

FIND

FIND	$\left[\left[\begin{array}{l} \text{ALL} \\ \text{(operand1)} \\ \text{FIRST} \\ \text{NUMBER} \\ \text{UNIQUE} \end{array} \right] \right]$	$[MULTI-FETCH-clause] [] \left[\left\{ \begin{array}{l} \text{RECORD} \\ \text{RECORDS} \end{array} \right\} \right]$	$[\text{IN}] [\text{FILE}] \text{view-name}$
	$[\text{PASSWORD}=\text{operand2}]$		
	$[\text{CIPHER}=\text{operand3}]$		
	$[\text{WITH}] [[\text{LIMIT}] (\text{operand4})] \text{basic-search-criterion}$		
	$[\text{COUPLED-clause}] \dots 4/42$		
	$[\text{STARTING WITH ISN}=\text{operand5}]$		
	$[\text{SORTED-BY-clause}]$		
	$[\text{RETAIN-clause}]$		
	$[\text{WHERE-clause}]$		
	$[\text{IF-NO-RECORDS-FOUND-clause}]$		
	$\text{statement} \dots$		
END-FIND		$(\text{structured mode only})$	
[LOOP]		$(\text{reporting mode only})$	

This chapter covers the following topics:

- Function
- Restrictions
- Syntax Description
- Examples

For an explanation of the symbols used in the syntax diagram, see *Syntax Symbols*.

Related Statements: ACCEPT/REJECT | AT BREAK | AT START OF DATA | AT END OF DATA | BACKOUT TRANSACTION | BEFORE BREAK PROCESSING | DELETE | END TRANSACTION | FIND | GET | GET SAME | GET TRANSACTION | HISTOGRAM | LIMIT | PASSW | PERFORM BREAK PROCESSING | READ | RETRY | STORE | UPDATE

Belongs to Function Group: *Database Access and Update*

Function

The FIND statement is used to select a set of records from the database based on a search criterion consisting of fields defined as descriptors (keys).

This statement causes a processing loop to be initiated and then executed for each record selected. Each field in each record may be referenced within the processing loop. It is not necessary to issue a READ statement following the FIND in order to reference the fields within each record selected.

See also *FIND Statement* (in the *Programming Guide*).

Database-Specific Considerations

DL/I	When accessing a field starting after the last byte of the given segment occurrence, the storage copy of this field is filled according to its format (numeric, blank, etc.). The term segment occurrences should be substituted for the term records as used in this description of the FIND statement.
VSAM	The FIND statement is only valid for key-sequenced (KSDS) and entry-sequenced (ESDS) VSAM datasets. For ESDS, an alternate index for the base cluster must be defined.
SQL	<p>FIND FIRST as well as the PASSWORD, CIPHER, COUPLED and RETAIN clauses are not permitted.</p> <p>FIND UNIQUE is not permitted. (Exception: FIND UNIQUE can be used for primary keys; however, this is only permitted for compatibility reasons and should not be used.)</p> <p>The SORTED BY clause corresponds with the SQL clause ORDER BY.</p> <p>The basic search criterion for an SQL-database table may be specified in the same manner as for an Adabas file. The term record used in this context corresponds with the SQL term "row".</p>

System Variables with the FIND Statement

The Natural system variables *ISN, *NUMBER, and *COUNTER are automatically created for each FIND statement issued. A reference number must be supplied if the system variable was referenced outside the current processing loop or through a FIND UNIQUE, FIND FIRST, or FIND NUMBER statement. The format/length of each of these system variables is P10; this format/length cannot be changed.

*ISN	Adabas VSAM DL/I and SQL Entire System Server	* ISN contains the Adabas internal sequence number (ISN) of the record currently being processed. * ISN is not available for the FIND NUMBER statement. See <i>*ISN for VSAM</i> in the <i>System Variables</i> documentation. * ISN is not available. * ISN is not available.
*NUMBER	Adabas VSAM DL/I Entire System Server	*NUMBER contains the number of records which satisfied the basic search criterion specified in the WITH clause. See <i>*NUMBER for VSAM</i> in the <i>System Variables</i> documentation. See <i>*NUMBER for DL/I</i> in the <i>System Variables</i> documentation. *NUMBER is not available.
*COUNTER	The system variable *COUNTER contains the number of times the processing loop has been entered.	

See also *Example 13 - Using System Variables with the FIND Statement*.

Issuing Multiple FIND Statements

Multiple FIND statements may be issued to create nested loops whereby an inner loop is entered for each record selected in the outer loop.

See also *Example 14 - Multiple FIND Statements*.

Restrictions

With Entire System Server, FIND NUMBER and FIND UNIQUE as well as the PASSWORD, CIPHER, COUPLED and RETAIN clauses are not permitted.

Syntax Description

Operand Definition Table:

Operand	Possible Structure				Possible Formats								Referencing Permitted	Dynamic Definition	
<i>operand1</i>	C	S				N	P	I	B*					yes	no
<i>operand2</i>	C	S			A									yes	no
<i>operand3</i>	C	S				N								yes	no
<i>operand4</i>	C	S				N	P	I	B*					yes	no
<i>operand5</i>	C	S				N	P	I	B*					yes	no

* Format B of *operand1*, *operand4* and *operand5* may be used only with a length of less than or equal to 4.

Syntax Element Description:

<p>ALL/<i>operand1</i></p>	<p>Processing Limit:</p> <p>The number of records to be processed from the selected set may be limited by specifying <i>operand1</i> either as a numeric constant (in the range from 0 to 4294967295) or as the name of a numeric variable enclosed in parentheses.</p> <p>ALL may be optionally specified and emphasizes that all selected records are to be processed.</p> <p>If you specify a limit with <i>operand1</i>, this limit applies to the FIND loop being initiated. Records rejected for processing by the WHERE clause are not counted against this limit.</p> <pre>FIND (5) IN EMPLOYEES WITH ...</pre> <pre>MOVE 10 TO #CNT(N2)</pre> <pre>FIND (#CNT) EMPLOYEES WITH ...</pre> <p>For this statement, the specified limit has priority over a limit set with a LIMIT statement.</p> <p>If a smaller limit is set with the LT parameter, the LT limit applies.</p> <p>Notes:</p> <ol style="list-style-type: none"> 1. If you wish to process a 4-digit number of records, specify it with a leading zero: (0nnnn); because Natural interprets every 4-digit number enclosed in parentheses as a line-number reference to a statement. 2. <i>operand1</i> has no influence on the size of an ISN set that is to be retained by a RETAIN clause. <i>operand1</i> is evaluated when the FIND loop is entered. If the value of <i>operand1</i> is modified within the FIND loop, this does not affect the number of records processed.
-----------------------------------	--

