package com.twitter.search.common.query;

import java.util.Collections;

import java.util.IdentityHashMap;

import java.util.List;

import java.util.Map;

import java.util.function.Function;

import com.twitter.search.common.schema.base.Schema;

import com.twitter.search.queryparser.query.Query;

import com.twitter.search.queryparser.query.QueryParserException;

import com.twitter.search.queryparser.visitors.MultiTermDisjunctionRankVisitor;

import com.twitter.search.queryparser.visitors.NodeRankAnnotator;

import com.twitter.search.queryparser.visitors.QueryTreeIndex;

/\*\*

\* A helper class to collect field and query node hit attributions.

\*/

public class QueryHitAttributeHelper extends HitAttributeHelper {

private final Query annotatedQuery;

protected QueryHitAttributeHelper(HitAttributeCollector collector,

Function<Integer, String> fieldIdsToFieldNames,

IdentityHashMap<Query, Integer> nodeToRankMap,

Query annotatedQuery,

Map<Query, List<Integer>> expandedRanksMap) {

super(collector, fieldIdsToFieldNames, nodeToRankMap, expandedRanksMap);

this.annotatedQuery = annotatedQuery;

}

/\*\*

\* Constructor specific for com.twitter.search.queryParser.query.Query

\*

\* This helper visits a parsed query to construct a node-to-rank mapping,

\* and uses a schema to determine all of the possible fields to be tracked.

\* A collector is then created.

\*

\* @param query the query for which we will collect hit attribution.

\* @param schema the indexing schema.

\*/

public static QueryHitAttributeHelper from(Query query, final Schema schema)

throws QueryParserException {

IdentityHashMap<Query, Integer> nodeToRankMap;

Query annotatedQuery;

// First see if the query already has node rank annotations on it. If so, we'll just use those

// to identify query nodes.

// We enforce that all provided ranks are in the range of [0, N-1] so not to blow up the size

// of the collection array.

QueryRankVisitor rankVisitor = new QueryRankVisitor();

if (query.accept(rankVisitor)) {

nodeToRankMap = rankVisitor.getNodeToRankMap();

annotatedQuery = query;

} else {

// Otherwise, we will assign all nodes in-order ranks, and use those to track per-node hit

// attribution

QueryTreeIndex queryTreeIndex = QueryTreeIndex.buildFor(query);

NodeRankAnnotator annotator = new NodeRankAnnotator(queryTreeIndex.getNodeToIndexMap());

annotatedQuery = query.accept(annotator);

nodeToRankMap = annotator.getUpdatedNodeToRankMap();

}

// Extract ranks for multi\_term\_disjunction operators

MultiTermDisjunctionRankVisitor multiTermDisjunctionRankVisitor =

new MultiTermDisjunctionRankVisitor(Collections.max(nodeToRankMap.values()));

annotatedQuery.accept(multiTermDisjunctionRankVisitor);

Map<Query, List<Integer>> expandedRanksMap =

multiTermDisjunctionRankVisitor.getMultiTermDisjunctionRankExpansionsMap();

return new QueryHitAttributeHelper(

new HitAttributeCollector(),

(fieldId) -> schema.getFieldName(fieldId),

nodeToRankMap,

annotatedQuery,

expandedRanksMap);

}

public Query getAnnotatedQuery() {

return annotatedQuery;

}

}