batch Verification Job (VSE/ESA only).

The user ID and password for the connection to SQL/DS must be assigned to the Natural program variables USER-ID and PASSWORD, respectively. An interface return code (RETCODE) is returned.

Command Syntax

```
the CALL 'DB2SERV' 'U' USER-ID PASSWORD RETCODE
```

The variables used in this command are described in the following table:

Variable	Format/Length	Explanation
USER-ID	A8	A Natural variable that contains the user ID for the connection to SQL/DS.
PASSWORD	A8	A Natural variable that contains the user password for the connection to SQL/DS.
RETCODE	I2	<p>A Natural variable that returns an interface return code. The following codes are possible:</p> <ul style="list-style-type: none"> 0 No warning or error occurred. 4 SQL statement produced an SQL warning. 8 SQL statement produced an SQL error. 12 Internal error occurred; information on this error can be displayed with SYSERR.

Environment-Specific Considerations

Natural for SQL/DS can be run in the TP-monitor environment CICS and in VSE/ESA batch mode.

As all dynamic access to SQL/DS is performed by NDBIOMO, all users of Natural for SQL/DS must have RUN privilege on the package NDBIOMO. If running in static mode, users must also have RUN privilege on all static SQL/DS packages.

This section covers the following topics:

- Natural for SQL/DS under CICS
 - Natural for SQL/DS in VSE/ESA Batch Mode
-

Natural for SQL/DS under CICS

Under CICS, Natural uses the SQL/DS online support to access SQL/DS. Therefore ensure that this attachment is started. If not, the Natural session is abnormally terminated with CICSabend code AEY9, which leads to Natural error message NAT0954 if the Natural profile parameter DU is set to "OFF".

Since Natural for SQL/DS does not issue any explicit CONNECT statements, it takes advantage of the implicit CONNECT facility of the SQL/DS online support.

Under CICS, a Natural program which accesses an SQL/DS table can also be run in pseudo-conversational mode. Then, at the end of a CICS task, all SQL/DS cursors are closed, and there is no way to reposition an SQL/DS cursor when the task is resumed.

To circumvent the problem of CICS terminating a pseudo-conversational transaction during loop processing and thus causing SQL/DS to close all cursors and lose all selection results, Natural switches from pseudo-conversational mode to conversational mode for the duration of a Natural loop which accesses an SQL/DS table.

To enable multiple Natural sessions to run concurrently, all Natural areas are written to the threads just before a terminal I/O operation is executed. When the terminal input is received, storage is acquired again, and all Natural areas are read from the threads.

Natural for SQL/DS in VSE/ESA Batch Mode

An explicit connection to the database must be performed. The sample program DEM2CONN can be used for this purpose. DEM2CONN calls the DB2SERV module with function code "U" which in turn calls the database connect services.