FIND FIRST FIND NUMBER FIND UNIQUE	<p>These options are used</p> <ul style="list-style-type: none"> ● to select the first record of a selected set (see <i>FIND FIRST</i>), ● to determine the number of records in a selected set (see <i>FIND NUMBER</i>), or ● to ensure that only one record satisfies a selection criterion (see <i>FIND UNIQUE</i>). <p>For a detailed description of these options, see below.</p>
<i>MULTI-FETCH-clause</i>	<p>For Adabas databases, Natural offers a MULTI-FETCH clause, that allows one to read more than one record per database access. For further information, see <i>MULTI-FETCH Clause</i>.</p>
<i>view-name</i>	<p>The name of a view as defined either within a DEFINE DATA block or in a separate global or local data.</p> <p>In reporting mode, <i>view-name</i> is the name of a DDM if no DEFINE DATA LOCAL statement is used.</p>
<i>PASSWORD=operand2</i>	<p>PASSWORD Clause:</p> <p>The PASSWORD clause applies only for Adabas or VSAM databases. This clause is not permitted with Entire System Server.</p> <p>The PASSWORD clause is used to provide a password (<i>operand2</i>) when reading/writing data from an Adabas or VSAM file which is password protected. If you require access to a password-protected file, contact the person responsible for database security concerning password usage/assignment.</p> <p>If the password is specified as a constant, the PASSWORD clause should always be coded at the very beginning of a source-code line; and there should be no blank between the keyword PASSWORD and the equal sign; this ensures that the password is not visible/displayable in the source code of the program.</p> <p>In TP mode, you may enter the PASSWORD clause invisible by entering the terminal command <code>%*</code> before you type in the PASSWORD clause.</p> <p>If the PASSWORD clause is omitted, the default password specified with the PASSW statement applies.</p> <p>The password value must not be changed during the execution of a processing loop.</p> <p>See also <i>Example 1 - PASSWORD Clause</i>.</p>

CIPHER=operand3	<p>CIPHER Clause :</p> <p>The CIPHER clause only applies to Adabas databases. This clause is not permitted with Entire System Server.</p> <p>The CIPHER clause is used to provide a cipher key (<i>operand3</i>) when retrieving data from Adabas files which are enciphered. If you require access to an enciphered file, contact the person responsible for database security concerning cipher key usage/assignment.</p> <p>The cipher key may be specified as a numeric constant with 8 digits or as a user-defined variable with format/length N8.</p> <p>If the cipher key is specified as a constant, the CIPHER clause should always be coded at the very beginning of a source-code line; this ensures that the cipher key is not visible/displayable in the source code of the program. In TP mode, you may enter the CIPHER clause invisible by entering the Natural terminal command %* before you type in the CIPHER clause.</p> <p>The value of the cipher key must not be changed during the processing of a loop initiated by a FIND statement.</p> <p>See also <i>Example 2 - CIPHER Clause</i>.</p>
------------------------	--

<p>WITH LIMIT <i>operand4</i> <i>basic-search-criterion</i></p>	<p>WITH Clause:</p> <p>The WITH clause is required. It is used to specify the basic-search-criterion (see <i>Search Criterion for Adabas Files</i>) consisting of key fields (descriptors) defined in the database.</p> <p>The following database-specific considerations apply:</p> <p>For Adabas files: You may use Adabas descriptors, subdescriptors, superdescriptors, hyperdescriptors, and phonetic descriptors within a WITH clause. A non-descriptor (that is, a field marked in the DDM with N) can also be specified.</p> <p>For DL/I files: You may only use key fields marked with D in the DDM.</p> <p>For VSAM files: You may use VSAM key fields only.</p> <p>The number of records to be selected as a result of a WITH clause may be limited by specifying the keyword LIMIT together with a numeric constant or a user-defined variable, enclosed within parentheses, which contains the limit value (<i>operand4</i>). If the number of records selected exceeds the limit, the program will be terminated with an error message.</p> <p>Note: If the limit is to be a 4-digit number, specify it with a leading zero: (0nnnn); because Natural interprets every 4-digit number enclosed in parentheses as a line-number reference to a statement.</p>
<p>COUPLED-clause</p>	<p>This clause may be used used to specify a search which involves the use of the Adabas coupling facility. See <i>COUPLED Clause</i>.</p>
<p>STARTING WITH ISN=<i>operand5</i></p>	<p>This clause may be used for repositioning within a FIND loop whose processing has been interrupted. See <i>STARTING WITH Clause</i>.</p>
<p>SORTED-BY-clause</p>	<p>This clause may be used to cause Adabas to sort the selected records based on the sequence of one to three descriptors. See <i>SORTED BY Clause</i>.</p>
<p>RETAIN-clause</p>	<p>This clause may be used to retain the result of an extensive search in large files for further processing. See <i>RETAIN Clause</i>.</p>
<p>WHERE-clause</p>	<p>This clause may be used to specify an additional selection criterion (<i>logical-condition</i>). See <i>WHERE Clause</i>.</p>

<i>IF-NO-RECORDS-FOUND-clause</i>	This clause may be used to cause a processing loop initiated with a FIND statement to be entered in the event that no records meet the selection criteria specified in the WITH clause and the WHERE clause. See <i>IF NO RECORDS FOUND Clause</i> .
END-FIND	The Natural reserved keyword END-FIND must be used to end the FIND statement.

FIND FIRST

The FIND FIRST statement may be used to select and process the first record which meets the WITH and WHERE criteria.

For Adabas databases, the record processed will be the record with the lowest Adabas ISN from the set of qualifying records.

This statement does *not* initiate a processing loop.

Restrictions with FIND FIRST

- FIND FIRST can only be used in reporting mode.
- FIND FIRST is not available for DL/I and SQL databases.
- The IF NO RECORDS FOUND clause must not be used in a FIND FIRST statement.

System Variables with FIND FIRST

The following Natural system variables are available with the FIND FIRST statement:

*ISN	The system variable *ISN contains the Adabas ISN of the selected record. *ISN will be zero if no record is found after the evaluation of the WITH and WHERE criteria. *ISN is not available for VSAM databases or with Entire System Server.
*NUMBER	The system variable *NUMBER contains the number of records found after the evaluation of the WITH criterion and before evaluation of any WHERE criterion. *NUMBER will be zero if no record meets the WITH criterion. *NUMBER is not available with Entire System Server.
*COUNTER	The system variable *COUNTER contains 1 if a record was found; contains 0 if no record was found.

Example of FIND FIRST Statement: See the Program FNDFIR (reporting mode)

FIND NUMBER

The FIND NUMBER statement is used to determine the number of records which satisfy the WITH/WHERE criteria specified. It does *not* result in the initiation of a processing loop and *no data fields from the database are made available*.

Note:

Use of the WHERE clause may result in significant overhead.

Restrictions with FIND NUMBER

- The SORTED BY clause and the IF NO RECORDS FOUND clause must not be used with the FIND NUMBER statement.
- The WHERE clause cannot be used in structured mode.
- FIND NUMBER is not available for DL/I databases or with Entire System Server.

