Verity®
Query Language
Guide V5.0 for PeopleSoft®

November 15, 2003
Original Part Number DM0603

Verity, Incorporated
894 Ross Drive
Sunnyvale, California 94089
(408) 541-1500

Verity Benelux BV
Coltbaan 31
3439 NG Nieuwegein
The Netherlands

Table of Contents

Preface

Chapter 1 Overview

Chapter 2 Elements of Query Expressions

Chapter 3 Operators

Chapter 4 Modifiers

Chapter 5 Advanced Query Language

Appendix A Query Parsers

Index

Preface

The *Verity Query Language Reference Guide* describes how to construct simple queries with Verity query language, and how the four parsers that are included with Verity products parse those queries.

Using This Manual

The following sections briefly describe the organization of this manual and stylistic conventions used within it.

Version

The information in this manual is current as of K2 Enterprise version 5.0. The content of the manual was last modified November 15, 2003.

Organization of This Manual

This manual contains the following chapters and appendixes:

Chapter 1, “Overview”—Provides an overview to the Verity query language and the contents of the entire guide.

Chapter 2, “Elements of Query Expressions”—Provides detail on how to compose simple and complex queries using Verity query language elements, such as operators, modifiers, and syntax.

Chapter 3, “Operators”—Provides a reference to Verity query language operators.

Chapter 4, “Modifiers”—Provides a reference to Verity query language modifiers.

Chapter 5, “Advanced Query Language”—Provides a reference to Verity query language advanced (most sophisticated) operators.

Appendix A, “Query Parsers”—Provides a description of Verity query parsers.

Stylistic Conventions

The following stylistic conventions are used in this manual.

Convention	Usage
`Courier type`	Used to describe API constructs in structure member descriptions and code examples
`Courier oblique type`	Used for user-replaceable constructs
`Courier bold`	Used to denote API data type names in structure member descriptions and application names embedded in text
Palatino	Used in narrative text
italics	Used at the first introduction of a new term, and for book titles

1

Overview

The Verity query language provides a rich language for writing queries that return relevant information. Queries can be composed to search full text only or full text in combination with field information. Simple syntax, such as words and phrases separated by commas, and more complex syntax involving operators and modifiers can be used.

This chapter provides an overview of the operators and modifiers that comprise the Verity query language. Material covered includes:

- Evidence Operators
- Proximity Operators
- Relational Operators
- Concept Operators
- Modifiers
- Advanced Operators

Special queries called are included as part of the Verity query language. Topics are discussed in Chapter 2, “Elements of Query Expressions.”

Verity toolkit and server products include query parsers. For information about the available query parsers, see Appendix A, “Query Parsers.”

Evidence Operators

Evidence operators can specify either a basic word search or an intelligent word search. A basic word search finds documents that contain only the word or words specified in the query. An intelligent word search expands the query terms to create an expanded word list so that the search returns documents that contain variations of the query terms. For example, the `THESAURUS` operator selects documents that contain the word specified, as well as its synonyms.

Documents retrieved using evidence operators are not relevance-ranked unless you include the `MANY` modifier. (For examples and more detailed information, see "MANY" in Chapter 4.)

The following table briefly describes each evidence operator. For examples and more detailed descriptions, see "Operators for Searching Full Text"in Chapter 3.

Operator Name	Description
SOUNDEX	Expands the search to include the word you enter and one or more words that "sound like," or whose letter pattern is similar to, the word specified. Collections do not have sound-alike indexes by default; you must build the sound-alike indexes to use this feature.
STEM	Expands the search to include the word you enter and its variations.
THESAURUS	Expands the search to include the word you enter and its synonyms.
TYPO / N	Expands the search to include the word you enter and words similar to the query term. This operator performs "approximate pattern matching" to identify similar words.
WILDCARD	Matches wildcard characters in search strings. Certain characters automatically indicate a wildcard specification.
WORD	Performs a basic word search and selects documents that include one or more instances of the specific word you enter.

Proximity Operators

Proximity operators specify the relative location of specific words in the document; that is, specified words must be in the same phrase, paragraph, or sentence for a document to be retrieved. In the case of the NEAR and NEAR/*N* operators, retrieved documents are relevance-ranked based on the proximity of the specified words. When proximity operators are nested, use the ones with the broadest scope first. Phrases or individual words can appear within SENTENCE or PARAGRAPH operators, and SENTENCE operators can appear within PARAGRAPH operators. The following table briefly describes each proximity operator; see "Operators for Searching Full Text" in Chapter 3 for examples and more detailed descriptions.

Operator Name	Description
IN	Selects documents that contain specified values in one or more document zones. A document zone represents a region of a document, such as the document's summary, date, or body text. To search for a term only within the one or more zones upon which certain conditions have been placed, qualify the IN operator with the WHEN operator.
NEAR	Selects documents containing specified search terms. The closer the search terms are within a document, the higher the document's score.
NEAR/N	Selects documents containing two or more search terms within *N* number of words of each other, where *N* is an integer between 1 and 1024. The closer the search terms are within a document, the higher the document's score.
PARAGRAPH	Selects documents that include all of the search elements you specify within the same paragraph.
PHRASE	Selects documents that include the phrase you specify. A phrase is a grouping of two or more words that occur in a specific order.
SENTENCE	Selects documents that include all of the words you specify within the same sentence.

Relational Operators

Relational operators search document fields (such as AUTHOR) that have been defined in the collection. These operators perform a filtering function by selecting documents that contain specified field values. The fields that are used with relational operators can contain alphanumeric characters. Documents retrieved using relational operators are not relevance-ranked, and you cannot use the MANY modifier with relational operators.

A number of relational operators are available for numeric and date comparisons, including the following: = (equals), > (greater than), >= (greater than or equal to), < (less than), <= (less than or equal to). See "Operators for Searching Numeric Fields" in Chapter 3 for examples and more detailed descriptions.

The following table briefly describes the relational operators are available for text comparisons; see "Operators for Searching Text Fields" in Chapter 3 for examples and more detailed descriptions.

Operator Name	Description
CONTAINS	Selects documents by matching the character string you specify with the values stored in a specific document field.
ENDS	Selects documents by matching the character string you specify with the ending characters of the values stored in a specific document field.
MATCHES	Selects documents by matching the character string you specify with values stored in a specific document field. Documents are selected only if the search elements specified match the field value exactly. When a partial match is found, the document is not selected.
STARTS	Selects documents by matching the character string you specify with the starting characters of the values stored in a specific document field.
SUBSTRING	Selects documents by matching the character string you specify with a portion of the strings of the values stored in a specific document field.

When using operators in combination with attributes, some operators are interpreted differently than when they are used in a field search. The specific difference is as follows:

- In the following construct, the MATCHES operator is equivalent to the equals sign (=) and no wildcards are allowed:

```
<WHEN> attribute <OPERATOR> value
```

The following table shows the actual operators and values used when matching the attribute value in the query with the stored attributes in a collection's index. The use of wildcards is denoted by an asterisk (*):

Operator in the query	Actual operator used	Interpretation of the attribute's value
= or MATCHES	WORD	*value*
STARTS	WILDCARD	*value**
ENDS	WILDCARD	**value*
SUBSTRING or CONTAINS	WILDCARD	**value**

Concept Operators

Concept operators combine the meaning of search elements to identify a concept in a document. Documents retrieved using concept operators are relevance ranked. The following table briefly describes each concept operator; see “Operators for Searching Full Text” in Chapter 3 for examples and more detailed descriptions.

Operator Name	Description
ACCRUE	Selects documents that include at least one of the search elements you specify. The more search elements that are present, the higher the score will be.
ALL	Selects documents that contain all of the search elements you specify. A score of 1.00 is assigned to each retrieved document. <ALL> and <AND> are similar and they retrieve the same results. Queries using <ALL> are not relevance-ranked; all retrieval results are assigned a score of 1.00.
AND	Selects documents that contain all of the search elements you specify. An score is calculated for each retrieved document. <AND> and <ALL> are similar and they retrieve the same results. Queries using <AND> are relevance-ranked; retrieved documents are assigned a score between 0 and 1.00.
ANY	Selects documents that contain at least one of the search elements you specify. A score of 1.00 is assigned to each retrieved document. <ANY> and <OR> are similar and they retrieve the same results. Queries using <ANY> are not relevance-ranked; all retrieval results are assigned a score of 1.00.
OR	Selects documents that contain at least one of the search elements you specify. A score is calculated for each retrieved document. <OR> and <ANY> are similar and they retrieve the same results. Queries using <OR> are relevance-ranked; retrieval documents are assigned a score between 0 and 1.00.

Modifiers

Modifiers are used in conjunction with operators to change the standard behavior of an operator in some way. When specified, a modifier changes the standard behavior of an operator in some way. For example, you can use the CASE modifier with an operator to specify that the case of the search word you enter be considered a search element as well.

The following table briefly describes each modifier. For examples and more detailed descriptions, see Chapter 4, “Modifiers.”

Modifier	Description
CASE	Performs a case-sensitive search.
MANY	Incorporates the density of search words in the calculation of the relevance-ranked score.
NOT	Excludes documents containing the words or phrases.
ORDER	Specifies the order in which search elements much occur in the document.

Two syntax formats are used to specify modifiers with operators.

The first format specifies the modifier name before the operator name, as shown in the table below. This format is valid for all four types of modifiers. Certain operators are valid only with certain modifiers.

