What are the indexing options in KE EMu 3.1?

Overview

KE EMu has a number of indexing methods for efficient and timely access to data. An indexing method is an algorithm or set of rules to search data in an indirect way. The simplest type of indexing, known as an *exhaustive search*, is no index at all! In this case, each record in the system is read sequentially and compared against the search terms entered. If there is a match, the record is added to the set of matching records, then the next record is read. The *exhaustive search* method is very space efficient as only the data needs to be stored. However, a search may take some time to complete if there is a very large number of records (say several hours for 750,000 records).

To facilitate the search of large numbers of records, indexes are built that provide rapid access to data that match given search criteria. Indexes provide an indirect means of searching data in a judicious manner: when a search term is entered, the indexes are consulted to produce the matching records. There is a cost associated with indexing: the need to store indexing information along with the data. Thus indexing has a space overhead, in so far as indexes use disk space to hold information, with the trade-off being rapid retrieval.

There are a large number of indexing methods available to designers of databases, each one with associated pros and cons. The EMu database engine employs two flexible indexing methods to provide rapid retrieval of data from large numbers of records:

- The first is known as *linear hashing* and provides high speed key retrieval.
- The second goes by the long name of *A two level superimposed coding scheme for partial match retrieval*, (shortened to the *two level* method) and provides a general purpose framework for implementing a wide range of *term* based searches. A *term* is simply a sequence of characters that forms the basic entity for searching. For example, in word based searching (where you need only enter a word to find matching records), a *term* is a word.

This document describes the indexing methods available in EMu 3.1. The indexing that was available prior to the release of EMu 3.1 is described first, followed by a description of the additional methods in EMu 3.1. For the first time, it is also possible to adjust indexing via Registry entries. These entries allow institutions to tune indexing methods to provide the most efficient searching possible without wasting disk space on unused methods.

Existing index methods

As mentioned above, EMu uses two basic indexing methods:

- The *linear hashing* method for key based searching (that is IRN searching)
- The *two level* method for all other searching

While the *two level* method dictates how a search should be executed internally, it is a flexible method that supports a wide number of searching variations within the one framework. For example, EMu provides word based searching for all fields. Simply type in one or more words in any order into the field to be searched, and matches will be quickly retrieved. EMu also supports phonetic based searching via the @ operator. If a word is preceded with the @ character, all words that sound like the word entered will be found. Phonetic searching uses the same indexing as word based searching even though there are different results. The difference is in the definition of a *term*:

- For word based searching a *term* is a word.
- For phonetic searching a *term* is a series of numbers that represents the sound of the word (using a complex algorithm that converts a word into its basic sound groups).

Key lookup: linear hashing method

The *linear hashing* method is used to provide key based lookup. A key is a unique value used to identify a record. In EMu this is the IRN (Internal Record Number). Not only must key searching be fast, it must also enforce uniqueness. It must not be possible to have two records with the same key (IRN). The linear hashing algorithm ensures that a record can be located via the IRN in an average of 1.1 disk accesses, which is very efficient. It also ensures key uniqueness.

Many institutions have numbering sequences that uniquely identify an object, for example an Accession number. In some instances the number is almost always unique apart from a few exceptions. EMu allows some fields to be designated as unique, ensuring that when a record is saved the value in the field has not been entered before. Sometimes this check may need to be relaxed to allow "duplicate" numbers to be saved. The unique check performed on these fields (e.g. Loan Number) also uses the *linear hash* method.

Word indexing: two level method

The basic searching provided in EMu is word based. A *term* in word based searching is a sequence of alphabetic and numeric characters up to a word break character. A word break character consists of all punctuation characters except for apostrophes (') and underscores (_), and all space characters. For example the string:

```
Relax. You know you're in safe hands.
```

consists of the words:

```
relax
you
know
youre
in
safe
hands
```

Notice how the punctuation is removed and the case is converted to lowercase (searching is case insensitive). The words above are the search *terms* that can be used to locate the sample string. When word based indexing is used, extra *terms* are generated for word pairs (two consecutive words within the same sentence). For the above sample the word pairs generated are:

```
relax you
you know
know youre
youre in
in safe
safe hands
```

The word pairs provide direct support for *phrase* based searching, that is a search where the words are enclosed in double quotes (e.g. "in safe hands"). Enabling word based searching automatically provides *phrase* based searching.

Stem indexing

In many instances it may be useful to search for variations on a base word regardless of the tense or form of the word used. Consider a search for the word *election*:

- A word based search will return all occurrences of the word *election*, but it will not find *elect, elected, elector, electing*, etc.
- A stem based search, which is specified by preceding a word with a tilde (~), will find all variations of the word.

In stem based indexing the basic *term* is the root of the word (in this case *elect*). Entering any variation of the word preceded by the ~ operator will result in a search for the root word (e.g. searching for ~*elected* will result in the search term *elect*). The algorithm to convert a word to its root word is a complex one that has been refined over many years. It is not a simple truncation mechanism, otherwise ~*elect* would match *electric*. The rules used to determine the root word are effective for English text only.

