

Using Natural Statements and System Variables

This section contains special considerations concerning Natural data manipulation language (DML) statements (that is, Natural native DML statements and Natural SQL DML statements), and Natural system variables when used with SQL/DS

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

For an explanation of the symbols used in this section to describe the syntax of Natural statements, see *Syntax Symbols* in the *Natural Statements* documentation.

For information on logging SQL statements contained in a Natural program, refer to *DBLOG Trace Screen for SQL Statements* in the *DBLOG Utility* documentation.

This section covers the following topics:

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

Using Natural Native DML Statements

This section summarizes particular points you have to consider when using Natural native data manipulation language (DML) statements with SQL/DS. Any Natural statement not mentioned in this section can be used with SQL/DS without restriction.

Below is information on the following Natural DML statements:

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

- STORE
- UPDATE

BACKOUT TRANSACTION

The Natural native DML statement `BACKOUT TRANSACTION` 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.

How the statement is translated and which command is actually issued depends on the environment:

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, see *Natural for DB2 under CICS*.

In batch mode, the `BACKOUT TRANSACTION` statement is translated into an `SQL ROLLBACK` command.

Note:

Be aware that with terminal input in SQL/DS database loops, Natural switches to conversational mode if no file server is used.

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 must not contain its own `ROLLBACK` command if the Natural program issues database calls, too. The calling Natural program must issue the `BACKOUT TRANSACTION` statement for the external program.

DELETE

The Natural native DML statement `DELETE` is used to delete a row from a DB2 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 CURSOR1 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 Natural native DML DELETE statement into a Natural SQL DELETE statement in the same way it translates a Natural native DML FIND statement into a Natural SQL SELECT statement.

A row read with a FIND SORTED BY cannot be deleted due to DB2 restrictions explained with the FIND statement. A row read with a READ LOGICAL cannot be deleted either.

END TRANSACTION

The Natural native DML statement END TRANSACTION indicates the end of a logical transaction and releases all DB2 data locked during the transaction. All data modifications are committed and made permanent.

How the statement is translated and which command is actually issued depends on the environment:

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

In batch mode, the END TRANSACTION statement is translated into an SQL COMMIT WORK 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 must not contain its own COMMIT command if the Natural program issues database calls, too. The calling Natural program must 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 Natural native DML statement FIND corresponds to the Natural SQL statement SELECT.

Example:

Natural native DML statements:

```

FIND EMPLOYEES WITH NAME = 'BLACKMORE'
  AND AGE EQ 20 THRU 40
OBTAIN PERSONNEL_ID NAME AGE

```

Equivalent Natural SQL statement:

```
SELECT PERSONNEL_ID, NAME, AGE
FROM EMPLOYEES
WHERE NAME = 'BLACKMORE'
AND AGE BETWEEN 20 AND 40
```

Natural internally translates a FIND statement into an SQL SELECT statement as described in *Processing of SQL Statements Issued by Natural* in the section *Internal Handling of Dynamic Statements*. The SELECT statement is executed by an OPEN CURSOR statement 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.

The WITH clause of a FIND statement is converted to the WHERE clause of the SELECT statement. The basic search criterion for an 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 Natural *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 as described in the Natural *System Variables* documentation.

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 the 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 as described in the section *Using Natural SQL Statements*.

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 time. If this limit is exceeded, a Natural error message is returned.

GET

The Natural native DML statement GET is based on Adabas internal sequence numbers (ISNs) and therefore cannot be used with SQL/DS tables.

HISTOGRAM

The Natural native DML statement HISTOGRAM 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 as described in the Natural *System Variables* documentation.

Example:

Natural native DML statements:

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

Equivalent Natural 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 Natural native DML statement READ can also be used to access SQL/DS tables. Natural translates a READ statement into a Natural SQL SELECT statement.

READ PHYSICAL and READ LOGICAL can be used; READ BY ISN, however, cannot be used, as there is no DB2 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). The start value can only be a constant or a 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).

Example 1:

The Natural native DML statements:

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

Equivalent Natural SQL statement:

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

Example 2:

The Natural native DML statements:

```
READ PERSONNEL PHYSICAL
OBTAIN NAME
```

Equivalent Natural 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 in the search criterion.

STORE

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

Example:

The Natural native DML statement:

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

Equivalent Natural SQL statement:

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

The PASSWORD, CIPHER and USING/GIVING NUMBER clauses of the STORE statement cannot be used.

UPDATE

The Natural native DML statement UPDATE 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 Natural native DML statement FIND, 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 DB2 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, a DB2 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 (see also Natural SQL `UPDATE` statement in the section *Using Natural SQL Statements*).