Modifier	Valid Operators	Examples
CASE	WORD WILDCARD	<CASE><WORD> iMac
MANY	WORD WILDCARD STEM SOUNDEX PHRASE SENTENCE PARAGRAPH THESAURUS	<MANY><WORD> virtual
NOT	all operators	cat <AND> dog <AND> <NOT> pet
ORDER	PARAGRAPH SENTENCE NEAR/N ALL	president <ORDER> <PARAGRAPH> washington <ORDER> <SENTENCE> ("president", "washington")

The second syntax format specifies the operator name before the modifier name, as shown in the following table. This syntax is valid only for the CASE and NOT modifiers.

Modifier	Valid Operators	Examples
CASE	WORD WILDCARD CONTAINS MATCHES STARTS ENDS SUBSTRING	author <CONTAINS/CASE>Don
NOT	all operators	author<CONTAINS/NOT>don author<STARTS/NOT>xxx

Advanced Operators

Two advanced classes of Verity operators are not used with modifiers.

The score operators (YESNO, PRODUCT, SUM, and COMPLEMENT) affect how the search engine calculates scores for retrieved documents. When a score operator is used, the search engine first calculates a separate score for each search element found in a document, and then performs a mathematical operation on the individual element scores to arrive at the final score for each document.

The natural language operators (FREETEXT and LIKE) enable you to specify search criteria using natural language syntax. The search engine uses natural language analysis to translate the query text into Verity query language expression for evaluating and scoring documents.

The following table briefly describes each advanced operator. For examples and more detailed descriptions, see Chapter 5, "Advanced Query Language."

Operator Name	Description
COMPLEMENT	Score operator. Calculates scores for documents matching a query by taking the complement (subtracting from 1) of the scores for the query's search elements.
FREETEXT	Natural language operator. Interprets text using the free text query parser, and scores documents using the resulting query expression. All retrieved documents are relevance-ranked. For information about the free text query parser, see Appendix A.
LIKE	Searches for other documents that are *like* the sample one or more documents or text passages you provide. The search engine analyzes the provided text to find the most important terms to use for the search. Retrieved documents are relevance-ranked.
PRODUCT	Score operator. Calculates scores for documents matching a query by multiplying the scores for the query's search elements together.
SUM	Score operator. Calculates scores for documents matching a query by adding together the scores for the query's search elements. The maximum sum is always 1.
YESNO	Score operator. Enables you to limit a search to only those documents matching a query, without the score of that query affecting the final scores of the documents.

2

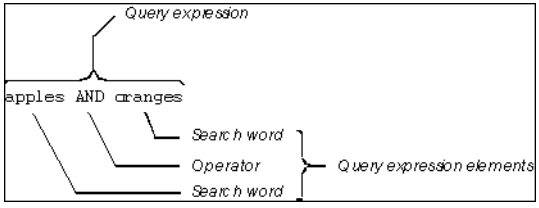
Elements of Query Expressions

This chapter describes the elements of Verity query language used to write simple query expressions. It provides the following information:

- Overview
- Simple Queries
- Explicit Queries
- Syntax Options
- Precedence Evaluation
- Delimiters in Expressions
- Special Characters
- Qualify Instance Queries

Overview

A *query expression* is any statement you enter as criteria for performing a search. The words and operators you use in a query expression comprise its *elements.*



When the simple query parser (the default parser) is used, you can state a query expression using simple or explicit syntax. The syntax you use determines whether the search words you enter will be stemmed, and whether the words that are found will contribute to relevance-ranked scoring.

For further information, see Appendix A.

Simple Queries

When you use simple syntax, the search engine implicitly interprets single words you enter as if they were preceded by the MANY modifier and the STEM operator. By implicitly applying the MANY modifier, the search engine calculates each document's score based on the *word density* it finds; the denser the occurrence of a word in a document, the higher the document's score.

As a result, the search engine relevance-ranks documents according to word density as it searches for the word you specify, as well as words that have the same stem. For example, "films," "filmed," and filming" are stemmed variations of the word "film." To search for documents containing the word "film" and its stem words, enter the word "film" using simple syntax:

```
film
```

When documents are relevance-ranked, they are listed in an order based on their relevance to your search criteria. Relevance-ranked results are presented with the most relevant documents at the top of the list.

Operator/Modifier Names

Left and right angle brackets (< >) are reserved for designating operators and modifiers. They are optional for AND, OR, and NOT, but required in all other cases.

To include a backslash (\) in a search, insert two backslashes for each backslash character. To search for "C:\bin\print," enter the following simple syntax:

```
C:\\bin\\print
```

Topic Names

For simple queries, simply enter the topic name as you would a word or phrase.

The search engine also interprets words that are topic names as topics rather than as individual words when you use simple syntax. This means that if the text you enter contains a topic name, the query corresponding to that topic is used instead of the word itself.

Automatic Case-Sensitive Searches

In previous releases, the search engine conducted case-insensitive searches by default. Now the search engine attempts to match the case-sensitivity provided in the query expression when mixed case is used. For search terms entered completely in lower-case or completely in upper-case, the search engine looks for all mixed-case variations.

Search terms with mixed case automatically become case-sensitive. For example, a query on `Apple` behaves as if you had specified `<case>Apple` (which would find only the precise string `Apple`), while a query on `apple` finds all of the following: `APPLE`, `Apple`, `apple`.

A query all in upper case does not turn on case-sensitive searching. A query on `APPLE` finds all of the following: `APPLE`, `Apple`, `apple` (as before).

The CASE modifier has the same effect as in previous releases. When used, the case-sensitivity of the query is preserved. For example, if you want to search for the term "OCX" and want to find instances of "OCX" in upper case only, you could enter the following query:

```
<CASE> <WORD> OCX
```

The search engine would interpret the above query expression to mean: find all documents containing one or more instances of the word "OCX" spelled in upper case, not mixed case.

Auto-match Phrase to Topic Name

If your query expression includes a phrase, the search engine tries to match the phrase with a topic name by substituting the spaces with hyphens. For example, if the phrase "web server" is used in a query expression, the search engine looks for a topic named "web-server." If a match is found, the topic name is used to perform the search.

Partial matches are not valid. For example, the search engine does not match the phrase "web server features" with the topic name "web-server;" it matches the topic name "web-server-features."

Explicit Queries

When you enclose individual words in double-quotation marks (“), the Verity search engine interprets those words literally. For example, by entering the word “film” explicitly in double-quotation marks, the words “films,” “filmed,” and filming” are not considered in the search. To select documents containing the word “film” without searching for its stemmed words, enter the word “film” using explicit syntax:

```
"film"
```

The following example retrieves documents that contain both the literal phrase “pharmaceutical companies” and the literal word “stock”:

```
AND ("pharmaceutical companies", "stock")
```

The AND operator does not require angle brackets because it is automatically interpreted as an operator.

The following example retrieves documents containing the phrase “black and white”:

```
<PHRASE> (black "and" white)
```

The PHRASE operator requires angle brackets, and the “and” is enclosed in double-quotation marks (“) because it is to be interpreted as a literal word, not as an operator.

Additionally, when you enter a topic name enclosed in double-quotation marks (“), the search engine will interpret the topic name as a literal word instead of as a topic. This is useful when you want to search for a word that is the same as the name of a topic.

Syntax Options

Following is a summary of Verity query language syntax options, including alternative syntax and topics, that you can use to compose query expressions. These syntax options are available for simple and explicit queries.

Using Shorthand Notation

The Verity query language provides a few alternatives you can use to specify evidence operators. In the examples below, *"word"* represents the word to be located.

Standard Query Expression	Equivalent Format
<MANY><WORD>*word*	"word"
<MANY><STEM>*word*	'word'

Specifying Topic Names Explicitly

You can specify topics in expressions in a variety of ways. Use any of the following formats to specify a topic explicitly in an expression:

{*topic_name*}

<TOPIC>*topic_name*

<TOPIC>(*topic_name*)

{*KB*:*topic_name*}

In the examples above, *topic_name* represents the name of the topic used in the expression. *KB* represents the name of the knowledge base used in the expression.

Assigning Importance (Weights) to Search Terms

You can assign a weight to each search term in a query to indicate each search term's relative importance. The weight assignment is expressed as a number between 01 and 100, where 01 represents the very lowest importance rating and 100 represents the very highest importance rating.

To specify a weight with a search term, enter the weight in brackets just before the search term, as shown below:

```
[50]test, [80]help
```

For the above example, the search engine looks for stemmed variations of the words "test" and "help" and assigns a weight of 50 to the term "test" and a weight of 80 to the term "help." Search results with the highest density of stemmed variations of the term "help" would receive the highest possible scores.

Using explicit syntax, you could enter a query expression with weights as follows:

```
<ACCRUE> ([50]<WORD>(test), [80]<WORD>(help))
```

Searching Fields for Null Values

The search engine supports searching for fields that have a null value. This means that you can perform the basic search and find all of the documents that have a null value for a particular field. You can also search for fields that are populated with a non-null value.

The methods for searching for null or populated field values are indicated in the following table.

Syntax	Description
fieldname = ""	This syntax is used to search for documents that have a null value for the field named *fieldname*. The value for *fieldname* must be a valid Verity field. If the field name given does not exist for a document, meaning the field is not defined for the document's collection, it does not match the query.
fieldname != ""	Used to search for documents that have some value for the field named *fieldname*. The value for *fieldname* must be a valid Verity field. If the field name given does not exist for a document, meaning the field is not defined for the document's collection, it does not match the query.

Precedence Evaluation

The ways that precedence rules and syntax affect the evaluation of queries are described below.

