FilteringOptions

Filtering: six ways to skin a cat

When you want to filter a query using criteria such as a date range there are a number of options available and unfortunately many lucene users often pick the worst option.

This code illustrates 6 possible options using the latest Lucene version (as at 10/Dec/2005) along with the performance timings and good/bad points about each approach.

Code

```
package lucene.filter;

import java.io.IOException;

import org.apache.lucene.index.IndexReader;
import org.apache.lucene.index.Term;
import org.apache.lucene.search.BooleanClause;
import org.apache.lucene.search.BooleanQuery;
import org.apache.lucene.search.ConstantScoreQuery;
import org.apache.lucene.search.ConstantScoreRangeQuery;
import org.apache.lucene.search.Filter;
import org.apache.lucene.search.FilteredQuery;
import org.apache.lucene.search.Hits;
import org.apache.lucene.search.IndexSearcher;
import org.apache.lucene.search.Query;
import org.apache.lucene.search.QueryFilter;
import org.apache.lucene.search.RangeFilter;
import org.apache.lucene.search.RangeQuery;
import org.apache.lucene.search.TermQuery;

/**
 * Some tests on filter performance. Shows the various ways to skin a cat here.
 * Unsurprisingly, RangeQueries are much slower and can change scores.
 * FilteredQueries look to offer most flexibility and are marginally faster than
 * searcher.search(query,filter) in my tests
 * @author Mark Harwood
 */
public class FilterPerformanceTests
{

        private static IndexReader reader;
        private static IndexSearcher searcher;

        public static void main(String[] args) throws Exception
        {
                reader=IndexReader.open("/indexes/enron");
                searcher=new IndexSearcher(reader);

                ///The query to run in all our tests
                TermQuery tq=new TermQuery(new Term("contents","drink"));

                //The filter criteria used by all tests
                String filterField="date";
                String lowerRange="20000101";
                String upperRange="20001012";
                int numQueriesPerTest=100;


                // Control test to measure query speed without
                // filters and to generally "warm up" JVM
                // Takes avg 2.34 millis per query.
                timeQuery("Plain Term query", tq, null,numQueriesPerTest);


                /*
                 * Method 1: BooleanQuery with mandatory TermQuery and RangeQuery
```

```
                *
                * avg query time:                          22.5 milliseconds
                * filter changes score:         yes
                * notes:                                  Not recommended - can throw too many clauses
exception
                *                                                 if range is too wide! Changes score
based on scarcity of
                *                                                 filter terms - often not desirable.
Unfortunately this
                *                                                 approach is used more often than
expected because it
                *                                                 is the only approach listed here
that can be expressed
                *                                                 entirely in        the Lucene
QueryParser syntax.As such it gets
                *                                                 used by people who don't read up on
the Lucene
                *                                                 programming API.
                */
               RangeQuery rq1=new RangeQuery(new Term(filterField,lowerRange),
                               new Term(filterField,upperRange),true);
               BooleanQuery bq1=new BooleanQuery();
               bq1.add(new BooleanClause(tq,BooleanClause.Occur.MUST));
               bq1.add(new BooleanClause(rq1,BooleanClause.Occur.MUST));
               timeQuery("BooleanQuery with range", bq1, null,numQueriesPerTest);


               /*
                * Method 2: TermQuery with filter passed to searcher
                *
                * avg query time:                          4.53 milliseconds
                * filter changes score:         no
                * notes:                                  Recommended.
                */
               RangeFilter rf2=new RangeFilter(filterField,lowerRange,upperRange, true,true);
               timeQuery("Query + rangefilter", tq, rf2,numQueriesPerTest);


               /*
                * Method 3: FilteredQuery using TermQuery and RangeFilter
                *
                * avg query time:                          4.38 milliseconds
                * filter changes score:         no
                * notes:                                  Recommended. Fastest option (marginally) and
also most flexible
                *                                                 in that multiple filters can be used
in different parts of the
                *                                                 one (potentially complex) Lucene
query
                */
               RangeFilter rf3=new RangeFilter(filterField,lowerRange,upperRange, true,true);
               FilteredQuery fq3=new FilteredQuery(tq,rf3);
               timeQuery("FilteredQuery with rangefilter", fq3, null,numQueriesPerTest);



               /*
                * Method 4: BooleanQuery with mandatory TermQuery and ConstantScoreQuery(takes a filter)
