Flexible SQL

This chapter covers the following topics:

- Using Flexible SQL
- Specifying Text Variables in Flexible SQL

Using Flexible SQL

In addition to the SQL syntax described in the previous sections, flexible SQL enables you to use arbitrary SQL syntax.

Characters << and >>

Flexible SQL is enclosed in << and >> characters. It can include arbitrary SQL text and host variables. Within flexible SQL, host variables *must* be prefixed by a colon (:).

The flexible SQL string can cover several statement lines. Comments are possible, too (see also the statement `PROCESS SQL`).

Flexible SQL can be used as a replacement for any of the following syntactical SQL items:

- *atom*
- *column-reference*
- *scalar-expression*
- *predicate*

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

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

Note:
The SQL text used in flexible SQL is not recognized by the Natural compiler. The SQL text (with replaced host variables) is simply copied into the SQL string passed to the database system. Syntax errors in flexible SQL are detected at runtime when the database executes the corresponding statement.

Example 1

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

Example 2:

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

Example 3:

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

Specifying Text Variables in Flexible SQL

Within flexible SQL, you can also specify so-called "text variables".

<<:**T**:*host-variable* [**LINDICATOR**:*host-variable*]>>

The syntax items are described below:

`:T:`	A text variable is a *`host-variable`* prefixed by `:T:`. It must be in alphanumeric format. At runtime, a text variable within an SQL statement will be replaced by its contents that is, the text string contained in the text variable will be inserted into the SQL string. After the replacement, trailing blanks will be removed from the inserted text string. You have to make sure yourself that the content of a text variable results in a syntactically correct SQL string. In particular, the content of a text variable must not contain *`host-variables`*. A statement containing a text variable will always be executed in dynamic SQL mode.
`LINDICATOR`	**LINDICATOR Option:** The text variable can be followed by the keyword `LINDICATOR` and a length indicator variable (that is, a *`host-variable`* prefixed by colon). The length indicator variable has to be of format/length I2. If no `LINDICATOR` variable is specified, the entire content of the text variable will be inserted into the SQL string. If you specify a `LINDICATOR` variable, only the first *n* characters (*n* being the value of the `LINDICATOR` variable) of the text variable content will be inserted into the SQL string. If the number in the `LINDICATOR` variable is greater than the length of the text variable content, the entire text variable content will be inserted. If the number in the `LINDICATOR` variable is negative or 0, nothing will be inserted. See general information on *`host-variable`*.

Example Using Text Variable

```
DEFINE DATA LOCAL
01 TEXTVAR (A200)
01 TABLES VIEW OF SYSIBM-SYSTABLES
   02 NAME
   02 CREATOR
END-DEFINE
*
MOVE 'WHERE NAME > ''SYS'' AND CREATOR = ''SYSIBM''' TO TEXTVAR
*
SELECT * INTO VIEW TABLES
  FROM SYSIBM-SYSTABLES
  << :T:TEXTVAR >>
  DISPLAY TABLES
END-SELECT
*
END
```

The generated SQL statement (as displayed with the `LISTSQL` system command) will look as follows:

```
SELECT NAME, CREATOR FROM SYSIBM.SYSTABLES:T: FOR FETCH ONLY
```

The executed SQL statement will look as follows:

```
SELECT TABNAME, CREATOR FROM SYSIBM.SYSTABLES
  WHERE TABNAME > 'SYS' AND CREATOR = 'SYSIBM'
```