Precedence Rules

A Verity query expression is read using explicit precedence rules applying to the operators which are used. Although a query expression is read from left to right, some operators carry more weight than others; this affects the interpretation of the expression. For example, the AND operator takes precedence over the OR operator. For this reason, the following example is interpreted to mean: Look for documents that contain *b* and *c*, or documents that contain *a*.

```
a OR b AND c
```

To ensure that the OR operator is interpreted first, use parentheses as follows:

```
(a OR b) AND c
```

In general, the appropriate use of parentheses in query expressions, especially complex ones, ensures that the query expression is interpreted as intended.

Parentheses in Expressions

Parentheses indicate the order in which the directions are to be performed; information within parentheses is read first, then information outside parentheses is read next. There must be at least one space between operators and words used in the expression. The following example means: Look for documents that contain *a* and *b*, or documents that contain *c*.

```
(a AND b) OR c
```

When there are nested parentheses, start with the innermost level. The following example means: Look for documents that contain *b* or *c* as well as *a*, or that contain *d*.

```
(a AND (b OR c)) OR d
```

Prefix and Infix Notation

Words or topics that use any operator except evidence operators (SOUNDEX, STEM, THESAURUS, WILDCARD, and WORD) can be defined in prefix notation or in infix notation.

Prefix notation is a format that specifies that the operator is specified before the words or topics used with that operator. The following example means: Look for documents that contain *a* and *b*.

```
AND (a,b)
```

When prefix notation is used, precedence is explicit within the expression. The following example means: Look for documents that contain *b* and *c* first, then documents that contain *a*.

```
OR (a, AND (b,c))
```

Infix notation is a format that specifies that the operator is specified between each element within the expression. The following example means: Look for documents that contain *a* and *b* or documents that contain *c*.

```
a AND b OR c
```

When infix notation is used, precedence is implicit within the expression; for example, the AND operator takes precedence over the OR operator.

Delimiters in Expressions

Angle brackets (< >), braces ({ }), double-quotation marks ("), and backslashes (\) are used in expressions as described below.

Angle Brackets for Operators

Left and right angle brackets (< >) are reserved for designating operators and modifiers. They are optional for AND, OR, and NOT but required in all other cases. Examples in this guide appear with and without angle brackets. As the following simple syntax examples show, enter expressions either way:

```
future <AND> trends
future AND trends
```

Both expressions mean: Look for documents that contain the stemmed variations of the words “future” and “trends”.

You can also explicitly specify a topic by using <TOPIC>(*topic_name*), where *topic_name* represents the topic to be used. The following example means: Look for documents that contain elements of the topic named performing-arts and the stemmed variations of the word “acting.”

```
<TOPIC>(performing-arts) AND acting
```

Braces in Expressions

Use left and right braces ({ }) to specify a topic. The following example means: Look for documents that contain elements of the topics named philosophy and history.

```
{philosophy} AND {history}
```

Double Quotes for Reserved Words

To search for a word that is reserved as an operator (AND, OR, and NOT), enclose the word in double quotation marks. For example, to search for the phrase “black and white TV,” enter the following simple syntax:

```
black "and" white TV
```

Enclosing the word “and” in double-quotation marks (“) signifies that “and” should be considered as a word, not an operator.

Backslashes for Special Characters

To include a backslash (\) in a search, insert two backslashes for each backslash character. To search for “C:\bin\print,” enter the following simple syntax:

```
C:\\bin\\print
```

Special Characters

The following information describes how special characters are interpreted.

Characters with Special Meaning

Characters without special meaning in the Verity query language can be entered anywhere in a query. Characters with special meaning are shown in the following table.

Special Characters	Description
, ( ) [	These characters end a text token.
= > < !	These characters end a text token because they signify the start of a field operator. (! is special: != ends a token.)
‘ @ ` < { [!	These characters signify the start of a delimited token. These are terminated by the end character associated with the start character.

A backslash removes special meaning from the next character. To enter a literal backslash in a query, use two backslashes. The following examples illustrate the use of the backslash.

```
<FreeText>("\"Hello\", said Emilie.")
'Emilie\'s'
"phrase containing a backslash (\\)"
```

Punctuation in Queries

Punctuation in queries is handled by an automatic expansion mechanism, in which, for example, the string “AT&T” becomes the following:

```
<Any>("AT&T", "AT T", "AT & T")
```

Qualify Instance Queries

The qualify instance feature can be implemented through the `VdkCollectionQualifyCBFnc` call, as described in *Verity Developer's Kit Advanced Features Guide*. When implemented, users can search for instance data. The qualify instance feature can be implemented using the Verity Developer's Kit product only.

To search for the word "steve" with an instance value of 52, you enter the following query:

```
steve[52]
```

If a particular search term is to be "qualified," then the qualification follows the search term in square brackets (`[]`).

When entering a search string, to qualify an individual word or set of words in an expansion list, append the qualification to the leaf in square brackets. For example:

```
 <WORD>apple[67] <AND> <WORD>banana[44]
```

In the example above, the only documents that would pass this query would be those that had the word "apple" with an instance value of 67 and the word "banana" with an instance value of 44.

Any valid leaf term can be qualified, except a leaf using the `TYPO` or `TYPO/N` operator. Examples:

```
<STEM>orange[45]
<SOUNDEX>kiwi[50]
```

To search for instance data using weights, you must use parentheses surrounding the qualify instance part of the query. For example, the queries below will be processed:

```
[80](orange)[45])
```

The weight of 80 will be applied to the qualify instance leaf: orange[45]

3

Operators

This chapter describes Verity query language operators. These sections are included:

- Operators for Searching Full Text
- Operators for Searching Text Fields
- Operators for Searching Numeric Fields

Operators for Searching Full Text

This section describes operators used for performing full text searches. The following three tables summarize the three "families" of text search operators. The operators and examples of their use are listed in alphabetical order after the tables.

Evidence Operators

Operator	Modifiers	Automatically Relevance-ranked
SOUNDEX	MANY, NOT	No
STEM	MANY, NOT	No
THESAURUS	MANY, NOT	No
TYPO/N	NOT	
WILDCARD	CASE, MANY, NOT	No
WORD	CASE, MANY, NOT	No

Proximity Operators

Operator	Modifiers	Automatically Relevance-ranked
IN	NOT, WHEN	
NEAR	NOT	Yes
NEAR/N	NOT, ORDER	Yes
PARAGRAPH	MANY, NOT, ORDER	Yes
PHRASE	MANY, NOT	Yes
SENTENCE	MANY, NOT, ORDER	Yes

Concept Operators

Operator	Modifiers	Automatically Relevance-ranked
ACCRUE	NOT	Yes
ALL	NOT, ORDER	No
AND	NOT	Yes

Concept Operators

Operator	Modifiers	Automatically Relevance-ranked
ANY	NOT	No
OR	NOT	Yes

ACCRUE

Selects documents that include at least one of the search elements you specify. Valid search elements are two or more words or phrases. Retrieved documents are relevance-ranked.

The ACCRUE operator scores retrieved documents according to the presence of each search element in the document using "the more, the better" approach; the more search elements found in the document, the better the document's score. The following examples illustrate the search syntax.

For example, to select documents containing stemmed variations of the words "computers" and "laptops," enter any of the following:

```
computers <ACCRUE> laptops

computers, laptops

<ACCRUE> (computers, laptops)
```

ALL

Selects documents that contain all of your search elements. Retrieved documents are not relevance-ranked. Scores cannot be assigned to this operator.

For example, to select documents which contain stemmed variations of the phrase "pharmaceutical companies" and stemmed variations of the word "stock," enter the following:

```
pharmaceutical companies ALL stock
```

Only those documents that contain both search elements, or stemmed variations of them (for example, "pharmaceutical company," "stocks," etc.), are retrieved. Each retrieved document is assigned a score of 1.00.

AND

Selects documents that contain all of your search elements. Documents retrieved using the AND operator are relevance-ranked.

For example, to select documents which contain stemmed variations of the phrase "pharmaceutical companies" and stemmed variations of the word "stock," enter the following:

```
pharmaceutical companies AND stock
```

Only those documents that contain both search elements, or stemmed variations of them (for example, "pharmaceutical company," "stocks," etc.), are retrieved. A calculated score is assigned to each retrieved document.

ANY

Selects documents that contain at least one of your search elements. Retrieved documents are not relevance-ranked. Scores cannot be assigned to this operator.

For example, to select documents that contain stemmed variations of the word "election" or the phrases "national elections" or "senatorial race", enter the following:

```
election ANY national elections ANY senatorial race
```

Only those documents that contain at least one of the search elements, or a stemmed variation of at least one of them, are retrieved. Each retrieved document is assigned a score of 1.00.

IN

Selects documents that contain specified values in one or more document zones. A *document zone* represents a region of a document, such as the document's summary, date, or body text. The `IN` operator works only if document zones have been defined in your collections. If you use the `IN` operator to search collections without defined zones, no documents will be selected. Also, the zone name you specify must match the zone names defined in your collections. Consult your collection administrator to determine which zones have been defined for specific collections.

The `IN` operator can be qualified with the `WHEN` operator to search for a term only within the one or more zones upon which certain conditions have been placed. Use of the `WHEN` operator is described below.