                *
                * avg query time:                          4.85 milliseconds
                * filter changes score:         yes - not to same extent as RangeQuery though
                * notes:                                  A way of expressing a filter as a (constant
scoring) query.
                *                                                 This is not a true filter like some
other examples because
                *                                                 these can be used on their own as a
query (ie without an
                *                                                 accompanying query)
                */
               RangeFilter rf4=new RangeFilter(filterField,lowerRange,upperRange, true,true);
               ConstantScoreQuery csq4=new ConstantScoreQuery(rf4);
```

```
                BooleanQuery bq4=new BooleanQuery();
                bq4.add(new BooleanClause(tq,BooleanClause.Occur.MUST));
                bq4.add(new BooleanClause(csq4,BooleanClause.Occur.MUST));
                timeQuery("ConstantScoreQuery ", bq4, null,numQueriesPerTest);



                /*
                 * Method 5: BooleanQuery with mandatory TermQuery and ConstantScoreRangeQuery
                 *
                 * avg query time:                          4.68 milliseconds
                 * filter changes score:         yes - not to same extent as RangeQuery though
                 * notes:                                        A slightly simpler way of coding Method 4 -
                 *                                                       wraps a RangeFilter in a
ConstantScoreQuery for us
                 */
                ConstantScoreRangeQuery crq5=new ConstantScoreRangeQuery(filterField,lowerRange,upperRange,true,
true);
                BooleanQuery bq5=new BooleanQuery();
                bq5.add(new BooleanClause(tq,BooleanClause.Occur.MUST));
                bq5.add(new BooleanClause(crq5,BooleanClause.Occur.MUST));
                timeQuery("ConstantScoreRangeQuery ", bq5, null,numQueriesPerTest);


                /*
                 * Method 6: TermQuery with filter of QueryFilter wrapping a RangeQuery
                 *
                 * avg query time:                          0.94 milliseconds *
                 * filter changes score:         no
                 * notes:                                        Not really a fair comparison - QueryFilter
runs the embedded
                 *                                                       RangeQuery once and caches the
matching docs in a bitset.
                 *                                                       All subsequent calls in this loop
reuse the bitset.
                 *                                                       In a more realistic application
setting the filtering
                 *                                                       requirements may be different with
each query request so
                 *                                                       such caching would be of no benefit
and wouldn't be recommended.
                 *                                                       The use of RangeQuery also
introduces the danger of a "Too many clauses"
                 *                                                       exception.
*/
                RangeQuery rq6=new RangeQuery(new Term(filterField,lowerRange),
                                new Term(filterField,upperRange),true);
                QueryFilter qf6=new QueryFilter(rq6);
                timeQuery("Query + QueryFilter wrapping a RangeQuery", tq, qf6,numQueriesPerTest);

                searcher.close();
                reader.close();
        }

        private static void timeQuery(String queryType,Query tq,
                        Filter filter, int numLoops) throws IOException
        {
                long start=System.currentTimeMillis();
                int numDocs=0;
                int top3Docs[]=new int[3];
                float top3Scores[]=new float[3];
                for(int l=0;l<numLoops;l++)
                {
                        Hits h = searcher.search(tq,filter);
                        numDocs=h.length();

                        for(int i=0;i<Math.min(top3Docs.length,numDocs);i++)
                        {
                                h.doc(i).get("title");
                                if(l==0)
                                {
```

```
                                        top3Docs[i]=h.id(i);
                                        top3Scores[i]=h.score(i);
                                }
                        }
                }
                long end=System.currentTimeMillis();
                float ave=((float)(end-start))/(float)numLoops;
                System.out.println(queryType+ " took avg "+ave+" millis found. numDocs="+numDocs);
                System.out.print("\t top docs[score]=");
                for(int i=0;i<top3Docs.length;i++)
                {
                        System.out.print(top3Docs[i]+"["+top3Scores[i]+"]\t");
                }
                System.out.println();
        }

}
```

Conclusion

There is definitely more than one way to go about filtering. Unfortunately using the Lucene query syntax and the QueryParser is the shortcut many people take to formulating queries and, as can be seen in the example code, the RangeQuery method it uses is rarely desirable and there are better alternatives to consider.