System Variables with FIND NUMBER

The following Natural system variables are available with the FIND NUMBER statement:

*NUMBER	The system variable *NUMBER contains the number of records found after the evaluation of the WITH criterion.
*COUNTER	The system variable *COUNTER contains the number of records found after the evaluation of the WHERE criterion. *COUNTER is only available if the FIND NUMBER statement contains a WHERE clause.

FIND UNIQUE

The FIND UNIQUE statement may be used to ensure that only one record is selected for processing. It does *not* result in the initiation of a processing loop. If a WHERE clause is specified, an automatic internal processing loop is created to evaluate the WHERE clause.

If no records or more than one record satisfy the criteria, an error message will be issued. This condition can be tested with the ON ERROR statement.

Restrictions with FIND UNIQUE

- FIND UNIQUE can only be used in reporting mode.
- FIND UNIQUE is not available for DL/I databases or with Entire System Server.
- For SQL databases, FIND UNIQUE cannot be used. (Exception: On mainframe computers, FIND UNIQUE can be used for primary keys; however, this is only permitted for compatibility reasons and should not be used.)
- The SORTED BY and IF NO RECORDS FOUND clauses must not be used with the FIND UNIQUE statement.

System Variables with FIND UNIQUE

*ISN	The system variable *ISN contains the unique ISN number of the record, which itself must be unique.
*NUMBER	The system variable *NUMBER always contains 1 for a valid FIND UNIQUE execution. *NUMBER may contain any other positive value ($=0$ or ≥ 2) if an error has occurred. This error condition may be used by the ON ERROR statement. *NUMBER is not allowed if the WHERE clause is missing.
*COUNTER	The system variable *COUNTER contains the number of records found after the evaluation of the WHERE criterion. *COUNTER is not allowed if the WHERE clause is missing.

Example for FIND UNIQUE: See the Program FNDUNQ (reporting mode).

MULTI-FETCH Clause

Note:

This clause can only be used for Adabas or DB2 databases.

MULTI-FETCH $\left\{ \begin{array}{l} \text{ON} \\ \text{OFF} \\ [\text{OF}] \text{ multi-fetch-factor} \end{array} \right\}$
--

For more information, see the section *Multi-Fetch Clause* (Adabas) in the *Programming Guide* or *Multiple Row Processing (SQL)* in the *Natural for DB2* part of the *Database Management System Interfaces* documentation.

Search Criterion for Adabas Files

1 <i>descriptor</i> [(i)]	{ EQ = EQUAL EQUAL TO }	<i>value</i>	{ [OR { EQ = EQUAL EQUAL TO } <i>value</i>] ... } THRU <i>value</i> [BUT NOT <i>value</i> [THRU <i>value</i>]]
2 <i>descriptor</i> [(i)]	{ EQ = EQUAL EQUAL TO NE ¬ = <> NOT = NOT EQ NOTEQUAL NOT EQUAL NOT EQUAL TO LT LESS THAN < GE GREATER EQUAL >= NOT < NOT LT GREATER THAN > LE LESS EQUAL <= NOT > NOT GT }	<i>value</i>	
3 <i>set-name</i>			

Operand Definition Table:

Operand	Possible Structure			Possible Formats												Referencing Permitted	Dynamic Definition	
<i>descriptor</i>	S	A		A	N	P	I	F	B	D	T	L					no	no
<i>value</i>	C	S		A	N	P	I	F	B	D	T	L					yes	no
<i>set-name</i>	C	S		A													no	no

Syntax Element Description:

<i>descriptor</i>	Adabas descriptor, subdescriptor, superdescriptor, hyperdescriptor, or phonetic descriptor. A field marked as non-descriptor in the DDM can also be specified.
(i)	A descriptor contained within a periodic group may be specified with or without an index. If no index is specified, the record will be selected if the value specified is located in any occurrence. If an index is specified, the record is selected only if the value is located in the occurrence specified by the index. The index specified must be a constant. An index range must not be used. No index must be specified for a descriptor which is a multiple-value field. The record will be selected if the value is located in the record regardless of the position of the value.
<i>value</i>	Search value. The formats of the descriptor and the search value must be compatible.
<i>set-name</i>	Identifies a set of records previously selected with a FIND statement in which the RETAIN clause was specified. The set referenced in a FIND must have been created from the same physical Adabas file. <i>set-name</i> may be specified as a text constant (maximum 32 characters) or as the content of an alphanumeric variable. <i>set-name</i> cannot be used with Entire System Server.

See also:

- *Example 3 - Basic Search Criterion in WITH Clause*
- *Example 4 - Basic Search Criterion with Multiple-Value Field*

Search Criterion with Null Indicator

<i>null-indicator</i>	$\left\{ \begin{array}{l} = \\ \mathbf{EQ} \\ \mathbf{EQUAL} \\ [\mathbf{TO}] \end{array} \right\}$	<i>value</i>
-----------------------	---	--------------

Operand Definition Table:

Operand	Possible Structure		Possible Formats										Referencing Permitted	Dynamic Definition	
<i>null-indicator</i>		S											I	no	no
<i>value</i>	C	S							N	P	I	F	B	yes	no

Syntax Element Description:

<i>null-indicator</i>	The null indicator.
<i>value</i>	Possible Value:
	-1 The corresponding field contains no value.
	0 The corresponding field does contain a value.

Connecting Search Criteria (for Adabas Files)

Basic-search-criteria can be combined using the Boolean operators AND, OR, and NOT. Parentheses may also be used to control the order of evaluation. The order of evaluation is as follows:

1. (): Parentheses
2. NOT: Negation (only for a *basic-search-criterion* of form [2]).
3. AND: AND connection
4. OR: OR connection

Basic-search-criteria may be connected by logical operators to form a complex *search-expression*. The syntax for such a complex *search-expression* is as follows:

$[NOT] \left\{ \begin{array}{l} basic-search-criterion \\ (search-expression) \end{array} \right\} \left[\left[\left\{ \begin{array}{l} OR \\ AND \end{array} \right\} search-expression \right] \dots \right]$

See also *Example 5 - Various Samples of Complex Search Expression in WITH Clause.*

Descriptor-Key Usage

Adabas users may use database fields which are defined as descriptors to construct basic search criteria.

Subdescriptors, Superdescriptors, Hyperdescriptors and Phonetic Descriptors

With Adabas, subdescriptors, superdescriptors, hyperdescriptors and phonetic descriptors may be used to construct search criteria.

- A subdescriptor is a descriptor formed from a portion of a field.
- A superdescriptor is a descriptor whose value is formed from one or more fields or portions of fields.

- A hyperdescriptor is a descriptor which is formed using a user-defined algorithm.
- A phonetic descriptor is a descriptor which allows the user to perform a phonetic search on a field (for example, a person's name). A phonetic search results in the return of all values which sound similar to the search value.

Which fields may be used as descriptors, subdescriptors, superdescriptors, hyperdescriptors and phonetic descriptors with which file is defined in the corresponding DDM.