The following query expression searches document zones named "summary" for the word "safety."

```
"safety" <IN> summary
```

To search with multiple words, phrases, or topics, enclose them in parentheses. The following query expression searches document zones named "summary" for the word "safety" and stemmed variations of the word "warning."

```
("safety", warning) <IN> summary
```

To search multiple zones, separate them with commas and enclose them in parentheses. The following query expression searches both the "summary" zone and the "title" zone for the word "safety" and stemmed variations of the word "warning."

```
("safety", warning) <IN> (summary, title)
```

You must enclose query expressions containing commas in parentheses. The following example searches the “summary” zone for the word “safety” and stemmed variations of the phrase “environmental regulation.”

```
("safety", environmental regulation) <IN> summary
```

The following query expression searches both the “summary” zone and the “title” zone for the word “safety” and stemmed variations of the phrase “environmental regulation.”

```
("safety", environmental regulation) <IN> (summary, title)
```

WHEN

Selects documents that contain specified values in one or more document zones upon which certain conditions have been placed. The following examples illustrate searching for terms within a zone upon which certain conditions have been placed.

Say you want to search for the word “here” in a zone named “A,” whose HREF attribute contains the string “verity,” and the text looks like this:

```
Our site is <A HREF = "www.verity.com">here</A>.
```

To search for the word “here” in the zone “A” when the HREF contains the string “verity,” you can write this query:

```
"here" <IN> A <WHEN> (HREF <CONTAINS> "verity")
```

A query condition for the WHEN operator must be enclosed in parentheses, as shown above. A query condition can include one or more Verity operators; it takes the form:

```
"attribute_name" <attribute_test_operator> "test_value"
```

where *attribute_test_operator* is one of the following operators: <STARTS>, <ENDS>, <CONTAINS>, <=>, or <MATCHES>. Except for =, all operators must be surrounded by angle brackets.

Attribute test operators can be combined with the combination operators <AND> or <OR>. For example, you can search for the string “IBM” in a zone named “Company,” when the attribute named “reference” is either equal to “major” or “significant” by using the following query:

```
"IBM" <IN> "Company" <WHEN> ("reference" = "major" <OR>
"reference" = "significant")
```

NEAR

Selects documents containing specified search terms within close proximity to each other. Document scores are calculated based on the relative number of words between search terms.

For example, if the search expression includes two words, and those words occur next to each other in a document (so that the region size is two words long), then the score assigned to that document is 1.0. Thus, the document with the smallest possible region containing all search terms always receives the highest score. As search terms appear further apart, the score drops toward zero. A document receives a zero score only if it does not contain all search terms.

The NEAR operator is similar to the other proximity operators in the sense that the search words you enter must be found within close proximity of one another. However, unlike other proximity operators, the NEAR operator calculates relative proximity and assigns scores based on its calculations.

To retrieve relevance-ranked documents that contain stemmed variations of the words "war" and "peace" within close proximity to each other, enter the following:

```
war <NEAR> peace
```

NEAR/N

Selects documents containing two or more words within *N* number of words of each other, where *N* is an integer. Document scores are calculated based on the relative distance of the specified words when they are separated by *N* words or less.

For example, if the search expression NEAR/5 is used to find two words within five words of each other, a document that has the specified words within three words of each other is scored higher than a document that has the specified words within five words of each other.

The *N* variable can be an integer between 1 and 1,024, where NEAR/1 searches for two words that are next to each other. If *N* is 1,000 or above, you must specify its value without commas, as in NEAR/1000. You can specify multiple search terms using multiple instances of NEAR/*N*, as long as the value of *N* is the same.

For example, to retrieve relevance-ranked documents that contain stemmed variations of the words "commute," "bicycle," "train," and "bus" within 10 words of each other, enter the following:

```
commute <NEAR/10> bicycle <NEAR/10> train <NEAR/10> bus
```

You can use the NEAR/*N* operator with the ORDER modifier to perform ordered proximity searches. For more information about the ORDER modifier, see "ORDER" in Chapter 4.

OR

Selects documents that show evidence of at least one of your search elements. Documents selected using the OR operator are relevance-ranked.

For example, to select documents that contain stemmed variations of the word "election" or the phrases "national elections" or "senatorial race", enter the following:

```
election OR national elections OR senatorial race
```

Only those documents that contain at least one of the search elements, or a stemmed variation of at least one of them, are retrieved. A calculated score is assigned to each retrieved document.

PARAGRAPH

Selects documents that include all of the search elements you specify within a paragraph. Valid search elements are two or more words or phrases. You can specify search elements in a sequential or a random order. Documents are retrieved as long as search elements appear in the same paragraph.

To retrieve relevance-ranked documents that contain stemmed variations of the word "drug" and the phrase "cancer treating" in the same paragraph, enter the following:

```
drug <PARAGRAPH> cancer treating
```

To search for three or more words or phrases, you must use the PARAGRAPH operator between each word or phrase.

You can use the PARAGRAPH operator with the ORDER modifier to perform ordered proximity searches. For more information about the ORDER modifier, see "ORDER" in Chapter 4.

PHRASE

Selects documents that include a phrase you specify. A phrase is a grouping of two or more words that occur next to each other in a specific order.

By default, two or more words separated by a space are considered to be a phrase in simple syntax. Two or more words enclosed in double quotes are also considered to be a phrase. To retrieve relevance-ranked documents that contain the phrase "mission oak," enter any of the following:

```
mission oak

"mission oak"

mission <PHRASE> oak

<PHRASE> (mission, oak)
```

SENTENCE

Selects documents that include all of the words you specify within a sentence. You can specify search elements in a sequential or a random order. Documents are retrieved as long as search elements appear in the same sentence.

To retrieve relevance -ranked documents that contain stemmed variations of the words "American," and "innovation" within the same sentence, enter the following:

```
american <SENTENCE> innovation

<SENTENCE> (american, innovation)
```

You can use the SENTENCE operator with the ORDER modifier to perform ordered proximity searches. For more information about the ORDER modifier, see "ORDER" in Chapter 4.

SOUNDEX

Selects documents that include one or more words that "sound like," or whose letter pattern is similar to, the word specified. Words must start with the same letter as the word you specify to be selected.

Your collection administrator must have configured collections to support the SOUNDEX operator. See your collection administrator for information.

For example, to retrieve documents containing a word that is close in structure to the word "sale," enter the following:

```
<SOUNDEX> sale
```

The documents retrieved will include words such as "sale," "sell," "seal," "shell," "soul," and "scale." Documents are not relevance-ranked unless the MANY modifier is used, as in:

```
<MANY><SOUNDEX> sale
```

STEM

Selects documents that include one or more variations of the search word you specify.

For example, to retrieve documents containing a variation of the word "film," enter the following:

```
 <STEM> film
```

The documents retrieved will include words such as "films," "filmed," and "filming." Documents are not relevance-ranked unless the MANY modifier is used, as in:

```
<MANY><STEM> film
```

THESAURUS

Selects documents that contain one or more synonyms of the word you specify.

For example, to retrieve documents containing synonyms of the word "altitude" enter the following:

```
<THESAURUS> altitude
```

The documents retrieved will include words such as "height" or "elevation." Documents are not relevance-ranked unless the MANY modifier is used, as in:

```
<MANY><THESAURUS> altitude
```

TYPO/N

Selects documents that contain the word you specify plus words that are similar to the query term. The TYPO/*N* operator performs "approximate pattern matching" to identify similar words. This makes it ideal for use in an environment where documents have been scanned using an Optical Character Reader (OCR).

The optional *N* variable in the operator name expresses the maximum number of errors between the query term and a matched term, a value called the error distance. If *N* is not specified, an error distance of 2 is used.

The error distance between two words is based on the calculation of errors, where an error is defined to be a character insertion, deletion, or transposition. For example, for these sets of words, the second word matches the first within an error distance of 1:

```
mouse, house (m→h)
agreed, greed (a is deleted)
cat, coat (o is inserted)
```

For the query below, documents with the words "sweeping" and "swimming" will match, since there are 3 transpositions in the word (e→i, e→m, p→m).

```
<TYPO/3> sweeping
```

Both of the queries below return the same results. Documents containing the words "swept" and "kept" match, since the "kept" word contains 1 transposition, 1 deletion.

```
<TYPO/2> swept
```

```
<TYPO> swept
```

The TYPO/*N* operator must scan the collection's word list to find candidate matching words. This makes it impractical for use in large collections (greater than 100,000 documents unless a current spanning word list is available) or in performance-sensitive environments. Performance can be improved by generating a spanning word list for the collections to be used.

NOTE: Please note these limitations. A query term specified with TYPO/*N* can have a maximum length of 32 characters. Also, TYPO/*N* is not supported with multi-byte character sets.

WILDCARD

Selects documents that contain matches to a wildcard character string. The WILDCARD operator lets you define a wildcard string, which can be used to locate related word matches in documents. A wildcard string consists of special characters.

For example, to retrieve documents that contain words such as, "pharmaceutical," "pharmacology," and "pharmacodynamics," enter the following:

```
pharmac*
```

Documents are not relevance-ranked unless the MANY modifier is used, as in:

```
<MANY> pharmac*
```

The wildcard characters "*" and "?" automatically enable wildcard searching. To use other constructs, use the WILDCARD operator explicitly with any of the characters in the following table.

