Btree.Extract Subroutine

Description

Searches one or more Btree indexes for data matching the search criteria passed in. Returns the keys to rows having matching data.

Syntax

Btree.Extract *(srch_strng, table, dictvar, keys, option, flag)*

Parameters

The Btree.Extract subroutine has the following parameters.

All informational messages will be suppressed.

<table>
<tr><th>Parameter</th><th>Description</th></tr>
<tr><td>Srch_strng</td><td>Must end with a field mark.
The basic unit of srch_strng is a search column. If you want to search by more than one criterion, include additional search columns in srch_strng. Each search column is delimited from another by a field mark (@FM), and multiple search columns imply an And relation (conditions must be satisfied for all search columns before any key is returned). The And relation may be overridden for individual search values by prefixing each with a semicolon (;).

Searchcolumn structure Syntax

<code>searchcolumn = IndexedColumn:@VM:data1 [:@VM:data2 ...]:@FM</code>

Made up from the name of an indexed column and the search values to be located within that column.

A Btree index must have been applied to the specified row column, or an error is generated. Within the search column, search values are delimited from the indexed column name and each other by a value mark (@VM). Within the search column, search values are located based on an Or relation: a row key is returned when any of the values is found within the specified column.

The implied Or relation for the @VM-delimited search values may be forced to an And relation by prefixing any desired search value with an ampersand (&). This situation presumes a multi-valued column, because every And relation means that two values must be found before a hit is registered.</td></tr>
<tr><td>Table</td><td>Pass the name of the table to be searched.</td></tr>
<tr><td>Dictvar</td><td>Pass the file handle for the dictionary of the specified table.</td></tr>
<tr><td>Keys</td><td>Returns row keys for all rows that satisfy the criteria in srch_strng. Multiple keys are delimited with value marks.</td></tr>
<tr><td>Option</td><td>Three values for option are possible:

<table><tr><th>Option</th><th>Meaning</th></tr><tr><td>Null</td><td>All messages will display</td></tr><tr><td>E</td><td>All error messages will be suppressed.</td></tr><tr><td>S</td><td>All informational messages will be suppressed.</td></tr></table></td></tr>
<tr><td>Flag</td><td>Error codes are returned in flag. After execution, keys contains a list of keys matching the search criteria. flag returns one of several possible values, depending on the success of the search process.

<table><tr><th>Value</th><th>Meaning</th></tr><tr><td>0</td><td>The search was successful. All possible keys were returned in keys.</td></tr><tr><td>-1</td><td>The search failed, for reasons other than no keys found. This occurs, for example, when a specified column does not have a Btree index.</td></tr></table></td></tr>
</table>

Btree.Extract can look into more than one index in one call, and can also look up more than one data value. The system allows you to use the And and Or logical operators to retrieve values from the Btree index. Btree.Extract only searches Native Table indexes.

Btree.Extract also allows you to provide your own routines for preprocessing search data, as well as for your own search algorithm. Refer to "User Index Facility" and "User Index Extension to Btree.Extract" topics for more information.

You must open the associated dictionary table before calling Btree.Extract.

Note: The Btree.Extract routine will update Btree indices prior to the extract if the environment parameter for ENV_BTREE_FLUSH_ON$ is set to true. The Database Manager's Environment Settings window contains a checkbox to turn the update flag on.

See also

BTree.Read, Extract_SI_Keys, IXLOOKUP event, Update_Index, Collect.IXVals()

Example

```
/* The following code fragment opens the dictionary to a table,
then sets up a search that looks for all records having either data1 or data2 as values in the specified
column.
The routine returns this list of keys to the calling procedure. */

Declare Subroutine Btree.Extract
table = "CAR_PARTS"
Open "DICT ":table To @DICT Else
RetVal = Set_FSError()
       Return
End
Column = "PART_NAME"
data1 = "WHEEL, STANDARD"
data2 = "TIRE, BIAS PLY"
search_criteria = column:@VM:data1:@VM:data2:@FM
keylist = ""
option = ""
flag = ""
Btree.Extract(search_criteria, table, @DICT, keylist, option, flag)
Return keylist
```

Searching by NOT

The search values in srch_strng are substrings, and as such, you can modify the search relations using substring search characters. For example,

```
srch_strng = "COMPANY_NAME":@VM:"#TRUST INSURANCE":@FM
```

will find all values of the indexed column COMPANY_NAME that are not "TRUST INSURANCE".

