package com.twitter.search.earlybird.queryparser;

import java.util.Arrays;

import java.util.Collection;

import java.util.Collections;

import java.util.HashSet;

import java.util.List;

import java.util.Map;

import java.util.Set;

import java.util.TreeSet;

import javax.annotation.Nullable;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.base.Functions;

import com.google.common.base.Optional;

import com.google.common.base.Preconditions;

import com.google.common.collect.ImmutableList;

import com.google.common.collect.ImmutableMap;

import com.google.common.collect.Lists;

import com.google.common.collect.Sets;

import org.apache.lucene.search.BooleanClause;

import org.apache.lucene.search.BooleanClause.Occur;

import org.apache.lucene.search.BooleanQuery;

import org.apache.lucene.search.BoostQuery;

import org.apache.lucene.search.MatchNoDocsQuery;

import org.apache.lucene.search.PhraseQuery;

import org.apache.lucene.search.Query;

import org.apache.lucene.search.TermQuery;

import org.locationtech.spatial4j.shape.Point;

import org.locationtech.spatial4j.shape.Rectangle;

import org.locationtech.spatial4j.shape.impl.PointImpl;

import org.locationtech.spatial4j.shape.impl.RectangleImpl;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.decider.Decider;

import com.twitter.search.common.constants.QueryCacheConstants;

import com.twitter.search.common.decider.DeciderUtil;

import com.twitter.search.common.encoding.features.ByteNormalizer;

import com.twitter.search.common.indexing.thriftjava.ThriftGeoLocationSource;

import com.twitter.search.common.metrics.SearchCounter;

import com.twitter.search.common.query.BoostUtils;

import com.twitter.search.common.query.FieldWeightUtil;

import com.twitter.search.common.query.FilteredQuery;

import com.twitter.search.common.query.HitAttributeHelper;

import com.twitter.search.common.query.MappableField;

import com.twitter.search.common.schema.ImmutableSchema;

import com.twitter.search.common.schema.SchemaUtil;

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

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

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

import com.twitter.search.common.schema.earlybird.EarlybirdCluster;

import com.twitter.search.common.schema.earlybird.EarlybirdFieldConstants;

import com.twitter.search.common.schema.earlybird.EarlybirdFieldConstants.EarlybirdFieldConstant;

import com.twitter.search.common.schema.earlybird.EarlybirdThriftDocumentBuilder;

import com.twitter.search.common.schema.earlybird.EarlybirdThriftDocumentUtil;

import com.twitter.search.common.schema.thriftjava.ThriftCSFType;

import com.twitter.search.common.search.TerminationTracker;

import com.twitter.search.common.search.termination.QueryTimeout;

import com.twitter.search.common.util.analysis.IntTermAttributeImpl;

import com.twitter.search.common.util.analysis.LongTermAttributeImpl;

import com.twitter.search.common.util.spatial.GeohashChunkImpl;

import com.twitter.search.common.util.text.HighFrequencyTermPairs;

import com.twitter.search.common.util.text.NormalizerHelper;

import com.twitter.search.earlybird.common.config.EarlybirdConfig;

import com.twitter.search.earlybird.common.userupdates.UserScrubGeoMap;

import com.twitter.search.earlybird.common.userupdates.UserTable;

import com.twitter.search.earlybird.partition.MultiSegmentTermDictionaryManager;

import com.twitter.search.earlybird.querycache.CachedFilterQuery;

import com.twitter.search.earlybird.querycache.QueryCacheManager;

import com.twitter.search.earlybird.search.queries.CSFDisjunctionFilter;

import com.twitter.search.earlybird.search.queries.DocValRangeFilter;

import com.twitter.search.earlybird.search.queries.FeatureValueInAcceptListOrUnsetFilter;

import com.twitter.search.earlybird.search.GeoQuadTreeQueryBuilder;

import com.twitter.search.earlybird.search.queries.MatchAllDocsQuery;

import com.twitter.search.earlybird.search.queries.RequiredStatusIDsFilter;

import com.twitter.search.earlybird.search.queries.SinceMaxIDFilter;

import com.twitter.search.earlybird.search.queries.SinceUntilFilter;

import com.twitter.search.earlybird.search.queries.TermQueryWithSafeToString;

import com.twitter.search.earlybird.search.queries.UserFlagsExcludeFilter;

import com.twitter.search.earlybird.search.queries.UserScrubGeoFilter;

import com.twitter.search.earlybird.search.queries.UserIdMultiSegmentQuery;

import com.twitter.search.earlybird.search.relevance.MinFeatureValueFilter;

import com.twitter.search.earlybird.search.relevance.ScoreFilterQuery;

import com.twitter.search.earlybird.search.relevance.scoring.ScoringFunctionProvider;