Character	Function
?	Specifies one of any alphanumeric character, as in ?an, which locates "ran," "pan," "can," and "ban." It is not necessary to specify the WILDCARD operator when you use the question mark. The question mark is ignored in a set ([ ]) or in an alternative pattern ({ }).
*	Specifies zero or more of any alphanumeric character, as in corp*, which locates "corporate," "corporation," "corporal," and "corpulent." It is not necessary to specify the WILDCARD operator when you use the asterisk; do not use the asterisk to specify the first character of a wildcard string. The asterisk is ignored in a set ([ ]) or in an alternative pattern ({ }).
[ ]	Specifies one of any character in a set, as in <WILDCARD> 'c[auo]t', which locates "cat," "cut," and "cot." You must enclose the word that includes a set in backquotes (`), and a set cannot contain spaces.
{ }	Specifies one of each pattern separated by commas, as in <WILDCARD> 'bank{s,er,ing}', which locates "banks," "banker," and "banking." You must enclose the word that includes a pattern in backquotes (`), and a set cannot contain spaces.
^	Specifies one of any character not in the set, as in <WILDCARD> 'st[^oa]ck', which excludes "stock" and "stack" but locates "stick" and "stuck." The caret (^) must be the first character after the left bracket ([) that introduces a set.
-	Specifies a range of characters in a set, as in <WILDCARD> 'c[a-r]t', which locates every three-letter word from "cat" to "crt."

Searching for Nonalphanumeric Characters

Remember that you can search for nonalphanumeric characters only if the style.lex file used to create the collections you are searching is configured to recognize the characters you want to locate. Consult your collection administrator for more information.

Searching for Wildcard Characters as Literals

Provided the style.lex file is configured for the collections to be searched, you can search for a word containing a wildcard character such as "/" or "*" by preceding the wildcard character with a backslash.

For example, when you enter the following search string:

```
abc\*d
```

the engine finds five-character words matching the “abc*d” string.

When you want to match a literal backslash, you must enter two backslashes.

Searching for Special Characters as Literals

The following nonalphanumeric characters perform special, internal search engine functions, and, by default, are not treated as literals in a wildcard string:

- comma ,
- left and right parentheses ( )
- double quotation mark "
- backslash \
- at sign @
- left curly brace {
- left bracket [
- less than sign <
- backquote `

To interpret special characters as literals, you must surround the whole wildcard string in backquotes (`). For example, to search for the wildcard string “a{b”, you surround the string with backquotes, as follows:

```
<WILDCARD> `a{b`
```

To search for a wildcard string that includes the literal backquote character (`), you must use two backquotes together and surround the whole wildcard string in backquotes (`). For example, to search for the wildcard string “*n`t”, you can enter the following query:

```
<WILDCARD> `*n``t`
```

You can search on backquotes only if the `style.lex` file used to create the collections you are searching is configured to recognize the backquote character. Consult your collection administrator for information.

WORD

Selects documents that include one or more instances of only the word you specify without locating stemmed variations of that word.

For example, to search for documents that contain the word “rhetoric,” without also considering the words “rhetorical” and “rhetorician,” enter the following:

```
<WORD> rhetoric
```

Documents are not relevance-ranked unless the `MANY` modifier is used, as in:

```
<MANY><WORD> rhetoric
```

Operators for Searching Text Fields

This section describes operators that can be used to search document fields.

CONTAINS

Selects documents by matching the word or phrase you specify with values stored in a specific document field. Documents are selected only if the search elements you specify appear in the same sequential and contiguous order in the field value. When you use the CONTAINS operator, you specify the field name to search, and the word or phrase to locate.

With the CONTAINS operator, the words stored in a document field are interpreted as individual, sequential units. You can specify one or more of these units as search criteria. To specify multiple words, each word must be sequential and contiguous and must be separated by a blank space.

For example, the following title contains eight sequential words:

```
American Version of 'Orient Express' Offers Opulent Ride
```

The following examples demonstrate how you can use the CONTAINS operator with sequential, contiguous words to match the document title listed above, assuming it is stored in a title field:

```
TITLE <CONTAINS> American Version
```

```
TITLE <CONTAINS> Express Offers
```

The following examples show how you can use a question mark (?) to represent individual variable characters of a word and an asterisk (*) to match multiple variable characters of a word:

```
TITLE <CONTAINS> Amer* Version
```

```
TITLE <CONTAINS> Version of Or????
```

Question marks and asterisks cannot be used to represent white space that appears between words.

The CONTAINS operator does not recognize nonalphanumeric characters. The CONTAINS operator interprets nonalphanumeric characters as spaces and treats the separated values as individual units.

For example, if you have defined a dash (-) as a valid character, and you enter search criteria that include this character, as in on-line, the value is defined as two individual units, as follows:

```
TITLE <CONTAINS> on line
```

ENDS

Selects documents by matching the character string you specify with the ending characters of the values stored in a specific document field.

For example, assume a document field named AUTHOR has been defined. To select documents written by Milner, Wagner, and Faulkner, enter the following:

```
AUTHOR <ENDS> ner
```

MATCHES

Selects documents by matching the character string you specify with values stored in a specific document field. Documents are selected only if the search elements specified match the field value exactly. If a partial match is found, a document is not selected. When you use the MATCHES operator, you specify the field name to search, and the word, phrase, or number to locate.

You can use question marks (?) to represent individual variable characters within a string, and asterisks (*) to match multiple characters within a string.

For example, assume a document field named SOURCE includes the following values:

```
COMPUTER

COMPUTERWORLD

COMPUTER CURRENTS

PC COMPUTING
```

To locate documents whose source is COMPUTER, the MATCHES operator is used as follows:

```
SOURCE <MATCHES> computer
```

Here, the MATCHES operator matches COMPUTER, but not COMPUTERWORLD, COMPUTER CURRENTS, or PC COMPUTING.

To locate documents whose source is COMPUTERWORLD, the MATCHES operator is used as follows:

```
SOURCE <MATCHES> computer?????
```

Now, the MATCHES operator matches COMPUTERWORLD, since each question mark (?) represents specific character positions within the string. COMPUTER and COMPUTER CURRENTS are not matched, because their character strings do not match the length represented by the question marks.

To locate documents whose sources are COMPUTER, COMPUTERWORLD, and COMPUTER CURRENTS, the MATCHES operator is used as follows:

```
SOURCE <MATCHES> computer*
```

Here, the MATCHES operator matches COMPUTER, COMPUTERWORLD, and COMPUTER CURRENTS, since the asterisk (*) represents zero or more variable characters at the end of the string.

To locate documents whose sources include COMPUTER, COMPUTERWORLD, COMPUTER CURRENTS, and PC COMPUTING, the MATCHES operator can be used as follows:

```
SOURCE <MATCHES> *comput*
```

Now, the MATCHES operator matches all four occurrences, since the asterisk (*) represents a string of characters of any length.

STARTS

Selects documents by matching the character string you specify with the starting characters of the values stored in a specific document field.

For example, assume a document field named REPORTER has been defined. To retrieve documents written by Jack, Jackson, and Jacks, enter the following:

```
REPORTER <STARTS> jack
```

SUBSTRING

Selects documents by matching the character string you specify with a portion of the strings of the values stored in a specific document field. The characters that comprise the string can occur at the beginning of a field value, within a field value, or at the end of a field value.

For example, assume a document field named TITLE has been defined. To retrieve documents whose titles contain words such as “solution,” “resolution,” “solve,” and “resolve,” enter the following:

```
TITLE <SUBSTRING> sol
```

Operators for Searching Numeric Fields

The sections below describe operators used to search numeric and date fields.

= (Equals)

Selects documents whose document field values are exactly the same as the search string you specify.

For example, assume a document field named ORGNO has been defined as the number of the organization that wrote the document. To select only those documents written by organization 104, enter the following:

```
ORGNO = 104
```

!= (Not Equals)

Selects documents whose document field values do not match the search string you specify.

For example, to search for documents with ORGNO field values not equal to 104, use this query:

```
ORGNO != 104
```

Another query using the NOT modifier could be used to perform the same search:

```
<NOT>(ORGNO=104)
```

Although using the NOT modifier returns the same results as using the != operator, the != operator functions much more efficiently.

> (Greater Than)

Selects documents whose document field values are greater than the search string you specify.

For example, assume a document field named REVISION has been defined. To select only those documents that have been revised more than three times, enter the following:

```
REVISION > 3
```

>= (Greater Than Or Equal To)

Selects documents whose document field values are greater than or equal to the search string you specify.

For example, assume a document field named REVISION has been defined. To select only those documents that have been revised three times or more, enter the following:

```
REVISION >= 3
```

< (Less Than)

Selects documents whose document field values are less than the search string you specify.

For example, assume a document field named PAGES has been defined. To select only those documents less than five pages long, enter the following:

```
PAGES < 5
```

<= (Less Than Or Equal To)

Selects documents whose document field values are less than or equal to the search string you specify.

For example, assume a document field named DATE has been defined. To select only those documents dated prior to and including February 14, 1991, enter the following:

```
DATE <= 02-14-91
```

4

Modifiers

This chapter describes Verity query language modifiers. These modifiers can be combined with operators to compose a query expression:

- CASE
- MANY
- NOT
- ORDER

CASE

Use the CASE modifier with the WORD or WILDCARD operator to perform a case-sensitive search, based on the case of the word or phrase specified. The CASE modifier is not valid with the SOUNDEX and STEM operators.

To use the CASE modifier, you simply enter the search word or phrase as you wish it to appear in retrieved documents — in all uppercase letters, in mixed uppercase and lowercase letters, or in all lowercase letters.

For example, to retrieve documents that contain the word “iMac” in mixed uppercase and lowercase letters, you would enter:

```
<CASE> <WORD> iMac
```

Only those documents that contain the word “iMac” will be selected. Occurrences of “imac,” “Imac,” or “IMAC” will not be selected.

When mixed uppercase and lowercase characters are included in a query, the search engine finds case-sensitive matches.