Stem based indexing is not enabled on many fields in EMu. If it is not enabled, stem based searches are still possible, however the *exhaustive search* method is used. In general, fields that contain descriptive text or notes based fields are good candidates for *stem indexing*.

Phonetic indexing

When searching for names or scientific terms it is often useful to be able to search for records that contain words that *sound* like the search term. *Phonetic based* indexing provides this functionality. The basic *term* for phonetic searching is a number sequence. Each word is converted into a number sequence that encodes the basic sound groups that make up the word. The number sequence is then used for the search and words that generate the same number sequence (and hence sound the same) will be matched.

A phonetic search is specified by preceding a word with the @ character.

The algorithm used to convert a word into its basic sound group is very complex as letters can take on different sounds depending on the letters that surround them. A number of refinements to the basic algorithm have been made over many years.

Phonetic based indexing is not enabled on many fields in EMu. It is enabled on fields that typically contain names (e.g. first name and surname fields). If phonetic indexing is not enabled, phonetic based searching is still available, however the *exhaustive search* method is used.

String indexing

Occasionally you may want a search to return records where only the exact value entered in a field is matched. *String based* indexing uses the complete contents of the field as the *term* for retrieval and only records that exactly match the search terms will be returned. For example, two records have either one or the other of the following values in a suburb field:

```
Melbourne
North Melbourne
```

A word based search using `Melbourne` as the search term will return both records (as they both contain the word Melbourne). A *string based* search, using `Melbourne` will only return the first record as only one record contains `Melbourne` and nothing else.

String based searching is not used widely in EMu (as word based searching provides a more flexible and useful searching mechanism). It is enabled for the security fields (`SecCanDisplay_tab`, `SecCanEdit_tab` and `SecCanDelete_tab`) used by Record Level Security. Since Record Level Security uses group names and user names it is useful to use *string based* searching to avoid mismatches on names (e.g. on a group called `Herpetology` and another called `Herpetology Managers`).

Range searching

Some forms of data may be searched by locating all records between start and end points, otherwise known as a range search. EMu provides range based searching for a number of data types: latitude, longitude, date, time and numeric (both integer and floating point) types. Range based searches are specified using the following relational operators:

- Less than (<)
- Less than or equal to (<=)
- Greater than (>)
- Greater than or equal to (>=)

To search for all records inserted between 1 January 2006 and 28 February 2006, in the *Date Inserted* field you would specify:

```
>="1 January 2006" <="28 February 2006"
```

Note the use (and position) of the double quotes as the dates contain spaces.

Range based searching is a variation on the standard *two level* scheme used by the other indexing methods. Essentially it divides the possible values for a range based field into a series of discrete intervals. Then, for each field, value *terms* are generated to indicate into which intervals the value falls. Using these *terms* it is possible to provide efficient range based searches.

New indexing methods

EMu 3.1 introduces two new indexing methods, designed to provide increased support for wildcard searches:

- *Null based* indexing provides very fast retrieval when searching for fields that are empty/not empty.
- *Partial based* indexing offers full index support for wildcard queries that specify leading letters.

Both these methods use the *two level* scheme. They simply provide different *terms* based on the type of indexing required.

NULL indexing

Sometimes you may only be interested in whether a field contains a value or not. For example, to determine whether records have been assigned to a department, you would use the wildcard search, `!*`, in the *Department* field (`SecDepartment_tab`).

NULL based searching provides an additional *term* that indicates whether the field contains a value or not. This *term* is used to provide high speed retrieval when certain wildcard searches are used:

`!*` finds records where a field does not contain a value
`*` finds records where a field contains a value

Null based searching can also provide support for searches used to determine whether an attachment field is empty or not:

`!+` for records that do not have any attachments
+ for records that do have one or more attachments

Partial indexing

Most wildcard based searches (i.e. searches looking for patterns or part of a word) specify leading letters. For example, you may want to find all people with a surname starting with `Al*`. Since the standard EMu indexing is word based, wildcard based searches must use the *exhaustive search* method to locate matches. Such a search may take some time for very large numbers of records.

Partial based indexing provides a mechanism for providing very efficient wildcard searching where leading letters are specified (that is where one or more letters or digits appear at the start of the pattern). The method takes each word in a field and generates a *term* based on the number of leading letters indexed. Part of the partial based indexing configuration is the number of letters to be used to generate partial terms. For example, a field may be configured to provide partial indexing for the first `1,3,5` letters of each word. If we take the sentence:

```
Relax. You know you're in safe hands.
```

the following *terms* are generated for *partial based* indexing:

```
r
rel
relax
y
you
k
kno
y
you
youre
i
s
saf
h
han
hands
```

Thus if we searched for `rel*` the three letter *terms* can be searched to locate the matching records. The searching mechanism makes use of any indexing it can; for example, if you search for `re*` the single letter *terms* are used to retrieve a set of potential matches, which are then checked to see if the second letter matches. Partial based indexing provides the same level of response for wildcard based searches where leading letters are supplied as does word based searching.