Searching by BETWEEN

Btree.Extract supports a special search operator, **between** ("~" (tilde)). The operator **between** differs from **range** in that it is not inclusive. For example, the following search string will return row keys for all rows having ZIP codes between, but not including, 98100 and 98111:

```
srch_strng = "ZIP":@VM:"98100~98111":@FM
```

Searching by AND

As noted above, multiple search criteria are linked with an implicit And. The OpenList filter sub-statement With STATE = "CT" And CITY = "Stamford" is performed in Btree.Extract by the code:

```
"STATE":@VM:"CT":@FM:"CITY":@VM:"Stamford":@FM
```

Searching by OR

To change the implicit And to an Or, insert a semicolon (;) before each search value in the second (and subsequent) search column(s). The statement With STATE = "CT" Or CITY = "Stamford" is performed in Btree.Extract by the code:

```
"STATE":@VM:"CT":@FM:"CITY":@VM:";Stamford":@FM
```

Starting, Ending, and Containing

The greater than sign (>) and greater than or equal to signs (>=) are valid search string arguments. To extract keys of all rows with City starting with S, code the following search string:

```
srch_strng = "CITY":@VM:">=S":@FM: "CITY" : @VM : "<T" : @FM
```

The "ending with" indicator (**]**) is also a valid search string argument. To extract keys of all rows with City ending with **P**, code the following search string:

```
srch_strng = "CITY":@VM:"P]":@FM
```

The "containing" indicators (**[]**) also are valid search string arguments, allowing for a substring search. To extract keys of all rows with City containing the letter **P**, code the following search string:

```
srch_strng = "CITY":@VM:"[P]":@FM
```

User Index Extension to Btree.Extract

When the standard comparison operators are not applicable to your task, you may specify your own search algorithm using the Btree.Extract user index facility.

The Btree.Extract user index facility consists of two parts: a parsing module and a compare module. The parsing module is a pre-process to Btree.Extract that determines whether the search value should cause Btree.Extract to call a custom comparison module. The compare module is called as an alternative to the comparison logic usually done by Btree.Extract.

User Index Facility

Create a row in the SYSPTRS table called %USER.INDEX%. When Btree.Extract is run, OpenInsight looks for this row before Btree.Extract does any other processing. Be aware that when you create %USER.INDEX%, all system processing that accesses Btree.Extract will access this row and attempt to follow its specifications.

The %USER.INDEX% row must have two fields, and can have an optional third field, which is described in the following table:

Column position	Parameter	Description
1	USER. INDEX	This literal text must be the first field in USER.INDEX.

<table>
<tr>
<td>2</td>
<td>parsing_module</td>
<td>
In this field of the %USER.INDEX% row, enter the name of your own parsing routine.
The routine named in this column should examine the search data that has been passed to Btree.Extract to determine whether the custom comparison routine should be called. If you choose, this preprocessor can alter the search data.

Btree.Extract passes four arguments to parsing_module:

Parser(search_val, start_value, user_index_flag, compare_mod)

Parameters for parsing_module (which must be entered in uppercase) are as follows:

<table>
<tr><th>Parameter</th><th>Description</th></tr>
<tr><td>Search_val</td><td>Value of the srch_strng argument, as passed to Btree.Extract. It may contain special characters or patterns that trigger the custom compare module. If a custom compare module is called later in the Btree.Extract process, the value of search_val as (possibly) modified by the parsing routine will be passed to that compare module, to determine an index value hit or miss.
If srch_strng contains multiple search criteria, Btree.Extract parses and passes each one to the parsing module one at a time. All And and Or logic for multiple search criteria is therefore handled by Btree.Extract.

parsing_module is invoked for each value.</td></tr>
<tr><td>start_value</td><td>Your parsing module can use a start_value argument to pass a value to Btree.Extract indicating the point in the index from which to start examining index values. If start_value is null (the default), the search will begin at one end or the other of the Btree index, depending on the direction of the search (determined in user_index_flag, described below). If a specific value is passed, Btree.Extract begins from the point in the index where that value would be found.</td></tr>
<tr><td>user_index_flag</td><td>An argument for user_index_flag determines whether the custom compare module will be called to examine the search value. The flag can have these values:

<table>
<tr><th>Flag</th><th>Description</th></tr>
<tr><td>0</td><td>Do not use the custom compare module. (Default)</td></tr>
<tr><td>1</td><td>Use the custom compare module and read index values in ascending order.</td></tr>
<tr><td>2</td><td>Use the custom compare module and read index values in descending order.</td></tr>
</table>
</td></tr>
<tr><td>compare_mod</td><td>parsing_module returns in compare_mod the name of the compare module that is to be called by Btree.Extract. If null (the default), the compare module used is that specified in default_compare (of the %USER.INDEX% row).
If user_index_flag is 1 or 2, but no comparison module is specified in either the SYSPTRS row or in compare_mod, a "null load" error will occur when Btree.Extract is called.</td></tr>
<tr><td>default_compare</td><td>The compare module specified in this column contains custom logic you have developed to compare the search data against values in the Btree index. Based on its own comparison logic, the compare module determines whether the search data matches values in the index. Btree.Extract calls the compare module for each value in the Btree index, starting as specified in start_value.
The compare module (the value of compare below) is called by Btree.Extract with the following parameters. Compare(candidate, search_val, flag)</td></tr>
<tr><td>default_compare</td><td>Compare(candidate, search_val, flag)</td></tr>
</table>