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

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

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

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

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

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

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

import com.twitter.search.queryparser.query.annotation.Annotation;

import com.twitter.search.queryparser.query.annotation.FloatAnnotation;

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

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

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

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

import com.twitter.search.queryparser.util.GeoCode;

import com.twitter.service.spiderduck.gen.LinkCategory;

import com.twitter.tweetypie.thriftjava.ComposerSource;

/\*\*

\* Visitor for {@link com.twitter.search.queryparser.query.Query}, which produces a Lucene

\* Query ({@link Query}).

\*/

public class EarlybirdLuceneQueryVisitor extends SearchQueryVisitor<Query> {

private static final Logger LOG = LoggerFactory.getLogger(EarlybirdLuceneQueryVisitor.class);

@VisibleForTesting

static final String UNSUPPORTED\_OPERATOR\_PREFIX = "unsupported\_query\_operator\_";

private static final String SMILEY\_FORMAT\_STRING = "\_\_has\_%s\_smiley";

private static final String PHRASE\_WILDCARD = "\*";

private static final float DEFAULT\_FIELD\_WEIGHT = 1.0f;

private static final SearchCounter SINCE\_TIME\_INVALID\_INT\_COUNTER =

SearchCounter.export("EarlybirdLuceneQueryVisitor\_since\_time\_invalid\_int");

private static final SearchCounter UNTIL\_TIME\_INVALID\_INT\_COUNTER =

SearchCounter.export("EarlybirdLuceneQueryVisitor\_until\_time\_invalid\_int");

private static final SearchCounter NUM\_QUERIES\_BELOW\_MIN\_ENGAGEMENT\_THRESHOLD =

SearchCounter.export(

"EarlybirdLuceneQueryVisitor\_num\_queries\_below\_min\_engagement\_threshold");

private static final SearchCounter NUM\_QUERIES\_ABOVE\_MIN\_ENGAGEMENT\_THRESHOLD =

SearchCounter.export(

"EarlybirdLuceneQueryVisitor\_num\_queries\_above\_min\_engagement\_threshold");

private static final SearchOperator OPERATOR\_CACHED\_EXCLUDE\_ANTISOCIAL\_AND\_NATIVERETWEETS =

new SearchOperator(SearchOperator.Type.CACHED\_FILTER,

QueryCacheConstants.EXCLUDE\_ANTISOCIAL\_AND\_NATIVERETWEETS);

private static final Map<String, List<SearchOperator>> OPERATORS\_BY\_SAFE\_EXCLUDE\_OPERAND =

ImmutableMap.of(

SearchOperatorConstants.TWEET\_SPAM, ImmutableList.of(

new SearchOperator(SearchOperator.Type.DOCVAL\_RANGE\_FILTER,

"extended\_encoded\_tweet\_features.label\_spam\_flag", "0", "1"),

new SearchOperator(SearchOperator.Type.DOCVAL\_RANGE\_FILTER,

"extended\_encoded\_tweet\_features.label\_spam\_hi\_rcl\_flag", "0", "1"),

new SearchOperator(SearchOperator.Type.DOCVAL\_RANGE\_FILTER,

"extended\_encoded\_tweet\_features.label\_dup\_content\_flag", "0", "1")),

SearchOperatorConstants.TWEET\_ABUSIVE, ImmutableList.of(

new SearchOperator(SearchOperator.Type.DOCVAL\_RANGE\_FILTER,

"extended\_encoded\_tweet\_features.label\_abusive\_flag", "0", "1")),

SearchOperatorConstants.TWEET\_UNSAFE, ImmutableList.of(

new SearchOperator(SearchOperator.Type.DOCVAL\_RANGE\_FILTER,

"extended\_encoded\_tweet\_features.label\_nsfw\_hi\_prc\_flag", "0", "1"))

);

private static final ImmutableMap<String, FieldWeightDefault> DEFAULT\_FIELDS =

ImmutableMap.of(EarlybirdFieldConstant.TEXT\_FIELD.getFieldName(),

new FieldWeightDefault(true, DEFAULT\_FIELD\_WEIGHT));

// All Earlybird fields that should have geo scrubbed tweets filtered out when searched.