Values for Subdescriptors, Superdescriptors, Phonetic Descriptors

Values used with these types of descriptors must be compatible with the internal format of the descriptor. The internal format of a subdescriptor is the same as the format of the field from which the subdescriptor is derived. The internal format of a superdescriptor is binary if all of the fields from which it is derived are defined with numeric format; otherwise, the format is alphanumeric. Phonetic descriptors always have alphanumeric format.

Values for subdescriptors and superdescriptors may be specified in the following ways:

- Numeric or hexadecimal constants may be specified. A hexadecimal constant must be used for a value for a superdescriptor which has binary format (see above).
- Values in user-defined variable fields may be specified using the REDEFINE statement to select the portions that form the subdescriptor or superdescriptor value.

Using Descriptors Contained within a Database Array

A descriptor which is contained within a database array may also be used in the construction of basic search criterion. For Adabas databases, such a descriptor may be a multiple-value field or a field contained within a periodic group.

A descriptor contained within a periodic group may be specified with or without an index. If no index is specified, the record will be selected if the value specified is located in any occurrence. If an index is specified, the record is selected only if the value is located in the occurrence specified by the index. The index specified must be a constant. An index range must not be used.

No index must be specified for a descriptor which is a multiple-value field. The record will be selected if the value is located in the record regardless of the position of the value.

See also *Example 6 - Various Samples Using Database Arrays*.

Search Criterion for VSAM Files - basic-search-criterion

1 <i>descriptor</i>	EQ = EQUAL EQUAL TO NE ≠ <> NOT = NOT EQ NOTEQUAL NOT EQUAL NOT EQUAL TO LT LESS THAN < GE GREATER EQUAL >= NOT < NOT LT GT GREATER THAN > LE LESS EQUAL <= NOT > NOT GT	<i>value</i>
2 <i>descriptor</i>	EQ = EQUAL EQUAL TO	<i>value THRU value</i>

Operand Definition Table:

Operand	Possible Structure			Possible Formats										Referencing Permitted	Dynamic Definition	
<i>descriptor</i>	S	A		A	N	P	B								no	no
<i>value</i>	C	S		A	N	P	B								yes	no

Syntax Element Description:

<i>descriptor</i>	The descriptor must be defined in a VSAM file as a VSAM key field and is marked in the DDM with P for primary key or A for alternate key.
<i>value</i>	The search value.

The formats of the *descriptor* and the search *value* must be compatible.

Search Criterion for DL/I Files - basic-search-criterion

1 <i>descriptor</i>	EQ = EQUAL EQUAL TO NE ≠ <> NOT = NOT EQ NOTEQUAL NOT EQUAL NOT EQUAL TO LT LESS THAN < GE GREATER EQUAL >= NOT < NOT LT GT GREATER THAN > LE LESS EQUAL <= NOT > NOT GT	<i>value</i>
2 <i>descriptor</i>	EQ = EQUAL EQUAL TO	<i>value</i> THRU <i>value</i> [BUT NOT <i>value</i>]

Operand Definition Table:

Operand	Possible Structure		Possible Formats										Referencing Permitted	Dynamic Definition									
<i>descriptor</i>	S	A				A	N	P			B											no	no
<i>value</i>	C	S				A	N	P			B											yes	no

Syntax Element Description:

<i>descriptor</i>	The descriptor must be a field defined in DL/I and is marked in the DDM with D.
<i>value</i>	The search value.

For HDAM databases, only the following *basic-search-criterion* is possible:

<i>descriptor</i>	$\left\{ \begin{array}{l} \text{EQ} \\ = \\ \text{EQUAL} \\ [\text{TO}] \end{array} \right\}$	<i>value</i>
-------------------	---	--------------

Connecting Search Criteria - for DL/I Files

$[\text{NOT}] \left\{ \begin{array}{l} \textit{basic-search-criterion} \\ (\textit{search-expression}) \end{array} \right\} \left[\left[\left\{ \begin{array}{l} \text{OR} \\ \text{AND} \end{array} \right\} \textit{search-expression} \right] \dots \right]$

basic-search-criteria that refer to different segment types must not be connected with the OR logical operator.

Examples:

```
FIND COURSE WITH COURSEN > 1
FIND COURSE WITH COURSEN > 1 AND COURSEN < 100
FIND OFFERING WITH (COURSEN-COURSE > 1 OR TITLE-COURSE = 'Natural')
                    AND LOCATION = 'DARMSTADT'
```

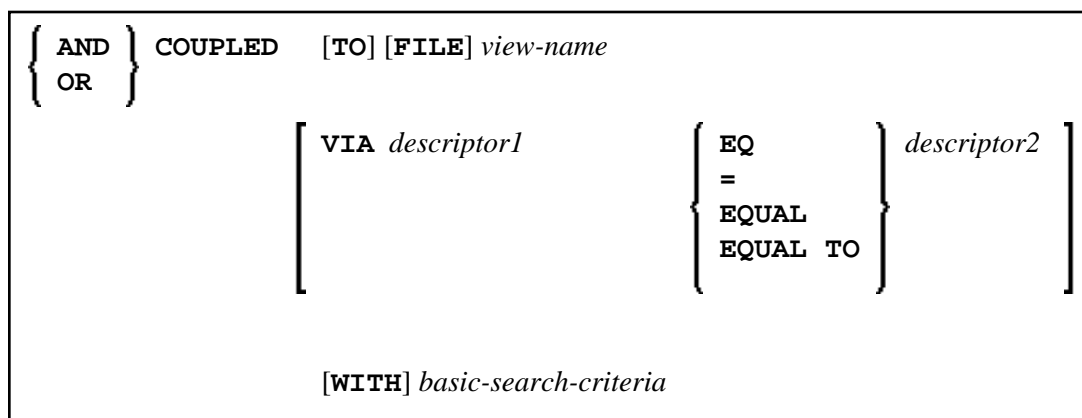
Invalid example:

```
FIND OFFERING WITH COURSEN-COURSE > 1 OR LOCATION = 'DARMSTADT'
```

COUPLED Clause

This clause only applies to Adabas databases.

This clause is not permitted with Entire System Server.



Operand Definition Table:

Operand	Possible Structure				Possible Formats										Referencing Permitted	Dynamic Definition		
<i>descriptor1</i>	S	A			A	N	P		B								no	no
<i>descriptor2</i>	S	A			A	N	P		B								no	no

Note:

Without the VIA clause, the COUPLED clause may be specified up to 4 times; with the VIA clause, it may be specified up to 42 times.

The COUPLED clause is used to specify a search which involves the use of the Adabas coupling facility. This facility permits database descriptors from different files to be specified in the search criterion of a single FIND statement.

The same Adabas file must not be used in two different FIND COUPLED clauses within the same FIND statement.

A *set-name* (see *RETAIN Clause*) must not be specified in the *basic-search-criteria* .

