Accessing Data in an SQL Database

This chapter describes how to use Natural with SQL databases via Entire Access. For information about installation and configuration, see *Natural and Entire Access* in the *Database Management System Interfaces* documentation and the separate Entire Access documentation.

This chapter covers the following topics:

- Generating Natural DDMs
- Setting Natural Profile Parameters
- Natural DML Statements
- Natural SQL Statements
- Flexible SQL
- RDBMS-Specific Requirements and Restrictions
- Data-Type Conversion
- Date/Time Conversion
- Obtaining Diagnostic Information about Database Errors
- SQL Authorization

Note:
On principle, the features and examples contained in the document *Accessing Data in an Adabas Database* also apply to the SQL databases supported by Natural. Differences, if any, are described in the documents for the individual database access statements (see the *Statements* documentation) in paragraphs named *Database-Specific Considerations* or in the documents for the individual Natural parameters (see the *Parameter Reference*). In addition, Natural offers a specific set of statements to access SQL databasess.

Generating Natural DDMs

Entire Access is an application programming interface (API) that supports Natural SQL statements and most Natural DML statements.

Natural DML and SQL statements can be used in the same Natural program. At compilation, if a DML statement references a DDM for a data source defined in *NATCONF.CFG* with DBMS type `SQL`, Natural translates the DML statement into an SQL statement.

Natural converts DML and SQL statements into calls to Entire Access. Entire Access converts the requests to the data formats and SQL dialect required by the target RDBMS and passes the requests to the database driver.

Setting Natural Profile Parameters

ETEOP Parameter

This parameter can be set only by Natural administrators.

The Natural profile parameter ETEOP controls transaction processing during a Natural session. It is required, for example, if a single logical transaction is to span two or more Natural programs. In this case, Natural must not issue an END TRANSACTION command (that is, not "commit") at the termination of a Natural program.

If the ETEOP parameter is set to:

ON	Natural issues an END TRANSACTION statement (that is, automatically "commits") at the end of a Natural program if the Natural session is not at ET status.
OFF	Natural does not issue an END TRANSACTION command (that is, does not "commit") at the end of a Natural program. This setting thus enables a single logical transaction to span more than one Natural program. This is the default.

Note:
The ETEOP parameter applies to Natural Version 6.1 and above. With previous Natural versions, the Natural profile parameter OPRB has to be used instead of ETEOP (ETEOP=ON corresponds to OPRB=OFF, ETEOP=OFF corresponds to ORPB=NOOPEN).

Natural DML Statements

The following table shows how Natural translates DML statements into SQL statements:

DML Statement	SQL Statement
BACKOUT TRANSACTION	ROLLBACK
DELETE	DELETE WHERE CURRENT OF *cursor-name*
END TRANSACTION	COMMIT
EQUAL ... OR	IN (...)
EQUAL ... THRU ...	BETWEEN ... AND ...
FIND ALL	SELECT
FIND NUMBER	SELECT COUNT (*)
HISTOGRAM	SELECT COUNT (*)
READ LOGICAL	SELECT ... ORDER BY
READ PHYSICAL	SELECT ... ORDER BY
SORTED BY ... [DESCENDING]	ORDER BY ... [DESCENDING]
STORE	INSERT
UPDATE	UPDATE WHERE CURRENT of *cursor-name*
WITH	WHERE

Note:
Boolean and relational operators function the same way in DML and SQL statements.

Entire Access does not support the following DML statements and options:

- CIPHER
- COUPLED
- FIND FIRST, FIND UNIQUE, FIND ... RETAIN AS
- GET, GET SAME, GET TRANSACTION DATA, GET RECORD
- PASSWORD
- READ BY ISN
- STORE USING/GIVING NUMBER

BACKOUT TRANSACTION

Natural translates a BACKOUT TRANSACTION statement into an SQL ROLLBACK command. This statement reverses all database modifications made after the completion of the last recovery unit. A recovery unit may start at the beginning of a session or after the last END TRANSACTION (COMMIT) or BACKOUT TRANSACTION (ROLLBACK) statement.

Note:
Because all cursors are closed when a logical unit of work ends, do not place a BACKOUT TRANSACTION statement within a database loop; place it outside the loop or after the outermost loop of

nested loops.

DELETE

The DELETE statement deletes a row from a database table that 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 last row that was read can be deleted.