MANY

Counts the density of words, stemmed variations, or phrases in a document, and produces a relevance-ranked score for retrieved documents. The more occurrences of a word, stem, or phrase proportional to the amount of document text, the higher the score of that document when retrieved. Because the MANY modifier considers density in proportion to document text, a longer document that contains more occurrences of a word can score lower than a shorter document that contains fewer occurrences. You can use the MANY modifier with these operators: WORD, WILDCARD, STEM, SOUNDEX, PHRASE, SENTENCE, PARAGRAPH.

For example, to select documents based on the density of stemmed variations of the word "apple," you would enter:

```
<MANY> <STEM> apple
```

To select documents based on the density of the phrase "mission oak," you would enter:

```
<MANY> mission oak
```

The MANY modifier cannot be used with AND, OR, ACCRUE, or relational operators.

NOT

Use the NOT modifier with a word or phrase to exclude documents that show evidence of that word or phrase.

For example, to select only documents that contain the words “cat” and “mouse” but not the word “dog,” you would enter:

```
cat <AND> mouse <AND> <NOT> dog
```

To search for documents that contain the word “not,” enclose the word “not” in double-quotation marks (‘). For example, to search for the phrase “love not war,” enter any of the following queries.

```
<ORDER> love “not” war
“love not war”
<PHRASE> (love, “not”, war)
```

ORDER

Use the ORDER modifier to specify that search elements must occur in the same order in which they were specified in the query. If search values do not occur in the specified order in a document, the document is not selected. You can use the ORDER modifier with these operators: PARAGRAPH, SENTENCE, NEAR/*N*, and ALL.

Always place the ORDER modifier just before the operator. The following syntax examples show how you can use either simple syntax or explicit syntax to retrieve documents containing the word “president” followed by the word “washington” in the same paragraph:

Simple syntax

```
president <ORDER><PARAGRAPH> washington
```

Explicit syntax

```
<ORDER><PARAGRAPH> ("president", "washington")
```

To search for documents containing the words “diver,” “kills,” “shark” in that order within 20 words of each other, use one of the following queries:

```
diver <ORDER><NEAR/20> kills <ORDER><NEAR/20> shark
```

```
<ORDER> <NEAR/20> (diver, kills, shark)
```

You can use the NEAR/*N* operator with the ORDER modifier to duplicate the behavior of the PHRASE operator. For example, to search for documents containing the phrase “world wide web,” you can use the following syntax:

```
world <ORDER><NEAR/1> wide <ORDER><NEAR/1> web
```

To search for a word between two other words, you can use the ORDER modifier with the ALL operator, like this:

```
<ORDER><ALL>(dog, cat, squirrel)
```

The above query searches for “cat” between “dog” and “squirrel”. Stemmed variations of the words will match the query.

The query can be extended to include sub-query expressions. For example:

```
<ORDER><ALL>(dog, fat cat, squirrel)
```

The above query searches for the phrase “fat cat” between the words “dog” and “squirrel”. Again, stemmed variations of the words are considered a match.

Modifiers
ORDER

5

Advanced Query Language

This chapter describes six advanced Verity operators that are not used with modifiers. Four of them enable sophisticated combinations of query components for advanced document scoring, and two provide support for natural language analysis of query text.

It includes syntax and usage information for the following operators:

- YESNO
- PRODUCT
- SUM
- COMPLEMENT
- FREETEXT
- LIKE

These operators can be combined together or combined with other Verity query language.

Score Operators

The score operators affect how the search engine calculates scores for retrieved documents. When a score operator is used, the search engine first calculates a separate score for each search element found in a document, and then performs a mathematical operation on the individual element scores to arrive at the final score for each document.

The YESNO operator has wide application, whereas the PRODUCT, SUM, and COMPLEMENT operators are intended for use mainly by application developers who want to generate queries programmatically.

YESNO

Forces the score of an element to 1, if the element's score is nonzero. Examples help clarify this.

```
<YesNo> ("Chloe")
```

If the retrieval result of the search on "Chloe" was .75, with the YesNo operator, the result would be 1; if the retrieval result is 0, it remains 0.

This operator allows you to limit a search to only those documents matching a query, without the score of that query affecting the final scores of the documents. For example, to search among documents that contain "Chloe," with "Mead" as the determinant for ranking, you cannot simply specify the following:

```
"Chloe" <AND> "Mead"
```

because that would produce documents ranked with scores combined from both elements. The following would do what you want:

```
<YesNo> ("Chloe") <AND> "Mead"
```

If the retrieval result of the search on "Chloe" was .5 and that on "Mead" was .75, without the YesNo operator, the combined result would be .5; with the operator, however, it is .75, because the score of AND is calculated to be the minimum score of all its search elements.

PRODUCT

Calculates scores for documents matching a query by multiplying the scores for the query's search elements together. To arrive at a document's score, the search engine calculates a score for each search element and multiplies these scores together.

Following is an example of search syntax:

```
<PRODUCT> ("computers","laptops")
```

If a search on "computers" generated a score of .5 and a search on "laptops" generated a score of .75, the preceding search would produce a score of .375.

SUM

Calculates scores for documents matching a query by adding together, to a maximum of 1, the scores for the query's search elements. To arrive at a document's score, the search engine calculates a score for each search element and adds these scores together.

Following is an example query expression:

```
<SUM> ("computers","laptops")
```

If a search on "computers" generated a score of .5 and a search on "laptops" generated a score of .2, the preceding search would produce a score of .7 If a search on "computers" generated a score of .5 and a search on "laptops" generated a score of .75, the preceding search would produce a score of 1.00 (the maximum).

COMPLEMENT

Calculates scores for documents matching a query by taking the complement (subtracting from 1) the scores for the query's search elements. To arrive at a document's score, the search engine calculates a score for each search element and takes the complement of these scores.

Following is an example of search syntax:

```
<COMPLEMENT> ("computers")
```

The COMPLEMENT operator is a unary operator. It multiplies search elements as specified. The elements are combined, using the ACCRUE operator by default, to generate a single score which is then complemented. A sample query expression with two search elements is below:

```
<COMPLEMENT> ("computers","laptops")
```

In the above example, the query is evaluated as the word "computers" accrued using the ACCRUE operator with the word "laptops." The COMPLEMENT operator is applied to the result.

Natural Language Operators

The natural language operators enable you to specify search criteria using natural language syntax. The search engine uses natural language analysis to translate the query text into Verity query language expression for evaluating and scoring documents. The FREETEXT and LIKE natural language operators are intended mainly for use by application developers.

FREETEXT

Interprets text using the free text query parser and scores documents using the resulting query expression. All retrieved documents are relevance-ranked. For information about the free text query parser, refer to Appendix A.

This operator provides the functionality of the free text query parser, but allows you to combine free text queries with other search criteria using the full Verity query language. For example:

```
<FREETEXT> ( "peace negotiations in the Middle East" ) <AND>
(DATE > 01-01-96)
```

The quotation marks are required. If you want to include embedded quotes, they must be preceded with backslashes, as:

```
<FREETEXT> ( "\"Independence Day\""), ("\"The Arrival\""), science fiction" )
```

NOTE: In the case where a query or document contains only words defined as stop words in the collection `style.stp` file(s), the free text query parser uses the stop words for the query, ignoring the stop words list.

The FREETEXT operator can be combined with other operators in the same way as the ACCRUE operator.

LIKE

Searches for other documents that are *like* the sample one or more documents or text passages you provide. The search engine analyzes the provided text to find the most important terms to use for the search. If multiple samples are provided, the search engine assumes all of the samples are about a single theme and selects important terms common across the samples. Retrieved documents are relevance-ranked.

The LIKE operator accepts a single operand, called the QBE (query-by-example) specification. The QBE specification can be either the literal text of the example to query on, or it can be a specification of one or more full documents and text passages to use as positive and negative examples.

NOTE: In the case where a query or document contains only words defined as stop words in the collections `style.stp` file(s), a QBE query with the `LIKE` operator returns no results.

Syntax

Document specification is made with a series of text references enclosed in braces. The syntax for specifying references is:

```
{[name=]type:value [name=]type:value ...}
```

where:

name is either `posex` (*positive example*), or `negex` (*negative example*).

A *negative example* reduces the weights of terms when they occur in a positive example. If terms from a negative example do not exist within the positive example, the negative example has no effect. (Hence a `negex` by itself makes no sense.)

type can be one of the following:

- `VdkVgwKey`, to specify a document by external key
- `VdkDocId`, to specify a document by internal (session-specific) key,
- `File`, to specify a file containing the document text (plain text only; no HTML or other formatting)
- `Text`, to specify the text directly

value is a reference to a piece of text to use as the positive or negative example.

If *name* is not specified, *value* is assumed to be a reference to a positive example (that is, `posex` is the implied name).

The value of *value* depends on *type*:

- `VdkVgwKey` and `VdkDocId`: the document key
- `File`:

  ```
  filename[:offset:range]
  ```

  where a byte offset into the file and a byte range from that offset can be optionally specified
- `Text`: literal text.

If there is no explicit type specifier, *value* is interpreted in the following ways:

- VdkDocId if it starts with a # character
- Literal text if it starts with a quotation mark
- VdkVgwKey for all other cases

The Like operator can be combined with other operators using the same rules as for the ACCRUE operator.

Special Characters in VdkVgwKey Fields

The syntax for the LIKE operator allows VdkVgwKeys to be enclosed in quotes (either single or double) to avoid parsing confusion. This means VdkVgwKeys containing things like whitespace, curly braces, and quotes can be handled. Backslash must be used to escape quote characters and backslashes embedded in the key, as is standard for string handling.