Database fields in a file specified within the COUPLED clause are not available for subsequent reference in the program unless another FIND or READ statement is issued separately against the coupled file.

Note:

If the COUPLED clause is used, the main WITH clause may be omitted. If the main WITH clause is omitted, the keywords AND/OR of the COUPLED clause must not be specified.

Physical Coupling without VIA Clause

The files used in a COUPLED clause without VIA must be physically coupled using the appropriate Adabas utility (as described in the Adabas documentation).

See also *Example 7 - Using Physically Coupled Files*.

The reference to NAME in the DISPLAY statement of the above example is valid since this field is contained in the EMPLOYEES file, whereas a reference to MAKE would be invalid since MAKE is contained in the VEHICLES file, which was specified in the COUPLED clause.

In this example, records will be found only if EMPLOYEES and VEHICLES have been physically coupled.

Logical Coupling - VIA Clause

The option `VIA descriptor1 = descriptor2` allows you to logically couple multiple Adabas files in a search query, where:

- *descriptor1* is a field from the first view.
- *descriptor2* is a field from the second view.

The two files need not be physically coupled in Adabas. This COUPLED option uses the soft-coupling feature of Adabas Version 5 and above, as described in the Adabas documentation.

See also *Example 8 - VIA Clause*.

STARTING WITH Clause

This clause applies only to Adabas and VSAM databases; for VSAM, it is only valid for ESDS.

You can use this clause to specify as *operand5* an Adabas ISN (internal sequence number) or VSAM RBA (relative byte address) respectively, which is to be used as a start value for the selection of records.

This clause may be used for repositioning within a FIND loop whose processing has been interrupted, to easily determine the next record with which processing is to continue. This is particularly useful if the next record cannot be identified uniquely by any of its descriptor values. It can also be useful in a distributed client/server application where the reading of the records is performed by a server program while further processing of the records is performed by a client program, and the records are not processed all in one go, but in batches.

Note:

The start value actually used will not be the value of *operand5*, but the next higher value.

Example:

See the program FNDSISN in the library SYSEXSYN.

SORTED BY Clause

This clause only applies to Adabas and SQL databases.

This clause is not permitted with Entire System Server.

<code>SORTED [BY] descriptor ... 3 [DESCENDING]</code>
--

The SORTED BY clause is used to cause Adabas to sort the selected records based on the sequence of one to three descriptors. The descriptors used for controlling the sort sequence may be different from those used for selection.

By default, the records are sorted in *ascending* sequence of values; if you want them to be in descending sequence, specify the keyword `DESCENDING`. The sort is performed using the Adabas inverted lists and does not result in any records being read.

Note:

The use of this clause may result in significant overhead if any descriptor used to control the sort sequence contains a large number of values. This is because the entire value list may have to be scanned until all selected records have been located in the list. When a large number of records is to be sorted, you should use the `SORT` statement.

Adabas sort limits (see the `ADARUN LS` parameter in the Adabas documentation) are in effect when the `SORTED BY` clause is used.

A descriptor which is contained in a periodic group must not be specified in the `SORTED BY` clause. A multiple-value field (without an index) may be specified.

Non-descriptors may also be specified in the `SORTED BY` clause. However, this function is not available on mainframes.

If the `SORTED BY` clause is used, the `RETAIN` clause must not be used.

See also *Example 9 - SORTED BY Clause*.

Considerations for Combined Use of STARTING WITH and SORTED BY Clauses

If both the `STARTING WITH` and the `SORTED BY` clause are used in the same `FIND` statement and the underlying database is Adabas, the following should be considered.

With Adabas for Mainframes

On Adabas for Mainframes, the `FIND` statement is executed in the following steps:

1. All records matching the search criterion are gathered and put in ISN sequence.
2. The records are sorted by the descriptor specified in the `SORTED BY` clause.
3. The record whose ISN value is specified in the `STARTING WITH` clause is positioned in the "sorted-by-descriptor" record list.
4. The records following the record found under Step 3 are returned in the `FIND` loop.

With Adabas for OpenSystems

On Adabas for OpenSystems (UNIX, OpenVMS, Windows) the same statement is executed as follows:

1. All records matching the search criterion are gathered and put in ISN sequence.
2. The record whose ISN value is specified in the `STARTING WITH` clause is positioned in the "sorted-by-ISN" record list.

- All records following the record found under Step 2 are sorted by the descriptor specified in the SORTED BY clause and returned in the FIND loop.

Example:

If the following program is executed with Adabas Version 8 for mainframes and Adabas Version 6.1 for UNIX/OpenVMS/Windows:

```

DEFINE DATA LOCAL
1 V1 VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 CITY
1 #ISN (I4)
END-DEFINE
FORMAT NL=5 SG=OFF PS=43 AL=15
*
PRINT 'FIND' (I)
FIND V1 WITH NAME = 'B' THRU 'BALBIN'
  RETAIN AS 'SET1'
  IF *COUNTER = 4 THEN
    #ISN := *ISN
  END-IF
  DISPLAY *ISN V1
END-FIND
*
PRINT / 'FIND .. SORTED BY NAME' (I)
FIND V1 WITH 'SET1'
  SORTED BY NAME
  DISPLAY *ISN V1
END-FIND
*
PRINT / 'FIND .. STARTING WITH ISN = ' (I) #ISN (AD=I)
FIND V1 WITH 'SET1'
  STARTING WITH ISN = #ISN
  DISPLAY *ISN V1
END-FIND
*
PRINT / 'FIND .. STARTING WITH ISN = ' (I) #ISN (AD=I)
      ' .. SORTED BY NAME' (I)
FIND V1 WITH 'SET1'
  STARTING WITH ISN = #ISN
  SORTED BY NAME
  DISPLAY *ISN V1
END-FIND
END

```

The result is as follows:

