LargeScaleDateRangeProcessing

Alternate Strategy For Large Scale Date Indexes

There is another way to do date processing - one which I haven't really seen discussed, but which we've found serves us well.

For systems with very large numbers of dates to process - especially on systems where documents may be associated with a range of dates that they can be considered for (say, September 5, 2003 - August 9, 2005, inclusive, for example), but even on large-scale indexes of individual dates across large numbers of documents, there is another way of doing date processing which can result in greatly reduced query size and greatly increased performance - often better by an order of magnitude.

Indexing Date Fields

- If you are processing a range of dates attached to a document, index each individual date separately.
- For each date which applies to the document:
    - Format the date for YYYY (year only) and index that value.
    - Format the date for YYYYMM (year-month) and index that value.
    - Format the date for YYYYWW (year-week) and index that value.
    - Format the date for YYYYMMDD (year-month-date) and index that value.
- Once you have indexed all the dates into their appropriate representations in all of their possible fields, you are done.

On our own implementation, any one document may have several hundred dates on which that document applies to. (Each document in our system comprises a range of dates on which that document can be considered as applying to)

Searching Date Fields

- Take the date range to be applied to the search, and convert that into concrete start/end dates. "open ended" queries must be closed.
- Iterate the range; at each step in the iterator, create a TermQuery against the appropriate index.
    - If the current point in time can be incremented by a year (Calendar.DAY_OF_YEAR==1, and the iterated date which would result does not extend past the end date) then add a YYYY term for the current year to the list of terms to check and perofrm the increment.
    - If the current point in time can be incremented by a month (Calendar.DAY_OF_MONTH==1, and the iterated date which would result does not extend past the end date) then add a YYYYMM term for the current year/month to the list of terms to check and perform the increment.
    - If the current point in time can be incremented by a week (Calendar.DAY_OF_WEEK=getFirstDayOfWeek() && Calendar. WEEK_OF_YEAR!=53, and the iterated date which would result does not extend past the end date) then add a YYYYWW term for the current year/week to the list of terms to check and perform the increment.
    - If the current point in time can be incremented by a single day (current is not past the end date), add a YYYYMMDD to the list of terms to check and perform the increment.
- Build a boolean query of the above TermQueries, where each term is an OR.

Example Queries

Here are some examples of date range queries built using this strategy.

One month: from 2005/11/16 yyyymmdd:20051116 yyyymmdd:20051117 yyyymmdd:20051118 yyyymmdd:20051119 yyyymmdd:20051120 yyyyww: 200547 yyyyww:200548 yyyyww:200549 yyyymmdd:20051212 yyyymmdd:20051213 yyyymmdd:20051214 yyyymmdd:20051215 yyyymmdd:20051216

Results

As the range gets larger, the granularity widens; careful construction of the mechanisms of iteration allow for 100% coverage of the effective date ranges, with a (very) minimal number of date terms included therein.

This logic isn't always easy, and isn't always obvious (I just noticed that week increments in my current implementation, because they increment by a week, for example, may overshoot the beginning of the month - so an additional rule on a week increment is required to ensure that we stop at the first of the month when performing a week increment when the whole of the following month is within range.)

However, the rewards should be obvious: the ability to, with very little difficulty, index a large number of documents, over a wide range of dates covered, and perform efficient searches. These date queries can be easily transformed into query filters, cached with the searcher and reused for optimal performance, and result in a very small number of terms which need to be processed; the best level of granularity is chosen at each step in order to get to the next step - the system never returns documents that aren't actually correct, so no postprocessing of results is required.

It Just Works.

Extending For Time

This strategy can be extended for handling time as well; though in order to keep the number of terms to a minimum, it requires a bit of forethought. Though the above works for either single dates or documents with contain a range of dates, different strategies are needed at this point for an ideal handling of time information embedded in those documents.

For a document which contains a single date/time representation, along with YYYY, YYYYMM, YYYYWW, and YYYYDD, you'll also need to encode the HH, HHMM, and HHMMss. Ranges which cross the day border need to look at those fields only when they are less than a full day, or when partial days are involved - increment seconds, minutes, and then hours in the same strategy as used above when generating query terms.

For a document with a range for which times apply on both start and end date, do the normal date processing from the above set, but also encode, separately, the start- and end- versions of the HH, HHMM, and HHMMss fields. When building the queries, before incrementing by days, increment seconds, then minutes, then hours until you get to full date granularity, and then perform the same on the end-date once you can no longer increment by full days. The start- versions apply to the early part of the query to ramp up to the end of the first day, the end- versions apply to the late part of the query to ramp down to the individual second included.

Why Not Postprocess?

As always, postprocessing on hours/minutes-seconds could have been chosen instead; however, in situations where there are, within the general level of granularity being used, the possibility that there are many thousands of indexed terms (a large index size) postprocessing isn't always the best approach. Spending the time at index-time can be a good strategy for limiting the number of results to something sane, which can then be beneficial during sorting and the like. If postprocessing works for you, then you obviously don't need to worry - but if you find that postprocessing *doesn't* work in your environment, there is a way forward, with a bit of elbow grease.

More Information

More information on why we took this route, and contact information should you have questions, is on the blog entry at the blog entry on why I did this