// See go/realtime-geo-filtering

@VisibleForTesting

public static final List<String> GEO\_FIELDS\_TO\_BE\_SCRUBBED = Arrays.asList(

EarlybirdFieldConstant.GEO\_HASH\_FIELD.getFieldName(),

EarlybirdFieldConstant.PLACE\_FIELD.getFieldName(),

EarlybirdFieldConstant.PLACE\_ID\_FIELD.getFieldName(),

EarlybirdFieldConstant.PLACE\_FULL\_NAME\_FIELD.getFieldName(),

EarlybirdFieldConstant.PLACE\_COUNTRY\_CODE\_FIELD.getFieldName());

// Geo scrubbing doesn't remove user profile location, so when using the geo location type filters

// we only need to filter out geo scrubbed tweets for the geo location types other than

// ThriftGeoLocationSource.USER\_PROFILE.

// Separately, we also need to filter out geo scrubbed tweets for the place\_id filter.

private static final List<String> GEO\_FILTERS\_TO\_BE\_SCRUBBED = Arrays.asList(

EarlybirdFieldConstants.formatGeoType(ThriftGeoLocationSource.GEOTAG),

EarlybirdFieldConstants.formatGeoType(ThriftGeoLocationSource.TWEET\_TEXT),

EarlybirdThriftDocumentUtil.formatFilter(

EarlybirdFieldConstant.PLACE\_ID\_FIELD.getFieldName()));

// queries whose parents are negated.

// used to decide if a negated query is within a negated parent or not.

private final Set<com.twitter.search.queryparser.query.Query> parentNegatedQueries =

Sets.newIdentityHashSet();

private final ImmutableSchemaInterface schemaSnapshot;

private final ImmutableMap<String, FieldWeightDefault> defaultFieldWeightMap;

private final QueryCacheManager queryCacheManager;

private final UserTable userTable;

private final UserScrubGeoMap userScrubGeoMap;

@Nullable

private final TerminationTracker terminationTracker;

private final Map<MappableField, String> mappableFieldMap;

private final MultiSegmentTermDictionaryManager multiSegmentTermDictionaryManager;

private final Decider decider;

private final EarlybirdCluster earlybirdCluster;

private float proximityPhraseWeight = 1.0f;

private int proximityPhraseSlop = 255;

private ImmutableMap<String, Float> enabledFieldWeightMap;

private Set<String> queriedFields;

// If we need to accumulate and collect per-field and per query node hit attribution information,

// this will have a mapping between the query nodes and their unique ranks, as well as the

// attribute collector.

@Nullable

private HitAttributeHelper hitAttributeHelper;

@Nullable

private QueryTimeout queryTimeout;

public EarlybirdLuceneQueryVisitor(

ImmutableSchemaInterface schemaSnapshot,

QueryCacheManager queryCacheManager,

UserTable userTable,

UserScrubGeoMap userScrubGeoMap,

EarlybirdCluster earlybirdCluster,

Decider decider) {

this(schemaSnapshot, queryCacheManager, userTable, userScrubGeoMap, null, DEFAULT\_FIELDS,

Collections.emptyMap(), null, decider, earlybirdCluster, null);

}

public EarlybirdLuceneQueryVisitor(

ImmutableSchemaInterface schemaSnapshot,

QueryCacheManager queryCacheManager,

UserTable userTable,

UserScrubGeoMap userScrubGeoMap,

@Nullable TerminationTracker terminationTracker,

Map<String, FieldWeightDefault> fieldWeightMap,

Map<MappableField, String> mappableFieldMap,

MultiSegmentTermDictionaryManager multiSegmentTermDictionaryManager,

Decider decider,

EarlybirdCluster earlybirdCluster,

QueryTimeout queryTimeout) {

this.schemaSnapshot = schemaSnapshot;

this.defaultFieldWeightMap = ImmutableMap.copyOf(fieldWeightMap);

this.enabledFieldWeightMap = FieldWeightDefault.getOnlyEnabled(defaultFieldWeightMap);

this.queryCacheManager = queryCacheManager;

this.userTable = userTable;

this.userScrubGeoMap = userScrubGeoMap;

this.mappableFieldMap = Preconditions.checkNotNull(mappableFieldMap);

this.terminationTracker = terminationTracker;

this.multiSegmentTermDictionaryManager = multiSegmentTermDictionaryManager;

this.decider = decider;

this.earlybirdCluster = earlybirdCluster;

this.queryTimeout = queryTimeout;

this.queriedFields = new TreeSet<>();