Results on Natural for Mainframes (Adabas Version 8)				Results on Natural for OpenSystems (Adabas Version 6.1)			
ISN	NAME	FIRST-NAME	CITY	ISN	NAME	FIRST-NAME	CITY
FIND V1 WITH NAME = 'B' THRU 'BALBIN'				FIND V1 WITH NAME = 'B' THRU 'BALBIN'			
12	BAILLET	PATRICK	LYS LEZ LANNOY	12	BAILLET	PATRICK	LYS LEZ LANNOY
58	BAGAZJA	MARJAN	MONTHERME	58	BAGAZJA	MARJAN	MONTHERME
351	BAECKER	JOHANNES	FRANKFURT	351	BAECKER	JOHANNES	FRANKFURT
355	BAECKER	KARL	SINDELFINGEN	355	BAECKER	KARL	SINDELFINGEN
370	BACHMANN	HANS	MUENCHEN	370	BACHMANN	HANS	MUENCHEN
490	BALBIN	ENRIQUE	BARCELONA	490	BALBIN	ENRIQUE	BARCELONA
650	BAKER	SYLVIA	OAK BROOK	650	BAKER	SYLVIA	OAK BROOK
913	BAKER	PAULINE	DERBY	913	BAKER	PAULINE	DERBY
FIND .. SORTED BY NAME				FIND .. SORTED BY NAME			
370	BACHMANN	HANS	MUENCHEN	370	BACHMANN	HANS	MUENCHEN
351	BAECKER	JOHANNES	FRANKFURT	351	BAECKER	JOHANNES	FRANKFURT
355	BAECKER	KARL	SINDELFINGEN	355	BAECKER	KARL	SINDELFINGEN
58	BAGAZJA	MARJAN	MONTHERME	58	BAGAZJA	MARJAN	MONTHERME
12	BAILLET	PATRICK	LYS LEZ LANNOY	12	BAILLET	PATRICK	LYS LEZ LANNOY
650	BAKER	SYLVIA	OAK BROOK	650	BAKER	SYLVIA	OAK BROOK
913	BAKER	PAULINE	DERBY	913	BAKER	PAULINE	DERBY
490	BALBIN	ENRIQUE	BARCELONA	490	BALBIN	ENRIQUE	BARCELONA
FIND .. STARTING WITH ISN = 355				FIND .. STARTING WITH ISN = 355			
370	BACHMANN	HANS	MUENCHEN	370	BACHMANN	HANS	MUENCHEN
490	BALBIN	ENRIQUE	BARCELONA	490	BALBIN	ENRIQUE	BARCELONA
650	BAKER	SYLVIA	OAK BROOK	650	BAKER	SYLVIA	OAK BROOK
913	BAKER	PAULINE	DERBY	913	BAKER	PAULINE	DERBY
FIND .. STARTING WITH ISN = 355 .. SORTED BY NAME				FIND .. STARTING WITH ISN = 355 .. SORTED BY NAME			
58	BAGAZJA	MARJAN	MONTHERME	370	BACHMANN	HANS	MUENCHEN
12	BAILLET	PATRICK	LYS LEZ LANNOY	650	BAKER	SYLVIA	OAK BROOK
650	BAKER	SYLVIA	OAK BROOK	913	BAKER	PAULINE	DERBY
913	BAKER	PAULINE	DERBY	490	BALBIN	ENRIQUE	BARCELONA
490	BALBIN	ENRIQUE	BARCELONA				

A FIND statement with at most one of these options (SORTED BY or STARTING WITH ISN) always returns the same records in the same sequence, regardless under which system the statement is executed. If, however, both clauses are used together, the result returned depends on which Adabas platform is used to serve the database statement.

Therefore, if a Natural program is intended to be used on multiple platforms, the combination of a SORTED BY and STARTING WITH ISN clause in the same FIND statement should be avoided.

RETAIN Clause

This clause only applies to Adabas databases.

This clause is not permitted with Entire System Server.

RETAIN AS *operand6*

Operand Definition Table:

Operand	Possible Structure			Possible Formats												Referencing Permitted	Dynamic Definition
<i>operand6</i>	C	S		A												yes	no

Syntax Element Description:

RETAIN AS	<p>By using the <code>RETAIN</code> clause, the result of an extensive search in large files can be retained for further processing.</p> <p>The selection is retained as an <code>ISN-set</code> in the Adabas work file. The set may be used in subsequent <code>FIND</code> statements as a basic search criterion for further refinement of the set or for further processing of the records.</p> <p>The set created is file-specific and may only be used in another <code>FIND</code> statement that processes the same file. The set may be referenced by any Natural program.</p>
<i>operand6</i>	<p>Set Name:</p> <p>The set name is used to identify the record set. It may be specified as an alphanumeric constant or as the content of an alphanumeric user-defined variable. Duplicate set names are not checked; consequently, if a duplicate set name is specified, the new set replaces the old set.</p>

See also *Example 10 - RETAIN Clause*.

Releasing Sets

There is no specific limit for the number of sets that can be retained or the number of ISNs in a set. It is recommended that the minimum number of ISN sets needed at one time be defined. Sets that are no longer needed should be released using the `RELEASE SETS` statement.

If they are not released with a `RELEASE` statement, retained sets exist until the end of the Natural session, or until a logon to another library, when they are released automatically. A set created by one program may be referenced by another program for processing or further refinement using additional search criteria.

Updates by Other Users

The records identified by the ISNs in a retained set are not locked against access and/or update by other users. Before you process records from the set, it is therefore useful to check whether the original search criteria which were used to create the set are still valid: This check is done with another `FIND` statement, using the set name in the `WITH` clause as basic search criterion and specifying in a `WHERE` clause the original search criterion (that is, the basic search criteria as specified in the `WITH` clause of the `FIND` statement which was used to create the set).

Restriction

If the `RETAIN` clause is used, the `SORTED BY` clause must not be used.

WHERE Clause

WHERE <i>logical-condition</i>

The WHERE clause may be used to specify an additional selection criterion (*logical-condition*) which is evaluated *after* a value has been read and *before* any processing is performed on the value (including the AT BREAK evaluation).

The syntax for a *logical-condition* is described in the section *Logical Condition Criteria* (in the *Programming Guide*).

If a processing limit is specified in a FIND statement containing a WHERE clause, records which are rejected as a result of the WHERE clause are *not* counted against the limit. These records are, however, counted against a global limit specified in the Natural session parameter LT, the GLOBALS command, or LIMIT statement.

See also *Example 11 - WHERE Clause*.

IF NO RECORDS FOUND Clause

Structured Mode Syntax

```

IF NO  [RECORDS] [FOUND]
      { ENTER
        statement ...
      }
END-NOREC

```

Reporting Mode Syntax

```

IF NO [RECORDS] [FOUND]
{ ENTER
  statement
  DO statement ... DOEND
}

```

The IF NO RECORDS FOUND clause may be used to cause a processing loop initiated with a FIND statement to be entered in the event that no records meet the selection criteria specified in the WITH clause and the WHERE clause.

If no records meet the specified WITH and WHERE criteria, the IF NO RECORDS FOUND clause causes the FIND processing loop to be executed once with an "empty" record. If this is not desired, specify the statement ESCAPE BOTTOM within the IF NO RECORDS FOUND clause.

If one or more statements are specified with the IF NO RECORDS FOUND clause, the statements will be executed immediately before the processing loop is entered. If no statements are to be executed before entering the loop, the keyword ENTER must be used.

See also *Example 12 - IF NO RECORDS FOUND Clause*.

Database Values

Unless other value assignments are made in the statements accompanying an IF NO RECORDS FOUND clause, Natural will reset to empty all database fields which reference the file specified in the current loop.

Evaluation of System Functions

Natural system functions are evaluated once for the empty record that is created for processing as a result of the IF NO RECORDS FOUND clause.

Restriction

This clause cannot be used with FIND FIRST, FIND NUMBER and FIND UNIQUE.

