SearchNumericalFields

Searching Numerical Fields

NumericRangeQuery (in Lucene Core since version 2.9)

Because Apache Lucene is a full-text search engine and not a conventional database, it cannot handle numerical ranges (e.g., field value is inside user defined bounds, even dates are numerical values). We have developed an extension to Apache Lucene that stores the numerical values in a special string-encoded format with variable precision (called trie, all numerical values like doubles, longs, Dates, floats, and ints are converted to lexicographic sortable string representations and indexed with different precisions). A range is then divided recursively into multiple intervals for searching: The center of the range is searched only with the lowest possible precision in the trie, while the boundaries are matched more exactly. This reduces the number of terms dramatically. See: http://hudson.zones.apache.org/hudson/job/Lucene-trunk/javadoc/core/org/apache/lucene/search/NumericRangeQuery.html

This dramatically improves the performance of Apache Lucene with range queries, which is no longer dependent on the index size and number of distinct values because there is an upper limit not related to any of these properties.

NumericRange'***Query (formerly TrieRange***'Query) can be used for date/time searches (if you need variable precision of date and time downto milliseconds), double searches (e.g. spatial search for latitudes or longitudes), prices (if encoded as long using cent values, doubles are not good for price values because of rounding problems). The document fields containing the trie encoded values are generated by a special NumericToken'*_Stream or simplier using the new field implementation NumericField (see http://hudson.zones.apache.org/hudson/job/Lucene-trunk/javadoc/core/org/apache/lucene/document/NumericField.html). Numeric fields can be sorted on (a special parser is included into FieldCache) and used in function queries (through FieldCache)

Other possibilities with storing numerical values stored in more readable form in index

Original post: http://www.mail-archive.com/lucene-user@jakarta.apache.org/msg07107.html
Goal: We want to do a search for something like this:

number:[1 TO 2]

and not have it return 11 or 103, etc. But, return 1.5, for example.

Answer (from: http://issues.apache.org/eyebrowse/ReadMsg?listId=30&msgNo=7103, Erik Hatcher)

Utility to pad the numbers

public class NumberUtils {
private static final DecimalFormat formatter =
new DecimalFormat("00000"); // make this as wide as you need

public static String pad(int n) {
return formatter.format👎;
}
}

Index the relevant fields using the pad function

doc.add(Field.Keyword("id", NumberUtils.pad));

Use a Custom RangeFilter

If you have a size field indexed using NumberTools build a chained RangeFilter to include a subset such as 1-1500.

```
FilteredQuery fq=new FilteredQuery(query,cstm_range("size",1L,1500L));

private static Filter cstm_range(String sfld,long lmin,long lmax)
   {
   Filter lessthn_f=RangeFilter.Less(sfld,NumberTools.longToString(lmax));
   Filter morethn_f=RangeFilter.More(sfld,NumberTools.longToString(lmin));
   Filter[] fa=new Filter[]{lessthn_f,morethn_f};

   Filter rf=new ChainedFilter(fa,ChainedFilter.AND);
   return rf;
   }
```

Consider Using a Filter

Building a Query that for a number (or a range of numbers) is just like building a Query for a word – it involves scoring based on the frequency of that word (or number) in the index which isn't usually what people want. So you may want to consider "Filtering" using the RangeFilter class instead. It can be a lot more efficient then using the RangeQuery class because it can skip all of the score related issues.

http://nagoya.apache.org/eyebrowse/BrowseList?listName=lucene-user@jakarta.apache.org&by=thread&from=943115
FelixSchwarz: The link above does not work for me. Do you mean http://mail-archives.apache.org/mod_mbox/lucene-java-user/200411.mbox/%3cPine.LNX.4.58.0411221818360.19461@hal.rescomp.berkeley.edu%3e

Create a custom QueryParser subclass:

```
public class CustomQuery_*'Parser extends QueryParser {
public CustomQuery'*_Parser(String field, Analyzer analyzer) {
super(field, analyzer);
}

protected Query getRangeQuery(String field, Analyzer analyzer,
String part1, String part2,
boolean inclusive)
throws ParseException {
if ("id".equals(field)) {
try {
int num1 = Integer.parseInt(part1);
int num2 = Integer.parseInt(part2);
return new RangeQuery(new Term(field, NumberUtils.pad(num1)),
new Term(field, NumberUtils.pad(num2)),
inclusive);
} catch (NumberFormat_*'Exception e) {
throw new ParseException(e.getMessage());
}
}

return super.getRangeQuery(field, analyzer, part1, part2,
inclusive);
}
}
```

Note: Only the "id" field is treated special, but your logic may vary.

Use the custom QueryParser

```
Custom``QueryParser parser =
new CustomQuery``Parser("field", analyzer);

Query query = parser.parse("id:[37 TO 346]");

assertEquals("padded", "id:[00037 TO 00346]",
query.toString("field"));
```

For decimals

You can use a multiplier to make sure you don't have decimals if they cause problems.(comment by sv)

Handling positive and negative numbers.

If you want a numerical field that may contain positive and negative numbers, you still need to format them as strings. What you must ensure is that for any numbers a and b, if a<b then format(a)<format(b). The problem cases are

- when one number is negative and the other is positve
- when both are negative, ie; -200 is less than -1, even though "-200" is lexocographically **greater** than "-1"

The "trick" to handle these problems are:

- use a prefix char for positive and negative numbers so that a negative string is always less than positive. '-' and '0' are suitable for this
- you have to "invert" the magnitude of negative numbers

Here is some code for a encode/decoder that does both these things for ints in the range -10000 to 9999. You could modify it to accept a double so long as you change the FORMAT appropriately.

```
private static final char NEGATIVE_PREFIX = '-';
// NB: NEGATIVE_PREFIX must be < POSITIVE_PREFIX
private static final char POSITIVE_PREFIX = '0';
public static final int MAX_ALLOWED = 9999;
public static final int MIN_ALLOWED = -10000;
private static final String FORMAT = "00000";
    /**
 * Converts a long to a String suitable for indexing.
 */
public static String encode(int i) {
    if ((i < MIN_ALLOWED) || (i > MAX_ALLOWED)) {
        throw new IllegalArgumentException("out of allowed range");
    }
        char prefix;
    if (i < 0) {
        prefix = NEGATIVE_PREFIX;
        i = MAX_ALLOWED + i + 1;
    } else {
        prefix = POSITIVE_PREFIX;
    }
        DecimalFormat fmt = new DecimalFormat(FORMAT);
    return prefix + fmt.format(i);
}
    /**
 * Converts a String that was returned by {@link #encode} back to
 * a long.
 */
public static int decode(String str) {
        char prefix = str.charAt(0);
    int i = Integer.parseInt(str.substring(1));
        if (prefix == POSITIVE_PREFIX) {
        // nop
    } else if (prefix == NEGATIVE_PREFIX) {
        i = i - MAX_ALLOWED - 1;
    } else {
        throw new NumberFormatException("string does not begin with the correct prefix");
    }
        return i;
}
```

Handling larger numbers

The code for a class for handling all possible long values is here. http://www.mail-archive.com/lucene-dev@jakarta.apache.org/msg04790.html

That code handles some special cases near Long.MIN_VALUE, and uses a large radix so that the resulting strings are "compressed".