Example:

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

Natural translates the Natural statements above into the following SQL statements and assigns a cursor name (for example, CURSOR1). The SELECT statement and the DELETE statement refer to the same cursor.

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

Natural translates a DELETE statement into an SQL DELETE statement the way it translates a FIND statement into an SQL SELECT statement. For details, see the FIND statement description below.

Note:
You cannot delete a row read with a FIND SORTED BY or READ LOGICAL statement. For an explanation, see the FIND and READ statement descriptions below.

END TRANSACTION

Natural translates an END TRANSACTION statement into an SQL COMMIT command. The END TRANSACTION statement indicates the end of a logical transaction, commits all modifications to the database, and releases data locked during the transaction.

Notes:

1. Because all cursors are closed when a logical unit of work ends, do not place an END TRANSACTION statement within a database loop; place it outside the loop or after the outermost loop of nested loops.
2. The END TRANSACTION statement cannot be used to store transaction (ET) data when used with Entire Access.
3. Entire Access does not issue a COMMIT automatically when the Natural program terminates.

FIND

Natural 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 all records have been read or the program exits the FIND processing loop. A CLOSE CURSOR command ends the SELECT processing.

Example:

Natural statements:

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

Equivalent SQL statement:

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

You can use any table column (field) designated as a descriptor to construct search criteria.

Natural translates the `WITH` clause of a `FIND` statement into the `WHERE` clause of an SQL `SELECT` statement. Natural evaluates the `WHERE` clause of the `FIND` statement after the rows have been selected using the `WITH` clause. View fields may be used in a `WITH` clause only if they are designated as descriptors.

Natural translates a `FIND NUMBER` statement into an SQL `SELECT` statement containing a `COUNT(*)` clause. When you want to determine whether a record exists for a specific search condition, the `FIND NUMBER` statement provides better performance than the `IF NO RECORDS FOUND` clause.

Note:
A row read with a `FIND` statement containing a `SORTED BY` clause cannot be updated or deleted. Natural translates the `SORTED BY` clause of a `FIND` statement into the `ORDER BY` clause of an SQL `SELECT` statement, which produces a read-only result table.

HISTOGRAM

Natural translates the `HISTOGRAM` statement into an SQL `SELECT` statement. The `HISTOGRAM` statement returns the number of rows in a table that 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 statements:

```
SELECT AGE, COUNT(*) FROM EMPLOYEES
  GROUP BY AGE
  ORDER BY AGE
```

READ

Natural translates a READ statement into an SQL SELECT statement. Both READ PHYSICAL and READ LOGICAL statements can be used.

A row read with a READ LOGICAL statement (Example 1) cannot be updated or deleted. Natural translates a READ LOGICAL statement into the ORDER BY clause of an SQL SELECT statement, which produces a read-only result table.

A READ PHYSICAL statement (Example 2) can be updated or deleted. Natural translates it into a SELECT statement without an ORDER BY clause.

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
```

When a READ statement contains a WHERE clause, Natural evaluates the WHERE clause after the rows have been selected according to the search criterion.

STORE

The STORE statement adds a row to a database table. It corresponds to the SQL INSERT statement.

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')
```

UPDATE

The DML UPDATE statement updates a table row that has been read with a preceding FIND, READ, or SELECT statement. Natural translates the DML UPDATE statement into the SQL statement UPDATE WHERE CURRENT OF *cursor-name* (a positioned UPDATE statement), which means that only the last row that was read can be updated. In the case of nested loops, the last row in each nested loop can be updated.

UPDATE with FIND/READ

When a DML UPDATE statement is used after a Natural FIND statement, Natural translates the FIND statement into an SQL SELECT statement with a FOR UPDATE OF clause, and translates the DML UPDATE statement into an UPDATE WHERE CURRENT OF *cursor-name* statement.

Example:

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

Natural translates the Natural statements above into the following SQL statements and assigns a cursor name (for example, CURSOR1). The SELECT and UPDATE statements refer to the same cursor.

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

You cannot update a row read with a FIND SORTED BY or READ LOGICAL statement. For an explanation, see the FIND and READ statement descriptions above.

An END TRANSACTION or BACKOUT TRANSACTION statement releases data locked by an UPDATE statement.