}

public ImmutableMap<String, Float> getEnabledFieldWeightMap() {

return enabledFieldWeightMap;

}

public ImmutableMap<String, FieldWeightDefault> getDefaultFieldWeightMap() {

return defaultFieldWeightMap;

}

public EarlybirdLuceneQueryVisitor setProximityPhraseWeight(float weight) {

this.proximityPhraseWeight = weight;

return this;

}

public EarlybirdLuceneQueryVisitor setProximityPhraseSlop(int slop) {

this.proximityPhraseSlop = slop;

return this;

}

public void setFieldHitAttributeHelper(HitAttributeHelper newHitAttributeHelper) {

this.hitAttributeHelper = newHitAttributeHelper;

}

@Override

public final Query visit(Disjunction disjunction) throws QueryParserException {

BooleanQuery.Builder bqBuilder = new BooleanQuery.Builder();

List<com.twitter.search.queryparser.query.Query> children = disjunction.getChildren();

// Do a final round of check, if all nodes under a disjunction are MUST,

// treat them all as DEFAULT (SHOULD in Lucene).

boolean allMust = true;

for (com.twitter.search.queryparser.query.Query child : children) {

if (!child.mustOccur()) {

allMust = false;

break;

}

}

if (allMust) {

children = Lists.transform(children, QueryNodeUtils.MAKE\_QUERY\_DEFAULT);

}

// Actually converting all children now.

for (com.twitter.search.queryparser.query.Query child : children) {

final Query q = child.accept(this);

if (q != null) {

// if a node is marked with MUSTHAVE annotation, we set it to must even if it's a

// disjunction.

if (child.mustOccur()) {

bqBuilder.add(q, Occur.MUST);

} else {

bqBuilder.add(q, Occur.SHOULD);

}

}

}

Query bq = bqBuilder.build();

float boost = (float) getBoostFromAnnotations(disjunction.getAnnotations());

if (boost >= 0) {

bq = BoostUtils.maybeWrapInBoostQuery(bq, boost);

}

return bq;

}

@Override

public Query visit(Conjunction conjunction) throws QueryParserException {

BooleanQuery.Builder bqBuilder = new BooleanQuery.Builder();

List<com.twitter.search.queryparser.query.Query> children = conjunction.getChildren();

boolean hasPositiveTerms = false;

for (com.twitter.search.queryparser.query.Query child : children) {

boolean childMustNotOccur = child.mustNotOccur();

boolean childAdded = addQuery(bqBuilder, child);

if (childAdded && !childMustNotOccur) {

hasPositiveTerms = true;

}

}

if (!children.isEmpty() && !hasPositiveTerms) {

bqBuilder.add(new MatchAllDocsQuery(), Occur.MUST);

}

Query bq = bqBuilder.build();

float boost = (float) getBoostFromAnnotations(conjunction.getAnnotations());

if (boost >= 0) {

bq = BoostUtils.maybeWrapInBoostQuery(bq, boost);

}

return bq;

}

@Override

public Query visit(Phrase phrase) throws QueryParserException {

return visit(phrase, false);

}

@Override

public Query visit(Term term) throws QueryParserException {

return finalizeQuery(createTermQueryDisjunction(term), term);

}

@Override

public Query visit(SpecialTerm special) throws QueryParserException {

String field;

switch (special.getType()) {

case HASHTAG:

field = EarlybirdFieldConstant.HASHTAGS\_FIELD.getFieldName();

break;

case STOCK:

field = EarlybirdFieldConstant.STOCKS\_FIELD.getFieldName();

break;

case MENTION:

field = EarlybirdFieldConstant.MENTIONS\_FIELD.getFieldName();

break;

default:

field = EarlybirdFieldConstant.TEXT\_FIELD.getFieldName();