Partial indexing is enabled on a small number of fields, typically fields containing names or scientific terms.

Indexing in KE EMu 3.1

EMu 3.1 is distributed with a number of indexing options already configured. In particular a number of fields that contain name based data already have *phonetic based* indexing enabled. Also many descriptive fields (e.g. Notes) have *stem based* indexing set. The indexing characteristics for fields have not been changed between EMu 3.0 and 3.1, except for the *First* and *Last* name fields in the Parties module. These fields have *null based* indexing and *partial based* indexing (1,3,5) enabled.

A new *Admin Task* has been added to EMu 3.1 that allows users in group *Admin* to view indexing information. An admin script called **emuindexing** produces a summary on a per table basis of the type and indexing options set for each column in a table. If other users need to run the script, install the following Registry entry:

```
Group|group|Table|eadmin|Admin Task|View System
Indexing|emuindexing
```

setting *group* to the name of the group that requires access.

The output of the admin task looks like:

```
Table "eparties"
        irn
                Type: Integer
                Indexing: Key
        SummaryData
                Type: Text
                Indexing: Word, Phonetic
        ExtendedData
                Type: Text
                Indexing: Word
        NamPartyType
                Type: Text
                Indexing: Word
```

Using this output the indexing set on EMu fields can easily be determined. As there may be a large number of tables in some institutions the Admin Task may take some time to process all of the information (up to 10 minutes for a large system). The indexing information may also contain columns that are not used by your institution. These columns are sub-classed columns specific to a particular client. Enabling indexing on these columns does not affect the size of the indexes generated as empty fields do not generate any indexing information (unless *null based* indexing is enabled).

Adjusting indexing

The addition of *null* and *partial based* indexing provides new mechanisms for enhancing response times for certain types of searches. Users may be tempted to enable these new methods on a large number of fields, however there is a trade-off. The more fields that have extra indexing enabled, the more disk space required to store those indexes. As stated above, EMu 3.1 does not enable any new indexing (except for two fields in the Parties module).

So how can the new methods be used? A series of new Registry entries is available that allow indexing options to be set on fields. Using these Registry entries database managers can decide which fields should have which indexing options enabled. The new entries are:

```
System|Setting|Table|table|Stem Index|colname;colname;...
System|Setting|Table|table|Phonetic
Index|colname;colname;...
System|Setting|Table|table|Null Index|colname;colname;...
System|Setting|Table|table|Partial
Index|colname=parts;colname=parts;...
```

where:

table is the name of the table in which indexing is to be set
colname is the column on which the indexing is to be applied
parts (for partial indexing only) is a comma separated list of numbers indicating the initial letters used for the partial index

To enable *null based* indexing on the *Organisation* name for example, the following Registry entry could be used:

```
System|Setting|Table|eparties|Null Index|NamOrganisation
```

You can determine the name of a field by using the *What's this?* help facility and clicking the field you want to identify. The column name will appear in the title of the field help window.

You may also want to enable *partial based* indexing on the *Organisation* and *Other Names* fields. In order to provide fast wildcard searching you may decide to use the first, first three and first five letters to produce *terms*. The following Registry entry could be used:

```
System|Setting|Table|eparties|Partial
Index|NamOrganisation=1,3,5;NamOtherNames_tab=1,3,5
```

Once the entries have been added to the Registry module, the new indexing methods will be applied when the next index rebuild is performed. For most institutions this is on the weekend. If you require the indexes earlier than the next weekend, an index rebuild should be initiated manually. If your system administrator has access to the EMu server, then **emureindex -p -f** should be executed. If not:

1. Add the following entry to the Registry: `User|emu|Table|eadmin|Admin Task|Re-index System|emureindex -p -f`
2. Login as user *emu*.
3. Run the *Re-index System* admin job.

Re-indexing may take some time and must not be performed while access to the system is required. Once the re-index begins EMu is taken off-line (users cannot log in, web access is disabled) until it completes. Generally, for larger systems, re-indexing should only be performed overnight .

The current implementation only allows index methods to be added to columns. You cannot remove indexing methods once they have been added (unless you are familiar with altering schema characteristics via *texdesign*). Removing the Registry once the index has been added does not remove it from the system until the next upgrade occurs. When EMu is upgraded the changed tables are replaced with the "standard" distribution tables. When the first re-index is run (as part of the upgrade) your extra indexing methods will be applied.

Finally, it is not possible to have both *stem* and *phonetic based* indexing on the same column. When one of these methods is enabled, the other is disabled automatically. The current implementation does not provide any mechanism for enabling or tuning *range based* indexing. It is expected that a future version of EMu will provide such a mechanism.