UPDATE with SELECT

The DML UPDATE statement can be used after a SELECT statement only in the following case:

```
SELECT *
  INTO VIEW view-name
```

Natural rejects any other form of the SELECT statement used with the DML UPDATE statement. Natural translates the DML UPDATE statement into a non-cursor or "searched" SQL UPDATE statement, which means than only an entire Natural view can be updated; individual columns cannot be updated.

In addition, the DML UPDATE statement can be used after a SELECT statement only in Natural structured mode, which has the following syntax:

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

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%'
  OBTAIN NAME
    IF NAME = 'SMITH'
      ADD 1 TO AGE
    UPDATE
    END-IF
END-SELECT
```

In other respects, the DML `UPDATE` statement works with the `SELECT` statement the way it works with the Natural `FIND` statement (see *UPDATE with FIND/READ* above).

Natural SQL Statements

The SQL statements available within the Natural programming language comprise two different sets of statements: the common set and the extended set. On this platform, only the extended set is supported by Natural.

The common set can be handled by each SQL-eligible database system supported by Natural. It basically corresponds to the standard SQL syntax definitions. For a detailed description of the common set of Natural SQL statements, see *Common Set and Extended Set* (in the *Statements* documentation).

This section describes considerations and restrictions when using the common set of Natural SQL statements with Entire Access.

- DELETE
- INSERT
- PROCESS SQL
- SELECT
- UPDATE

DELETE

The Natural SQL `DELETE` statement deletes rows in a table without using a cursor.

Whereas Natural translates the DML `DELETE` statement into a positioned `DELETE` statement (that is, an SQL `DELETE WHERE CURRENT OF` *`cursor-name`* statement), the Natural SQL `DELETE` statement is a non-cursor or searched `DELETE` statement. A searched `DELETE` statement is a stand-alone statement unrelated to any `SELECT` statement.

INSERT

The `INSERT` statement adds rows to a table; it corresponds to the Natural `STORE` statement.

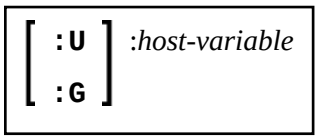
PROCESS SQL

The `PROCESS SQL` statement issues SQL statements in a *statement-string* to the database identified by a *ddm-name*.

Note:
It is not possible to run database loops using the `PROCESS SQL` statement.

Parameters

Natural supports the `INDICATOR` and `LINDICATOR` clauses. As an alternative, the *statement-string* may include parameters. The syntax item *parameter* is syntactically defined as follows:



A *host-variable* is a Natural program variable referenced in an SQL statement.

SET SQLOPTION *option=value*

With Entire Access, you can also specify `SET SQLOPTION` *`option=value`* as *statement-string*. This can be used to specify various options for accessing SQL databases. The options apply only to the database referenced by the `PROCESS SQL` statement.

Supported options are:

- `DATEFORMAT`
- `DBPROCESS` (for Sybase only)
- `TIMEOUT` (for Sybase only)
- `TRANSACTION` (for Sybase only)

DATEFORMAT

This option specifies the format used to retrieve SQL Date and Datetime information into Natural fields of type A. The option is obsolete if Natural fields of type D or T are used. A subset of the Natural date and time edit masks can be used:

YYYY	Year (4 digits)
YY	Year (2 digits)
MM	Month
DD	Day
HH	Hour
II	Minute
SS	Second

If the date format contains blanks, it must be enclosed in apostrophes.

Examples:

To use ISO date format, specify

```
PROCESS SQL sql-ddm << SET SQLOPTION DATEFORMAT = YYYY-MM-DD >>
```

To obtain date and time components in ISO format, specify