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` statement 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, 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 Natural native DML statement `UPDATE` 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 Natural native 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 Natural native DML `UPDATE` statement, any other form of the `SELECT` statement is rejected and an error message is returned.

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

Using Natural SQL Statements

This section covers 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; see *Common Set and Extended Set* in the section *SQL Statements* in the *Natural Statements* documentation. It also includes features not supported by SQL/DS.

For information on logging SQL statements contained in a Natural program, refer to *DBLOG Trace Screen for SQL Statements* in the *DBLOG Utility* documentation.

Below is information on the following Natural SQL statements and on common syntactical items:

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

Syntactical Items Common to Natural SQL Statements

The following common 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).

Below is information on the following common syntactical items:

- atom
- comparison
- factor
- scalar-function
- scalar-operator
- special-register
- units

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 SQL/DS literature by IBM.

comparison

The comparison operators specific to SQL/DS belong to the Natural Extended Set. For a description, refer to *Comparison Predicate* in *Search Condition*, *Natural SQL Statements* in the *Natural Statements* documentation.

factor

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

<i>special-register</i> <i>scalar-function</i> (<i>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 SQL 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 SQL 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 SQL Extended Set.

The following units are supported:

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

COMMIT - SQL

The Natural SQL statement `COMMIT` 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 native DML statement `END TRANSACTION` as described in the section *Using Natural Native DML Statements*.

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 must not contain its own `COMMIT` command if the Natural program issues database calls, too. The calling Natural program must issue the `COMMIT` statement on behalf of the external program.

For further details and statement syntax, see *COMMIT - SQL* in the *Natural Statements* documentation.

DELETE - SQL

Both the cursor-oriented or Positioned `DELETE`, and the non-cursor or Searched `DELETE` statements are supported as part of Natural SQL; the functionality of the Positioned `DELETE` statement corresponds to that of the Natural native 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 SQL Extended Set.

For further details and statement syntax, see *DELETE - SQL* in the *Natural Statements* documentation.

INSERT - SQL

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

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

For further details and statement syntax, see *INSERT - SQL* in the *Natural Statements* documentation.

PROCESS SQL

The Natural SQL statement `PROCESS SQL` 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 following 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`.

For further details and statement syntax, see *PROCESS SQL* in the *Natural Statements* documentation.

ROLLBACK - SQL

The Natural 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 statement `BACKOUT TRANSACTION` as described in the section *Using Natural Native DML Statements*.

As all cursors are closed when a logical unit of work ends, a `ROLLBACK` 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 must not contain its own `ROLLBACK` command if the Natural program issues database calls, too. The calling Natural program must issue the `ROLLBACK` statement on behalf of the external program.

For further details and statement syntax, see *ROLLBACK -SQL* in the *Natural Statements* documentation.

SELECT - SQL

The Natural SQL `SELECT` statement supports both the cursor-oriented selection, which is used to retrieve an arbitrary number of rows, and the non-cursor selection (Singleton `SELECT`), which retrieves at most one single row.

SELECT - Cursor-Oriented

Like the Natural native DML `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` statement (in reporting mode) or by an `END-SELECT` statement (in structured mode). With this construction, Natural uses the same loop processing as with the `FIND` statement. In addition, no cursor management is required from the application program; it is automatically handled by Natural.

For further details and syntax, see *SELECT SQL, Syntax 1 - Cursor-Oriented Selection* in the *Natural Statements* documentation.

SELECT SINGLE - Non-Cursor-Oriented

The Natural SQL statement `SELECT SINGLE` 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.

For further details and syntax, see *SELECT SQL, Syntax 2 - Non-Cursor Selection* in the Natural *Statements* documentation.

UPDATE - SQL

Both the cursor-oriented or Positioned `UPDATE` and the non-cursor or Searched `UPDATE` statements 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.

For further details and syntax, see *UPDATE - SQL* in the Natural *Statements* documentation.

Using Natural System Variables

When used with DB2, there are restrictions and/or special considerations concerning the following Natural system variables:

- `*ISN`
- `*NUMBER`
- `*ROWCOUNT`

For information on restrictions and/or special considerations, refer to the section *Database-Specific Information* in the corresponding system variable documentation.

Error Handling

In contrast to the normal Natural error handling, where either an `ON ERROR` statement is used to intercept execution time errors or standard error message processing is performed and program execution is terminated, the enhanced error handling of Natural for DB2 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.

For further information on Natural subprograms provided for SQL/DS, see the section *Interface Subprograms*.

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 2                               /* not DB2
  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

```