Examples

- Example 1 - PASSWORD Clause
- Example 2 - CIPHER Clause
- Example 3 - Basic Search Criterion in WITH Clause
- Example 4 - Basic Search Criterion with Multiple-Value Field
- Example 5 - Various Samples of Complex Search Expression in WITH Clause
- Example 6 - Various Samples of Using Database Arrays
- Example 7 - Using Physically Coupled Files
- Example 8 - VIA Clause
- Example 9 - SORTED BY Clause
- Example 10 - RETAIN Clause
- Example 11 - WHERE Clause
- Example 12 - IF NO RECORDS FOUND Clause
- Example 13 - Using System Variables with the FIND Statement
- Example 14 - Multiple FIND Statements

See also the example for FIND NUMBER: program FNDNUM.

Example 1 - PASSWORD Clause

```
** Example 'FNDPWD': FIND (with PASSWORD clause)
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 PERSONNEL-ID
*
```

```

1 #PASSWORD (A8)
END-DEFINE
*
INPUT 'ENTER PASSWORD FOR EMPLOYEE FILE:' #PASSWORD (AD=N)
LIMIT 2
*
FIND EMPLOY-VIEW PASSWORD = #PASSWORD
      WITH NAME = 'SMITH'
      DISPLAY NOTITLE NAME PERSONNEL-ID
END-FIND
*
END

```

Output of Program FNDPWD:

ENTER PASSWORD FOR EMPLOYEE FILE:

Example 2 - CIPHER Clause

```

** Example 'FNDCIP': FIND (with PASSWORD/CIPHER clause)
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 PERSONNEL-ID
*
1 #PASSWORD (A8)
1 #CIPHER (N8)
END-DEFINE
*
LIMIT 2
INPUT 'ENTER PASSWORD FOR EMPLOYEE FILE: ' #PASSWORD (AD=N)
      / 'ENTER CIPHER KEY FOR EMPLOYEE FILE: ' #CIPHER (AD=N)
*
FIND EMPLOY-VIEW PASSWORD = #PASSWORD
      CIPHER = #CIPHER
      WITH NAME = 'SMITH'
      DISPLAY NOTITLE NAME PERSONNEL-ID
END-FIND
*
END Output of Program FNDCIP:

ENTER PASSWORD FOR EMPLOYEE FILE:
ENTER CIPHER KEY FOR EMPLOYEE FILE:

```

Example 3 - Basic Search Criterion in WITH Clause

```

FIND STAFF WITH NAME = 'SMITH'
FIND STAFF WITH CITY NE 'BOSTON'
FIND STAFF WITH BIRTH = 610803
FIND STAFF WITH BIRTH = 610803 THRU 610811
FIND STAFF WITH NAME = 'O HARA' OR = 'JONES' OR = 'JACKSON'
FIND STAFF WITH PERSONNEL-ID = 100082 THRU 100100
      BUT NOT 100087 THRU 100095

```

Example 4 - Basic Search Criterion with Multiple-Value Field

When the descriptor used in the basic search criterion is a multiple-value field, basically four different kinds of results can be obtained (the field MU-FIELD in the following examples is assumed to be a multiple-value field):

```
FIND XYZ-VIEW WITH MU-FIELD = 'A'
```

This statement returns records in which *at least one* occurrence of MU-FIELD has the value A.

```
FIND XYZ-VIEW WITH MU-FIELD NOT EQUAL 'A'
```

This statement returns records in which *at least one* occurrence of MU-FIELD does *not* have the value A.

```
FIND XYZ-VIEW WITH NOT MU-FIELD NOT EQUAL 'A'
```

This statement returns records in which *every* occurrence of MU-FIELD has the value A.

```
FIND XYZ-VIEW WITH NOT MU-FIELD = 'A'
```

This statement returns records in which *none* of the occurrences of MU-FIELD has the value A.

Example 5 - Various Samples of Complex Search Expression in WITH Clause

```
FIND STAFF WITH BIRTH LT 19770101 AND DEPT = 'DEPT06'
```

```
FIND STAFF WITH JOB-TITLE = 'CLERK TYPIST'
                AND (BIRTH GT 19560101 OR LANG = 'SPANISH')
```

```
FIND STAFF WITH JOB-TITLE = 'CLERK TYPIST'
                AND NOT (BIRTH GT 19560101 OR LANG = 'SPANISH')
```

```
FIND STAFF WITH DEPT = 'ABC' THRU 'DEF'
                AND CITY = 'WASHINGTON' OR = 'LOS ANGELES'
                AND BIRTH GT 19360101
```

```
FIND CARS WITH MAKE = 'VOLKSWAGEN'
                AND COLOR = 'RED' OR = 'BLUE' OR = 'BLACK'
```

Example 6 - Various Samples of Using Database Arrays

The following examples assume that the field SALARY is a descriptor contained within a periodic group, and the field LANG is a multiple-value field.

```
FIND EMPLOYEES WITH SALARY LT 20000
```

Results in a search of all occurrences of SALARY.

```
FIND EMPLOYEES WITH SALARY (1) LT 20000
```

Results in a search of the first occurrence only.

```
FIND EMPLOYEES WITH SALARY (1:4) LT 20000 /* invalid
```

A range specification must not be specified for a field within a periodic group used as a search criterion.

```
FIND EMPLOYEES WITH LANG = 'FRENCH'
```

Results in a search of all values of LANG.

```
FIND EMPLOYEES WITH LANG (1) = 'FRENCH'          /* invalid
```

An index must not be specified for a multiple-value field used as a search criterion.

Example 7 - Using Physically Coupled Files

```
** Example 'FNDCPL': FIND (using coupled files)
** NOTE: Adabas files must be physically coupled when using the
**      COUPLED clause without the VIA clause.
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
1 VEHIC-VIEW VIEW OF VEHICLES
  2 MAKE
END-DEFINE
*
FIND EMPLOY-VIEW WITH CITY = 'FRANKFURT'
  AND COUPLED TO
    VEHIC-VIEW WITH MAKE = 'VW'
  DISPLAY NOTITLE NAME
END-FIND
*
END
```

Example 8 - VIA Clause

```
** Example 'FNDVIA': FIND (with VIA clause)
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
1 VEHIC-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
END-DEFINE
*
FIND EMPLOY-VIEW WITH NAME = 'ADKINSON'
  AND COUPLED TO VEHIC-VIEW
    VIA PERSONNEL-ID = PERSONNEL-ID WITH MAKE = 'VOLVO'
  DISPLAY PERSONNEL-ID NAME FIRST-NAME
END-FIND
*
END
```

Output of Program FNDVIA:

```
Page          1                                05-01-17  13:18:22

PERSONNEL      NAME          FIRST-NAME
  ID
-----
20011000  ADKINSON              BOB
```