The syntax supports the use of single quotes for enclosing literal text examples, as in {text:'sample text'}.

The syntax for text: and vdkvgwkey: references has been enhanced to allow the reference value to be enclosed in either single or double quotes, with the usual backslash escaping mechanisms for embedded backslashes and quotes.

Concerning the backslash character in document keys, follow these guidelines. When a backslash appears in a document key, you must enter two backslashes in the <LIKE> syntax. See "VdkVgwKey Fields on Windows Systems" below for important information about specifying paths on Windows systems.

Syntax examples are below:

```
<LIKE> ( "{text:'sample text'}" )
<LIKE> ( "{text:"sample text"}" )
<LIKE> ( "{text:"sample 'quote'"}" )
<LIKE> ( "{text:"sample \"quote\""}" )
<LIKE> ( "{vdkvgwkey:keyname}" )
<LIKE> ( "{vdkvgwkey:'{keyname}'}" )
<LIKE> ( "{vdkvgwkey:"{keyname}"}" )
<LIKE> ( "{vdkvgwkey:"c:\\my\\data"}" )
```

VdkVgwKey Fields on Windows Systems

To specify a VdkVgwKey including backslashes on Windows systems, you must double escape the two required backslashes. This means you must enter four backslashes, as shown in the example below:

```
<LIKE> ( "{vdkvgwkey:"c:\\\\my\\\\data"}" )
```

Examples of LIKE Expressions

The following examples illustrate uses of the LIKE operator.

Just literal text:

```
<LIKE> ("The dog ate the shoe.")
```

Explicit specification of a single positive example:

```
<LIKE> ( "{posex=vdkvgwkey:doc1}" )
```

Explicit specification of multiple positive and negative examples:

```
<LIKE> ( "{posex=vdkdocid:1234 posex=vdkvgwkey:doc1
negex=text:"stock market"}" )
```

Same as the preceding but with implied reference types:

```
<LIKE> ( "{posex=#1234 posex=doc1 negex=\"stock market\"}" )
```

Similar to the preceding but with implied posex names:

```
<LIKE> ( "{vdkdocid:1234 vdkvgwkey:doc1}" )
```

Same as the preceding, but using the most implicit syntax:

```
<LIKE> ( "{#1234 doc1}" )
```

You can combine a text reference list with literal text:

```
<LIKE> ( "{#1234 doc1} And more text" )
```

The preceding QBE specification is equivalent to this:

```
<LIKE> ( "{#1234 doc1 text: \"And more text\"}" )
```

The simplest way of specifying a single positive example by VgwKey:

```
<LIKE> ( "{doc1}" )
```

The example is in the file doc.txt, starting at the 100th byte:

```
<LIKE> ( "{posex=file:doc.txt:100:200}" )
```

To use a file reference with spaces in the file name, use single quotes, as follows:

```
<LIKE> ( "{file:‘my file.txt’}" )
```

To specify offset and length with single-quoted-filenames, use the following syntax:

```
<LIKE> ( "{file:‘my file.txt:0,5’}" )
```

Quotation marks embedded in LIKE expressions must be preceded by backslashes. The backslash indicates to the engine that the following character is supposed to be treated as a literal character.

Efficiency Considerations

In order to process a `LIKE` expression, the search engine must analyze the full text of the examples in the QBE specification. This has the potential to be time consuming, especially if the example documents are large or require expensive filtering.

The processing of `LIKE` queries can be accelerated by extracting feature vectors for documents at indexing time. Feature vectors are extracted during indexing when an appropriate entry is made in the `style.prm` file, as described in the *Verity Collection Reference Guide*. With feature vectors available in the collection, the search engine does not need to touch the original text of the example documents and `LIKE` queries are processed very efficiently.

A

Query Parsers

A user's query is interpreted by one of five predefined query parsers included with Verity products. The simple query parser is very popular since it allows users to enter simple words and phrases separated by commas, with or without additional query language.

This appendix covers these topics:

- Simple Queries
- Simple Query Parser
- Query-By-Example (QBE) Parser
- Internet-style Parser
- BooleanPlus Parser

Simple Queries

This section introduces how to write queries for interpretation by the simple query parser.

Words and Phrases Separated by Commas

A simple query is specified as words and phrases, separated by commas. To see documents about using text editors to create Web documents, start with a single-word query, such as:

```
editor
```

Your query finds all the documents that include the word “editor.” However, this search would include not only documents about text editors, but also documents about people who are editors. (You don’t have to specify the plural form, because a simple search includes stemmed variations, such as “editors.”) Documents about the Web that did not include the word “editor” would not be retrieved.

For more specific results, enter several words or phrases, separated by commas, that describe the subject more precisely, such as:

```
text editor, document, web
```

Case-sensitivity

The search engine attempts to match the case-sensitivity provided in the query expression, when mixed case is used. For search terms entered completely in lowercase or uppercase, the search engine looks for all mixed-case variations.

Search terms with mixed case automatically become case-sensitive. For example, the query of Apple behaves as if you had specified <case>Apple (which would find only the precise string Apple), while the query of apple finds all of the following: APPLE, Apple, apple.

The CASE modifier preserves case-sensitivity of the query. For example, if you want to search for the term “OCX” and want to find instances of “OCX” in uppercase only, you could enter this query:

```
<CASE> <WORD> OCX
```

The search engine would interpret the above query expression to mean: find all documents containing one or more instances of the word “OCX” spelled in uppercase, not mixed case.

How to Search Hyperlink Contents

Using the Verity operators IN and WHEN, you can search for all documents that refer to a particular HTML document by following HREF links in the source document. The following syntax can be used:

```
* <IN> A <WHEN> HREF <SUBSTRING> searchterm
```

The above query is evaluated in distinct query segments, as follows:

Query segment	Interpreted as
* <IN> A	The expression including the IN operator evaluates any contents (*) in the A tags (zones).
<WHEN> HREF <SUBSTRING> searchterm	The expression including the WHEN operator further qualifies the query for a specified HTML attribute, in this case HREF. The *searchterm* variable is a word or phrase. The SUBSTRING operator matches the character string you specify with strings in the target HREF.

The SUBSTRING operator can be substituted with the CONTAINS or MATCHES operator. These three operators have different ways of performing string comparisons. For more information about the operators, refer to “Operators.”

Simple Query Parser

The simple query parser supports searching over the full text of documents in addition to searching over collection fields and zones. Sometimes the simple parser is referred to as the “full text” parser. The simple parser interprets Verity query language.

A unique feature of the simple query parser is that it can translate a query expression supplied by the user into a more robust query form without requiring the user to specify a lot of syntax. For example, if a user enters a single word, the simple query parser applies the MANY modifier and STEM operator to the word by default. This more robust query form, specifically “<MANY> <STEM> word” causes the search engine to search for a broader range of documents containing evidence of the user’s query.

Behaviors of the simple query parser are listed below.

1. An individual word is interpreted as a stemmed word or a topic name, unless the word is surrounded by double quotation marks. When processing the search, the search engine first checks to see whether the word matches a topic name, and if a match is found, the topic is used. If a match is not found or if topics are not implemented, the word is interpreted as a stemmed word. The MANY modifier and the STEM operator are applied to a single word.
2. When a word is interpreted as a stemmed word, the search engine broadens the search to include the word itself along with the stemmed variations of the word. For example, if a user enters the word “meet” (without double quotation marks, as in: meet) in the search form and then starts a search, the search engine will look for these words: “meet,” “meets,” and “meeting.”
3. When matching a query expression (including two or more words) against topic names, spaces are interpreted as hyphens. This means that if a phrase named BIG COMPANIES is supplied as part of a query expression, the application looks for the topic BIG-COMPANIES. If a topic name match is found, the topic is used to process the search.
4. To specify a literal word so that words that have the same stem will not be considered in the search, the user can surround the word with double quotation marks. For example, to search for documents that contain the word “tropic” and not consider words that have the same stem, such as “tropics” or “tropical,” the word “tropic” needs to be surrounded with double quotation marks.
5. The PHRASE operator is applied to a phrase where a phrase is defined to be two or more words separated by spaces.
6. Queries are case-insensitive when the query terms are entered in lowercase or uppercase characters. Queries are case-sensitive when the query terms are entered in

mixed case characters. To force case-sensitive searches for words and phrases, users can use the CASE modifier in queries.

7. Special meaning is assigned to the following words in a query expression: AND, OR, NOT. These words are interpreted as Verity query language, unless they are enclosed in double quotation marks.

   For example, to search for the phrase “recycle and reuse,” ensuring that the word “and” is not interpreted as an operator, the following query can be used:

   ```
   recycle "and" reuse
   ```

   NOTE: Special characters such as “&” and “|” must also be enclosed in quotes.

8. The Verity query language can be used to perform zone and field searches. The zones that are available for searching depends on the type of documents in the collection.

Query-By-Example (QBE) Parser

The query-by-example (QBE) parser supports searching for similar documents, a search method sometimes referred to as similarity searching. The QBE parser supports searching over the full text of documents only. The QBE parser does not support searching over collection fields and zones. The QBE parser does not support Verity query language except topics as described below.

Sometimes the Verity products and documentation refer to the QBE parser as the Free Text Parser. These two parsers are identical in behavior.