```
PROCESS SQL sql-ddm << SET SQLOPTION DATEFORMAT = 'YYYY-MM-DD HH:II:SS'    >>
```

The DATEFORMAT is evaluated only if data are retrieved from the database. If data are passed to the database, the conversion is done by the database system. Therefore, the format specified with DATEFORMAT should be a valid date format of the underlying database.

If no DATEFORMAT is specified for Natural fields,

- the default date format DD-MON-YY is used (where MON is a 3-letter abbreviation of the English month name) and
- the following default datetime formats are used:

Adabas D	YYYYMMDDHHIISS
DB2	YYYY-MM-DD-HH.II.SS
INFORMIX	YYYY-MM-DD HH:II:SS
ODBC	YYYY-MM-DD HH:II:SS
ORACLE	YYYYMMDDHHIISS
SYBASE DBLIB	YYYYMMDD HH:II:SS
SYBASE CTLIB	YYYYMMDD HH:II:SS
Microsoft SQL Server	YYYYMMDD HH:II:SS
other	DD-MON-YY

DBPROCESS

This option is valid for Sybase and Microsoft SQL Server databases only.

This option is used to influence the allocation of SQL statements to Sybase and Microsoft SQL Server DBPROCESSes. DBPROCESSes are used by Entire Access to emulate database cursors, which are not provided by the Sybase and Microsoft SQL Server DBlib interface.

Two values are possible:

MULTIPLE	With DBPROCESS set to MULTIPLE, each SELECT statement uses its own secondary DBPROCESS, whereas all other SQL statements are executed within the primary DBPROCESS. The value MULTIPLE therefore enables your application to execute further SQL statements, even if a database loop is open. It also allows nested database loops.
SINGLE	With DBPROCESS set to SINGLE, all SQL statements use the same (that is, the primary) DBPROCESS. It is therefore not possible to execute a new database statement while a database loop is active, because one DBPROCESS can only execute one SQL batch at a time. Since all statements are executed in the same (primary) DBPROCESS, however, this setting enables SELECTions from non-shared temporary tables.

Notes:

1. The specified value can only be changed if no database loop is active.
2. As the DBPROCESS option only applies to the Sybase and Microsoft SQL Server DBlib interface, your application should use a central CALLNAT statement to change the value (at least for SINGLE), so that you can easily remove these calls once Sybase client libraries are supported. Your application should also use a central error handling that establishes the default setting (MULTIPLE).

TIMEOUT

This option is valid for Sybase and Microsoft SQL Server databases only.

With Sybase and Microsoft SQL Server, Entire Access uses a timeout technique to detect database-access deadlocks. The default timeout period is 8 seconds. With this option, you can change the duration of the timeout period (in seconds).

For example, to set the timeout period to 30 seconds, specify

```
PROCESS SQL sql-ddm << SET SQLOPTION TIMEOUT = 30 >>
```

TRANSACTION

This option is valid for Sybase and Microsoft SQL Server databases only.

This option is used to enable or disable transaction mode. It becomes effective after the next END TRANSACTION or BACKOUT TRANSACTION statement.

If transaction mode is enabled (this is the default), Natural automatically issues all required statements to begin a transaction.

Examples:

To disable transaction mode, specify

```
PROCESS SQL sql-ddm << SET SQLOPTION TRANSACTION = NO >>
...
END TRANSACTION
```

To enable transaction mode, specify

```
PROCESS SQL sql-ddm << SET SQLOPTION TRANSACTION = YES >>
...
END TRANSACTION
```

SQLDISCONNECT

With Entire Access, you can also specify `SQLDISCONNECT` as the *statement-string*. In combination with the `SQLCONNECT` statement (see below), this statement can be used to access different databases by one application within the same session, by simply connecting and disconnecting as required.

A successfully performed `SQLDISCONNECT` statement clears the information previously provided by the `SQLCONNECT` statement; that is, it disconnects your application from the currently connected SQL database determined by the DBID of the DDM used in the `PROCESS SQL` statement. If no connection is established, the `SQLDISCONNECT` statement is ignored. It will fail if a transaction is open.

Note:
If Natural reports an error in the `SQLDISCONNECT` statement, the connection status does not change. If the database reports an error, the connection status is undefined.

SQLCONNECT *option=value*

With Entire Access, you can also specify `SQLCONNECT` *`option=value`* as the *statement-string*. This statement can be used to establish a connection to an SQL database according to the DBID specified in the DDM addressed by the `PROCESS SQL` statement. The `SQLCONNECT` statement will fail if the specified connection is already established.

Supported options are:

- `USERID`
- `PASSWORD`
- `OS_PASSWORD`
- `OS_USERID`
- `DBMS_PARAMETER`

Notes:

1. If the `SQLCONNECT` statement fails, the connection status does not change.
2. If several options are specified, they must be separated by a comma.
3. The specified value can be either a character literal or a Natural variable of format A.
4. If Natural performs an implicit reconnect, because the connection to the database was lost, the values provided by the `SQLCONNECT` statement are used.

The options are evaluated as described below.

USERID and PASSWORD

Specifying USERID and PASSWORD for the database logon suppresses the default logon window and the evaluation of the environment variables SQL_DATABASE_USER and SQL_DATABASE_PASSWORD.

If only USERID is specified, PASSWORD is assumed to be blank, and vice versa.

If neither USERID nor PASSWORD is specified, default logon processing applies.

Note:
With database systems that do not require user ID and password, a blank user ID and password can be specified to suppress the default logon processing.

OS_USERID and OS_PASSWORD

Specifying OS_PASSWORD and OS_USERID for the operating system logon suppresses the logon window and the evaluation of the environment variables SQL_OS_USER and SQL_OS_PASSWORD.

If only OS_USERID is specified, OS_PASSWORD is assumed to be blank, and vice versa.

If neither OS_USERID nor OS_PASSWORD is specified, default logon processing applies.

Note:
With operating systems that do not require user ID and password, a blank user ID and password can be specified to suppress the default logon processing.

DBMS_PARAMETER

Specifying DBMS_PARAMETER dynamically overwrites the DBMS assignment in the Natural global configuration file.

Examples:

```
PROCESS SQL sql-ddm << SQLCONNECT USERID = 'DBA', PASSWORD = 'SECRET'    >>
```

This example connects to the database specified in the Natural global configuration file with user ID DBA and password SECRET.

```
DEFINE DATA LOCAL
1 #UID (A20)
1 #PWD (A20)
END-DEFINE
INPUT 'Please enter ADABAS D user ID and password' / #UID / #PWD
PROCESS SQL sql-ddm << SQLCONNECT USERID = : #UID,
                       PASSWORD          = : #PWD,
                       DBMS_PARAMETER    = 'ADABASD:mydb'
                    >>
```

This example connects to the Adabas D database mydb with the user ID and password taken from the INPUT statement.

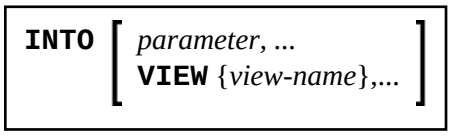
```
PROCESS SQL sql-ddm << SQLCONNECT USERID = ' ', PASSWORD = ' ',
                      DBMS_PARAMETER    = 'DB2:EXAMPLE' >>
```

This example connects to the DB2 database EXAMPLE without specifying user ID and password (since these are not required by DB2 which uses the operating system user ID).

SELECT

The INTO clause and scalar operators for the SELECT statement either are RDBMS-specific and do not conform to the standard SQL syntax definitions (the Natural common set), or impose restrictions when used with Entire Access.

Entire Access does not support the INDICATOR and LINDICATOR clauses in the INTO clause. Thus, Entire Access requires the following syntax for the INTO clause:



Note:
The concatenation operator (||) does not belong to the common set and is therefore not supported by Entire Access.

SELECT SINGLE

The SELECT SINGLE statement provides the functionality of a non-cursor SELECT operation (singleton SELECT); that is, a SELECT statement that retrieves a maximum of one row without using a cursor.

This statement is similar to the Natural FIND UNIQUE statement. However, Natural automatically checks the number of rows returned. If more than one row is selected, Natural returns an error message.

If your RDBMS does not support dynamic execution of a non-cursor SELECT operation, the Natural SELECT SINGLE statement is executed like a set-level SELECT statement, which results in a cursor operation. However, Natural still checks the number of returned rows and issues an error message if more than one row is selected.

UPDATE

The Natural SQL UPDATE statement updates rows in a table without using a cursor.

Whereas Natural translates the DML UPDATE statement into a positioned UPDATE statement (that is, the SQL DELETE WHERE CURRENT OF *cursor-name* statement), the Natural SQL UPDATE statement is a non-cursor or searched UPDATE statement. A searched UPDATE statement is a stand-alone statement unrelated to any SELECT statement.

Flexible SQL

Flexible SQL allows you to use arbitrary RDBMS-specific SQL syntax extensions. Flexible SQL can be used as a replacement for any of the following syntactical SQL items:

- atom
- column reference
- scalar expression
- condition

The Natural compiler does not recognize the SQL text used in flexible SQL; it simply copies the SQL text (after substituting values for the host variables, which are Natural program variables referenced in an SQL statement) into the SQL string that it passes to the RDBMS. Syntax errors in flexible SQL text are detected at runtime when the RDBMS executes the string.

Note the following characteristics of flexible SQL:

- It is enclosed in << and >> characters and can include arbitrary SQL text and host variables.
- Host variables must be prefixed by a colon (:).
- The SQL string can cover several statement lines; comments are permitted.

Flexible SQL can also be used between the clauses of a select expression:

```
SELECT selection
  << ... >>
  INTO ...
  FROM ...
  << ... >>
  WHERE ...
  << ... >>
  GROUP BY ...
  << ... >>
  HAVING ...
  << ... >>
  ORDER BY ...
  << ... >>
```

Examples:

```
SELECT NAME
FROM EMPLOYEES
WHERE << MONTH (BIRTH) >> = << MONTH (CURRENT_DATE) >>

SELECT NAME
FROM EMPLOYEES
WHERE << MONTH (BIRTH) = MONTH (CURRENT_DATE) >>

SELECT NAME
FROM EMPLOYEES
WHERE SALARY > 50000
<< INTERSECT
   SELECT NAME
   FROM EMPLOYEES
   WHERE DEPT = 'DEPT10'
>>
```

RDBMS-Specific Requirements and Restrictions

This section discusses restrictions and special requirements for Natural and some RDBMSs used with Entire Access.

The following topics are covered:

- Case-Sensitive Database Systems
- SYBASE and Microsoft SQL Server

Case-Sensitive Database Systems

In case-sensitive database systems, use lower-case characters for table and column names, as all names specified in a Natural program are automatically converted to lower-case.

Note:
This restriction does not apply when you use flexible SQL.

SYBASE and Microsoft SQL Server

To execute SQL statements against SYBASE and Microsoft SQL Server, you must use one or more `DBPROCESS` structures. A `DBPROCESS` can execute SQL command batches.

A command batch is a sequence of SQL statements. Statements must be executed in the sequence in which they are defined in the command batch. If a statement (for example, a `SELECT` statement) returns a result, you must execute the statement first and then fetch the rows one by one. Once you execute the next statement from the command batch, you can no longer fetch rows from the previous query.

With SYBASE and Microsoft SQL Server, an application can use more than one `DBPROCESS` structure; therefore, it is possible to have nested queries if you use a separate `DBPROCESS` for each query. Because SYBASE and Microsoft SQL Server lock data for each `DBPROCESS`, however, an application that uses more than one `DBPROCESS` can deadlock itself. Natural times out in case of a deadlock.

The following topics are covered below:

- How Natural Statements are Converted to Database Calls
- Natural Restrictions with SYBASE and Microsoft SQL Server

How Natural Statements are Converted to Database Calls

Natural uses one `DBPROCESS` for each open query and another `DBPROCESS` for all other SQL statements (`UPDATE`, `DELETE`, `INSERT`, ...).

If a query is referenced by a positioned `UPDATE` or `DELETE` statement, Natural automatically appends the `FOR BROWSE` clause to the generated `SELECT` statement to allow `UPDATE`s while rows are being read.

For a positioned `UPDATE` or `DELETE` statement, the SYBASE `dbqual` function is used to generate the following search condition:

```
WHERE unique-index = value AND tsequal (timestamp,old-timestamp)
```

This search condition can be used to reselect the current row from the query. The `tsequal` function checks whether the row has been updated by another user.

Natural Restrictions with SYBASE and Microsoft SQL Server

The following restrictions apply when using Natural with SYBASE and Microsoft SQL Server.

Case-Sensitivity

SYBASE and Microsoft SQL Server are case-sensitive, and Natural passes parameters in lowercase. Thus, if your SYBASE and Microsoft SQL Server tables or fields are defined in uppercase or mixed case, you must use database `SYNONYM`s or Natural flexible SQL.

Positioned UPDATE and DELETE Statements

To support positioned `UPDATE` and `DELETE` statements, the table to be accessed must have a unique index and a timestamp column. In addition, the timestamp column must not be included in the select list of the query.

Querying Rows

SYBASE and Microsoft SQL Server lock pages, and locked pages are owned by `DBPROCESS` structures.

Pages locked by an active `DBPROCESS` cannot subsequently be read (by the same or another `DBPROCESS`) until the lock is released by an `END TRANSACTION` or `BACKOUT TRANSACTION` statement.

Therefore, if you have updated, inserted, or deleted a row in a table:

- Do not start a new `SELECT` (`FIND`, `READ`, ...) loop against the same table.
- Do not fetch additional rows from a query that references the same table if the `SELECT` statement has no `FOR BROWSE` clause.

Natural automatically appends the `FOR BROWSE` clause if the query is referenced by a positioned `UPDATE` or `DELETE` statement.

Transaction/Non-Transaction Mode

SYBASE and Microsoft SQL Server differentiate between transaction and non-transaction mode. In transaction mode, Natural connects to the database allowing `INSERT`s, `UPDATE`s and `DELETE`s to be issued; thus, commands that run in non-transaction mode, for example, `CREATE TABLE`, cannot be issued.

Stored Procedures

It is possible to use stored procedures in SYBASE and Microsoft SQL Server using the `PROCESS SQL` statement. However, the stored procedures must not contain

- commands that work only in non-transaction mode; or
- return values.

Data-Type Conversion

When a Natural program accesses data in a relational database, Entire Access converts RDBMS-specific data types to Natural data formats, and vice versa. The RDBMS data types and their corresponding Natural data formats are described in the *Editors* documentation under *Data Conversion for RDBMS* (in the section *DDM Editors*.

The date/time or datetime format specific to a particular database can be converted into the Natural formats D and T; see below.

Date/Time Conversion

The RDBMS-specific date/time or datetime format can be converted into the Natural formats D and T.

To use this conversion, you first have to edit the Natural DDM to change the date or time field formats from A(lphanumeric) to D(ate) or T(ime). The `SQLOPTION DATEFORMAT` is obsolete for fields with format D or T.

Note:
Date or time fields converted to Natural D(ate)/T(ime) format may not be mixed with those converted to Natural A(lphanumeric) format.

- For update commands, Natural converts the Natural Date and Time format to the database-dependent representation of `DATE`/`TIME`/`DATETIME` to a precision level of seconds.
- For retrieval commands, Natural converts the returned database-dependent character representation to the internal Natural Date or Time format; see conversion tables below. The date component of Natural Time is not ignored and is initialized to `0000-01-02` (`YYYY-MM-DD`) if the RDBMS‘s time format does not contain a date component.
- For Natural Date variables, the time portion is ignored and initialized to zero.
- For Natural Time variables, tenth of seconds are ignored and initialized to zero.

Conversion Tables

Adabas D

RDBMS Formats	Natural Date	Natural Time
`DATE`	`YYYYMMDD`	
`TIME`		`00HHIISS`

DB2

RDBMS Formats	Natural Date	Natural Time
DATE	YYYY-MM-DD	
TIME		HH.II.SS

INFORMIX

RDBMS Formats	Natural Date	Natural Time
DATETIME, year to day	YYYY-MM-DD	
DATETIME, year to second (other formats are not supported)		YYYY-MM-DD-HH:II:SS*

ODBC

RDBMS Formats	Natural Date	Natural Time
DATE	YYYY-MM-DD	
TIME		HH:II:SS

ORACLE

RDBMS Formats	Natural Date	Natural Time
DATE (ORACLE session parameter NLS_DATE_FORMAT is set to YYYYMMDDHH24MISS)	YYYYMMDD000000 (ORACLE time component is set to null for update commands and ignored for retrieval commands.)	YYYYMMDDHHIISS *

SYBASE

RDBMS Formats	Natural Date	Natural Time
DATETIME	YYYYMMDD	YYYYMMDD HH:II:SS *

* When comparing two time values, remember that the date components may have different values.

Microsoft SQL Server

RDBMS Formats	Natural Date	Natural Time
DATETIME	YYYYMMDD	YYYYMMDD HH:II:SS *

Obtaining Diagnostic Information about Database Errors

If the database returns an error while being accessed, you can call the non-Natural program CMOSQERR to obtain diagnostic information about the error, using the following syntax:

```
CALL 'CMOSQERR' parm1 parm2
```

The parameters are:

Parameter	Format/Length	Description
parm1	I4	The number of the error returned by the database.
parm2	A70	The text of the error returned by the database.

SQL Authorization

The Natural Configuration Utility allows you to add DBID specific settings of user IDs and passwords for automatic login to SQL databases. It distinguishes between operating system authentication and database authentication, depending on the current database system. If the **Auto login** flag in the SQL Authorization table is set for an SQL DBID then no interactive login prompt will pop up. The login values will be taken from this table row.

Please refer to *SQL Assignments* in the *Configuration Utility* documentation for a more detailed description of the SQL Authorization table.