<table>
<tr><th>Parameter</th><th>Description</th></tr>
<tr><td>candidate</td><td>The argument for candidate is the value extracted from the Btree index by Btree.Extract. The candidate value is passed to the compare module, and is used to compare against the search data.</td></tr>
<tr><td>search_val</td><td>A value for search_val has been returned by the parser. Values are compared against candidate by the custom compare routine.</td></tr>
<tr><td>flag</td><td>If the value of flag is true upon entry, the value of candidate is the last value in the current Btree leaf node (flag is otherwise false). The compare module must set flag with one of the following values:

<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>0</td><td>Successful search.</td></tr>
<tr><td>1</td><td>Unsuccessful search.</td></tr>
<tr><td>2</td><td>Terminate search.</td></tr>
<tr><td>3</td><td>Cancel search and return a null list of keys from Btree.Extract.</td></tr>
</table>
</td></tr>
</table>
</td>
</tr>
</table>

In addition to the arguments just described, the labeled Common area USERIX is available to the parsing and compare routines. There are two labeled common variables to provide additional information about the indexed values.The labeled block is defined as follows.

Common /USERIX/ UIX.SM.FLAG, UIX.DCONV

The variable UIX.SM.FLAG is a Boolean value, true if the column is left-justified sorted, and false if the column is right-justified. The variable UIX.DCONV contains the output conversion to be applied to the value to change it to its external representation (data is stored in the index in its internal representation).

Example 1 (Parser)

```
Subroutine Parser(search_val, start_value, user_index_flag,|comp_mod)
/* This subroutine establishes three independent subroutines,
each with a different purpose within the user index facility of Btree.Extract. */

/* This code is an example of a user index parser routine to intercept Soundex lookups
(it looks for values starting with "$"). */
Declare Subroutine Soundex   ;* code supplied below

Equate TRUE  To 1
Equate TRIGGER To "$"
Equate COMP_MOD To "COMPARE"

/* The following code examines the index lookup value and determines whether it is a Soundex lookup.
If not, the search value is passed through to Btree.Extract as normal. */
If search_val[1,1] = TRIGGER Then
       search_val[1,1] = ""  ;* delete the trigger character
       Soundex(search_val)     ;* convert to Soundex value
       user_index_flag = TRUE;* use custom compare in ascending
   start_value = search_val[1,1];* start at first letter
       COMP_MOD = COMP_MOD ;      * specify the compare module
End
Return
```

Example 2 (Compare)

```
Subroutine COMPARE(candidate, search_val, flag)

Declare Subroutine Soundex

Equate HIT_TRUE$ To 1
Equate HIT_FALSE$ To 0
Equate QUIT_SEARCH$ To 2

If candidate[1,1] GT search_val[1,1] Then
  flag = QUIT_SEARCH$  ;* end search if first char not same
End Else
  Soundex(candidate)    ;* convert CANDIDATE to Soundex
  If search_val Eq CANDIDATE Then
    flag = HIT_TRUE$
   End Else
    flag = HIT_FALSE$
  End
End
Return
```

Example 3 (Soundex)

```
Subroutine Soundex(soundex_value)

/* This is code for a subroutine that returns the Soundex equivalent of a single word.
First, establish the numeric code equivalents for all letters of the alphabet-vowels,
plus 'w', 'y', and 'h' are ignored. */
Equate SOUND.CODES To "01230120022455012623010202"
Equate PUNCTUATION To ".,/`';][-=<>?:~}{+_)(*&^%$#@!\|":CHAR(34)
Equate NUMERALS To "1234567890"
Equate NULL$   To ""

/* Be sure there is only one word, no punctuation and no lower case or numeric characters. */
text = Trim(soundex_value)
text = Field(text, " ", 1)
Convert PUNCTUATION To null In text
Convert NUMERALS To null In text
Convert @LOWER.CASE To @UPPER.CASE In text

/* In accordance with the Soundex algorithm, start with the 2nd character. */
first_char  = text[1,1]
text     = text[2,999]
text_length  = Len(text)
soundex_value = first_char
previous_char = NULL$

FOR loop_count=1 To text_length While Len(soundex_value) < 4
* strip off next character
    next_char = text[loop_count, 1]
    If next_char NE previous_char Then
      Convert @UPPER.CASE To SOUND_CODES In next_char
      If next_char NE 0 Then
        soundex_value := next_char
        previous_char = next_char
      End
    End
Next loop_count

/* Format is four characters in length, zero-padded at right if necessary. */
soundex_value = Fmt(soundex_value, "L(0)#4")

Return
```