Example 9 - SORTED BY Clause

```

** Example 'FNDSOR': FIND (with SORTED BY clause)
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 NAME
  2 FIRST-NAME
  2 PERSONNEL-ID
END-DEFINE
*
LIMIT 10
FIND EMPLOY-VIEW WITH CITY = 'FRANKFURT'
      SORTED BY NAME PERSONNEL-ID

      DISPLAY NOTITLE NAME (IS=ON) FIRST-NAME PERSONNEL-ID
END-FIND
*
END
    
```

Output of Program FNDSOR:

NAME	FIRST-NAME	PERSONNEL ID
BAECKER	JOHANNES	11500345
BECKER	HERMANN	11100311
BERGMANN	HANS	11100301
BLAU	SARAH	11100305
BLOEMER	JOHANNES	11200312
DIEDRICHS	HUBERT	11600301
DOLLINGER	MARGA	11500322
FALTER	CLAUDIA	11300311
	HEIDE	11400311
FREI	REINHILD	11500301

Example 10 - RETAIN Clause

```

** Example 'RELEX1': FIND (with RETAIN clause and RELEASE statement)
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 BIRTH
  2 NAME
*
1 #BIRTH (D)
END-DEFINE
*
MOVE EDITED '19400101' TO #BIRTH (EM=YYYYMMDD)
*
FIND NUMBER EMPLOY-VIEW WITH BIRTH GT #BIRTH
      RETAIN AS 'AGESET1'
IF *NUMBER = 0
  STOP
END-IF
*
FIND EMPLOY-VIEW WITH 'AGESET1' AND CITY = 'NEW YORK'
      DISPLAY NOTITLE NAME CITY BIRTH (EM=YYYY-MM-DD)
    
```

```
END-FIND
*
RELEASE SET 'AGESET1'
*
END
```

Output of Example 10:

NAME	CITY	DATE OF BIRTH
RUBIN	NEW YORK	1945-10-27
WALLACE	NEW YORK	1945-08-04

Example 11 - WHERE Clause

```
** Example 'FNDWHE': FIND (with WHERE clause)
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 JOB-TITLE
  2 CITY
END-DEFINE
*
FIND EMPLOY-VIEW WITH CITY = 'PARIS'
      WHERE JOB-TITLE = 'INGENIEUR COMMERCIAL'
  DISPLAY NOTITLE
      CITY JOB-TITLE PERSONNEL-ID NAME
END-FIND
*
END
```

Output of Program FNDWHE:

CITY	CURRENT POSITION	PERSONNEL ID	NAME
PARIS	INGENIEUR COMMERCIAL	50007300	CAHN
PARIS	INGENIEUR COMMERCIAL	50006500	MAZUY
PARIS	INGENIEUR COMMERCIAL	50004700	FAURIE
PARIS	INGENIEUR COMMERCIAL	50004400	VALLY
PARIS	INGENIEUR COMMERCIAL	50002800	BRETON
PARIS	INGENIEUR COMMERCIAL	50001000	GIGLEUX
PARIS	INGENIEUR COMMERCIAL	50000400	KORAB-BRZOZOWSKI

Example 12 - IF NO RECORDS FOUND Clause

```
** Example 'FNDIFN': FIND (using IF NO RECORDS FOUND)
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
1 VEHIC-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
```

```

2 MAKE
END-DEFINE
*
LIMIT 15
EMP. READ EMPLOY-VIEW BY NAME STARTING FROM 'JONES'
/*
VEH. FIND VEHIC-VIEW WITH PERSONNEL-ID = PERSONNEL-ID (EMP.)

    IF NO RECORDS FOUND
        MOVE '*** NO CAR ***' TO MAKE
    END-NOREC
    /*
    DISPLAY NOTITLE
        NAME (EMP.) (IS=ON)
        FIRST-NAME (EMP.) (IS=ON)
        MAKE (VEH.)
    END-FIND
    /*
END-READ
END

```

Output of Program FNDIFN:

NAME	FIRST-NAME	MAKE
JONES	VIRGINIA	CHRYSLER
	MARSHA	CHRYSLER
		CHRYSLER
	ROBERT	GENERAL MOTORS
	LILLY	FORD
		MG
	EDWARD	GENERAL MOTORS
	MARTHA	GENERAL MOTORS
	LAUREL	GENERAL MOTORS
	KEVIN	DATSUN
	GREGORY	FORD
JOPER	MANFRED	*** NO CAR ***
JOUSSELIN	DANIEL	RENAULT
JUBE	GABRIEL	*** NO CAR ***
JUNG	ERNST	*** NO CAR ***
JUNKIN	JEREMY	*** NO CAR ***
KAISER	REINER	*** NO CAR ***

Example 13 - Using System Variables with the FIND Statement

```

** Example 'FNDVAR': FIND (using *ISN, *NUMBER, *COUNTER)
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 CITY
END-DEFINE
*
LIMIT 3
FIND EMPLOY-VIEW WITH CITY = 'MADRID'
  DISPLAY NOTITLE PERSONNEL-ID NAME
  *ISN *NUMBER *COUNTER
END-FIND
*
END

```

Output of Program FNDVAR

PERSONNEL ID	NAME	ISN	NMBR	CNT
60000114	DE JUAN	400	41	1
60000136	DE LA MADRID	401	41	2
60000209	PINERO	405	41	3

Example 14 - Multiple FIND Statements

In the following example, first all people named SMITH are selected from the EMPLOYEES file. Then the PERSONNEL-ID from the EMPLOYEES file is used as the search key for an access to the VEHICLES file.

```

** Example 'FNDMUL': FIND (with multiple files)
*****
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
1 VEHIC-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
  2 MAKE
END-DEFINE
*
LIMIT 15
EMP. FIND EMPLOY-VIEW WITH NAME = 'SMITH'
/*
VEH. FIND VEHIC-VIEW WITH PERSONNEL-ID = EMP.PERSONNEL-ID
  IF NO RECORDS FOUND
    MOVE '*** NO CAR ***' TO MAKE
  END-NOREC
  DISPLAY NOTITLE
    EMP.NAME (IS=ON)
    EMP.FIRST-NAME (IS=ON)
    VEH.MAKE
  END-FIND
END-FIND
END

```

Output of Program FNDMUL:

The resulting report shows the NAME and FIRST-NAME (obtained from the EMPLOYEES file) of all people named SMITH as well as the MAKE of each car (obtained from the VEHICLES file) owned by these people.

NAME	FIRST-NAME	MAKE
SMITH	GERHARD	ROVER
	SEYMOUR	*** NO CAR ***
	MATILDA	FORD
	ANN	*** NO CAR ***
	TONI	TOYOTA
	MARTIN	*** NO CAR ***
	THOMAS	FORD
	SUNNY	*** NO CAR ***

MARK
LOUISE
MAXWELL

ELSA
CHARLY
LEE
FRANK

FORD
CHRYSLER
MERCEDES-BENZ
MERCEDES-BENZ
CHRYSLER
CHRYSLER
*** NO CAR ***
FORD