Meaningful words are automatically treated as if they were preceded by the MANY modifier and the STEM operator. By implicitly applying the STEM operator, the search engine searches not only for the meaningful words themselves, but also for words that have the same stem. By implicitly applying the MANY modifier, the search engine calculates each document's score based on the word density it finds for meaningful words; the denser the occurrences of a word in a document, the higher the document's score.

By default, common words (such as "the," "has," and "for") are stripped away, and the query is built based on the more significant words (such as "personnel," "interns," "schools," and "mentors"). Therefore, the results of a query-by-example search are likely to be less precise than a search performed using the simple or BooleanPlus parser.

The QBE query parser interprets topic names as topic objects. This means that if the specified text block contains a topic name, the query expression represented by the topic is considered in the search.

Internet-style Parser

With the internet-style query parser (IQP), users can search entire documents or parts of documents (zones and fields) using a command syntax similar to the syntax used in many Web search engines.

Search Terms

In a search form enabled with the internet-style query parser, users can enter words, phrases, and plain language. The internet-style parser does not support the Verity query language (VQL).

However, if you are developing an application using the Verity Developer's Kit Application Programming Interface (VDK API), you can combine VQL and IQP syntax.

Words

To search for multiple words, separate them with spaces.

Phrases

To search for an exact phrase, surround it with double quotation marks.

A string of capitalized words is assumed to be a name. Separate a series of names with commas.

Commas aren't needed when the phrases are surrounded by quotation marks.

The following example searches for a document that contains the phrases "San Francisco" and "sourdough bread".

```
San Francisco "sourdough bread"
```

Plain Language

To search with plain language, enter a question or concept.

The Verity internet-style Query Parser identifies the important words and searches for them. For example, enter a question such as:

```
Where is the sales office in San Francisco?
```

This query produces the same results as entering:

```
sales office San Francisco
```

Including and Excluding Search Terms

You can limit searches by excluding or requiring search terms, or by limiting the areas of the document that are searched.

A minus sign (–) immediately preceding a search term (word or phrase) excludes documents containing the term.

A plus sign (+) immediately preceding a search term (word or phrase) means returned documents are guaranteed to contain the term.

If neither sign is associated with the search term, the results may include documents that do not contain the specified term as long as they meet other search criteria.

Search Scope

The internet-style parser supports searching over the full text of documents in addition to document zones and fields.

Zone Searches

The internet-style parser allows users to perform zone searches. Zones that are available for searching depend on the type of documents in the collection.

Zones are available in Markup Language documents (such as HTML and SGML) as well as Internet Message format documents (such as standard email and Usenet newsgroup messages).

To search a document zone, type the name of the zone, a colon (:) and the search term, with no spaces:

```
zone:term
```

If you enter a minus sign (–) immediately preceding `zone`, documents containing the specified term will be excluded from the search results. For example, if you enter `-zone:term`, documents containing `term` will be excluded from the results of the search of `zone`.

If you enter a plus sign (+) immediately proceeding the zone search specification, such as `+zone:term`, documents will be included in the zone search results only if the term is present.

Field Searches

The internet-style parser allows users to perform field searches. The fields that are available for searching depend on field extraction rules based on document type of documents in the collection.

To search a document field, type the name of the field, a colon (:) and the search term, with no spaces:

```
field:term
```

If you enter a minus sign (–) immediately proceeding `field`, documents containing the specified term will be excluded from the search results. For example, if you enter `–field:term`, documents containing the specified term in the specified field will be excluded from the results of the search.

If you enter a plus sign (+) immediately proceeding the field search specification, such as `+field:term`, documents will be included in the search results only if the search term is present in the specified field.

Field searches are enabled by the `enableField` parameter in a template file. This parameter, set to 0 by default, must be set to 1 to allow searching a document field.

The `enableField` is the only thing in a template file that should be changed without prior consultation with Verity Technical Support.

Template Files

Template files are located in a locale-specific subdirectory of your Verity `installdir`.:

- If you have a K2 system which uses the `englishx` locale, the templates will be located in the `k2/common/englishx` subdirectory of your Verity `installdir`.
- If you have a VDK system which uses the `englishx` locale, this directory will be located in the `vdk/common/englishx` subdirectory of your Verity `installdir`.

Template files allow the Verity engine to balance the performance of query parsing with the goal of increased relevance.

It also means you can customize the Verity engine to provide a query for different kinds of documents.

Templates are as follows:

Template Name	Filename	Description	Use
`Internet_Basic`	`basic.iqp`	Provides faster query parsing using fewer relevancy-ranking optimization techniques	Minimize search time
`Internet_Advanced`	`advanced.iqp`	Optimized to increase relevance-ranking over a wide range of document types	Documents are mostly WYSIWIG document types
`Internet_AdvancedWeb`	`advweb.iqp`	Optimized to increase relevancy-ranking of HTML-based documents	Search targets are mostly HTML documents
`Internet`	`legacy.iqp`		Backward compatibility

The **Template Name** is the identifier which is used to specify a template in the **`rcvdk`** command-line tool.

Use the `Internet_Advanced` template except under the following circumstances:

- If most searches will be directed at HTML documents, use the `Internet_Advanced` Web template.
- If search performance is unacceptable, use the `Internet_Basic` template.

Query Syntax

The query syntax is very similar to the syntax users expect to use on the Web. All queries produce valid results; therefore, be careful to form your query to produce the results you expect. Queries are interpreted according to the following rules:

- Individual search terms are separated by whitespace characters, such as a space, tab, or comma, as in the following example:

  ```
  cake recipes
  ```

- Search phrases are entered within double quotes, as in the following example:

  ```
  "chocolate cake" recipe
  ```

- Exclude terms with the negation operator, minus (-), or the `not` operator, as in the following example:

  ```
  cake recipes -rum
  ```

  or

  ```
  cake recipes not rum
  ```

- Require a compulsory term with the unary inclusion operator, plus (+), as in the following example:

  ```
  cake recipes +chocolate
  ```

  This example requires the term `chocolate` to be present.

- Require compulsory terms with the binary inclusion operator, `and`, as in the following example:

  ```
  cake recipes and chocolate
  ```

  This example requires both the term `recipes` and `chocolate` to be present.

Zone and Field Searches

You can search fields or zones by specifying *`name`*`:`*`term`*, where *`name`* is the name of the field or zone and *`term`* is an individual search term or phrase, such as

```
bakery city:"San Francisco"
```

or

```
bakery city:Sunnyvale
```

Pass-Through of Terms

Search terms are passed through to the VDK-level and are interpreted as Verity Query Language (VQL) syntax. No issues arise if the terms contain only alphabetic or numeric characters. Other kinds of characters may be interpreted by the locale. If a term contains a character that is not handled by the locale, it may be interpreted as VQL; for example, a search term that includes an asterisk (`*`) would be interpreted as a wildcard.

Stop Words

The configurable Internet-style query parser uses its own stop-word list, `qp_inet.stp`, to specify terms to ignore for natural language processing.

NOTE: You can override the "stop out" by using quotation marks around the word.

For example, the following stop words are provided in the query parser's stop word file for the `english` locale:

a	did	i	or	what
also	do	i'm	should	when
an	does	if	so	where
and	find	in	than	whether
any	for	is	that	which
am	from	it	the	who
are	get	its	there	whose
as	got	it's	to	why
at	had	like	too	will
be	has	not	want	with
but	have	of	was	would
can	how	on	were	<or>

Verity provides a populated stop-word file for the `english` and `englishx` locales; you need not modify the `qp_inet.stp` file for these locales. If you use the configurable Internet-style query parser for another locale, you must provide your own `qp_inet.stp` file containing the stop words you want to ignore in the locale. This stop word file must contain, at a minimum, the locale-equivalent words for `or` and `<or>`.

NOTE: The configurable Internet-style query parser's stop word file contains a different word list than the `vdk30.stp` word file, which is used for other purposes, such as summarization.

Testing the Templates

To see which templates are available to you, run the **`rcvdk`** command-line tool.

After the tool is loaded, enter the `x` command to enable expert mode, then enter the `qparser` command to see the list of query parsers currently available. In the following example, the tool is run from the `colls` subdirectory of a K2 data directory, and the example collection `verity_doccoll` is specified:

```
host:/users/user> rcvdk verity_doccoll
rcvdk  Verity, Inc. Version 5.0.0
Attaching to collection: verity_doccoll
Successfully attached to 1 collection.
Type 'help' for a list of commands.
RC> x
Expert mode enabled
RC> qparser
Available query parsers:
  Name                 Description
  ----                 -----------
  Simple               Simple Query Syntax
  BoolPlus             BooleanPlus Query Syntax
  Prefix               Explicit Prefix Query Syntax
  FreeText             Natural Language Query Syntax
  Internet_Basic       Basic Relevance Factors.
  Internet_Advanced    Advanced Relevance Factors.
  Internet_AdvancedWeb Advanced Relevance Factors for Web Gateway
                       Collections
  Internet             Legacy Internet Query Parser.
RC>
```

To select a query parser, enter the `qparser` command with the name of the parser:

```
qparser Internet_Basic
```

BooleanPlus Parser

The BooleanPlus query parser supports searching over the full text of documents in addition to searching over collection fields and zones. Sometimes the BooleanPlus parser is referred to as the “explicit” parser.

The BooleanPlus parser is similar to the simple parser in that it interprets all of the Verity query language and can interpret field and zone searches. Unlike the simple parser, queries can not be interpreted using the BooleanPlus parser unless explicit query syntax is used. For this reason, the BooleanPlus query parser typically is not used in end user search forms.

Index

O

P

Q

R

S

T

V

W

Z