}

String termText = special.getSpecialChar() + special.getValue();

Query q = createSimpleTermQuery(special, field, termText);

float boost = (float) getBoostFromAnnotations(special.getAnnotations());

if (boost >= 0) {

q = BoostUtils.maybeWrapInBoostQuery(q, boost);

}

return negateQueryIfNodeNegated(special, q);

}

@Override

public Query visit(Link link) throws QueryParserException {

Query q = createSimpleTermQuery(

link, EarlybirdFieldConstant.LINKS\_FIELD.getFieldName(), link.getOperand());

float boost = (float) getBoostFromAnnotations(link.getAnnotations());

if (boost >= 0) {

q = BoostUtils.maybeWrapInBoostQuery(q, boost);

}

return negateQueryIfNodeNegated(link, q);

}

@Override

public Query visit(final SearchOperator op) throws QueryParserException {

final Query query;

SearchOperator.Type type = op.getOperatorType();

switch (type) {

case TO:

query = visitToOperator(op);

break;

case FROM:

query = visitFromOperator(op);

break;

case FILTER:

query = visitFilterOperator(op);

break;

case INCLUDE:

query = visitIncludeOperator(op);

break;

case EXCLUDE:

query = visitExcludeOperator(op);

break;

case LANG:

query = visitLangOperator(op);

break;

case SOURCE:

query = visitSourceOperator(op);

break;

case SMILEY:

query = visitSmileyOperator(op);

break;

case DOCVAL\_RANGE\_FILTER:

query = visitDocValRangeFilterOperator(op);

break;

case CACHED\_FILTER:

query = visitCachedFilterOperator(op);

break;

case SCORE\_FILTER:

query = visitScoredFilterOperator(op);

break;

case SINCE\_TIME:

query = visitSinceTimeOperator(op);

break;

case UNTIL\_TIME:

query = visitUntilTimeOperator(op);

break;

case SINCE\_ID:

query = visitSinceIDOperator(op);

break;

case MAX\_ID:

query = visitMaxIDOperator(op);

break;

case GEOLOCATION\_TYPE:

query = visitGeoLocationTypeOperator(op);

break;

case GEOCODE:

query = visitGeocodeOperator(op);

break;

case GEO\_BOUNDING\_BOX:

query = visitGeoBoundingBoxOperator(op);

break;

case PLACE:

query = visitPlaceOperator(op);

break;

case LINK:

// This should never be called - the Link visitor (visitor(Link link)) should be.

query = visitLinkOperator(op);

break;

case ENTITY\_ID:

query = visitEntityIdOperator(op);

break;

case FROM\_USER\_ID:

query = visitFromUserIDOperator(op);

break;

case IN\_REPLY\_TO\_TWEET\_ID:

query = visitInReplyToTweetIdOperator(op);

break;

case IN\_REPLY\_TO\_USER\_ID:

query = visitInReplyToUserIdOperator(op);

break;

case LIKED\_BY\_USER\_ID:

query = visitLikedByUserIdOperator(op);

break;

case RETWEETED\_BY\_USER\_ID:

query = visitRetweetedByUserIdOperator(op);

break;

case REPLIED\_TO\_BY\_USER\_ID:

query = visitRepliedToByUserIdOperator(op);

break;

case QUOTED\_USER\_ID:

query = visitQuotedUserIdOperator(op);

break;

case QUOTED\_TWEET\_ID:

query = visitQuotedTweetIdOperator(op);

break;

case DIRECTED\_AT\_USER\_ID:

query = visitDirectedAtUserIdOperator(op);

break;

case CONVERSATION\_ID:

query = visitConversationIdOperator(op);

break;

case COMPOSER\_SOURCE:

query = visitComposerSourceOperator(op);

break;

case RETWEETS\_OF\_TWEET\_ID:

query = visitRetweetsOfTweetIdOperator(op);

break;

case RETWEETS\_OF\_USER\_ID:

query = visitRetweetsOfUserIdOperator(op);

break;

case LINK\_CATEGORY:

query = visitLinkCategoryOperator(op);

break;

case CARD\_NAME:

query = visitCardNameOperator(op);

break;

case CARD\_DOMAIN:

query = visitCardDomainOperator(op);

break;

case CARD\_LANG:

query = visitCardLangOperator(op);

break;

case HF\_TERM\_PAIR:

query = visitHFTermPairOperator(op);

break;

case HF\_PHRASE\_PAIR:

query = visitHFTermPhrasePairOperator(op);

break;

case PROXIMITY\_GROUP:

Phrase phrase = new Phrase(

Lists.transform(op.getOperands(),

s -> NormalizerHelper.normalizeWithUnknownLocale(

s, EarlybirdConfig.getPenguinVersion())));

query = visit(phrase, true);

break;

case MULTI\_TERM\_DISJUNCTION:

query = visitMultiTermDisjunction(op);

break;

case CSF\_DISJUNCTION\_FILTER:

query = visitCSFDisjunctionFilter(op);

break;

case SAFETY\_EXCLUDE:

query = visitSafetyExclude(op);

break;

case SPACE\_ID:

query = visitSpaceId(op);

break;

case NAMED\_ENTITY:

query = visitNamedEntity(op);

break;

case NAMED\_ENTITY\_WITH\_TYPE:

query = visitNamedEntityWithType(op);

break;

case MIN\_FAVES:

case MIN\_QUALITY\_SCORE:

case MIN\_REPLIES:

case MIN\_RETWEETS:

case MIN\_REPUTATION:

query = visitMinFeatureValueOperator(type, op);

break;

case FEATURE\_VALUE\_IN\_ACCEPT\_LIST\_OR\_UNSET:

query = visitFeatureValueInAcceptListOrUnsetFilterOperator(op);

break;

case NEAR:

case RELATED\_TO\_TWEET\_ID:

case SINCE:

case SITE:

case UNTIL:

case WITHIN:

case WITHIN\_TIME:

query = createUnsupportedOperatorQuery(op);

break;

case NAMED\_CSF\_DISJUNCTION\_FILTER:

case NAMED\_MULTI\_TERM\_DISJUNCTION:

query = logAndThrowQueryParserException(

"Named disjunction operator could not be converted to a disjunction operator.");

break;

default:

query = logAndThrowQueryParserException("Unknown operator " + op.toString());

}

return negateQueryIfNodeNegated(op, query);

}

protected Query visitToOperator(SearchOperator op) throws QueryParserException {

return createNormalizedTermQuery(

op, EarlybirdFieldConstant.TO\_USER\_FIELD.getFieldName(), op.getOperand());

}

protected Query visitFromOperator(SearchOperator op) throws QueryParserException {

return createNormalizedTermQuery(

op, EarlybirdFieldConstant.FROM\_USER\_FIELD.getFieldName(), op.getOperand());

}

protected Query visitFilterOperator(SearchOperator op) throws QueryParserException {

return visitFilterOperator(op, false);

}

protected Query visitIncludeOperator(SearchOperator op) throws QueryParserException {

// Include is a bit funny. If we have [include retweets] we are saying

// do include retweets, which is the default. Also conjunctions re-negate

// whatever node we emit from the visitor.

if (!isParentNegated(op) && !nodeIsNegated(op)) {

// positive include - no-op.

return null;

}

return visitFilterOperator(op, false);

}

protected Query visitExcludeOperator(SearchOperator op) throws QueryParserException {

// Exclude is a bit funny. If we have -[exclude retweets] we are saying

// dont exclude retweets, which is the default.

if (isParentNegated(op) || nodeIsNegated(op)) {

// Negative exclude. Do nothing - parent will not add this to the list of children.

return null;

} else {

// Positive exclude.

return visitFilterOperator(op, true);

}

}

protected Query visitFilterOperator(SearchOperator op, boolean negate)

throws QueryParserException {

Query q;

boolean negateQuery = negate;

if (op.getOperand().equals(SearchOperatorConstants.ANTISOCIAL)) {

// Since the object we use to implement these filters is actually an

// EXCLUDE filter, we need to negate it to get it to work as a regular filter.

q = UserFlagsExcludeFilter.getUserFlagsExcludeFilter(userTable, true, false, false);

negateQuery = !negateQuery;

} else if (op.getOperand().equals(SearchOperatorConstants.OFFENSIVE\_USER)) {

q = UserFlagsExcludeFilter.getUserFlagsExcludeFilter(userTable, false, true, false);

negateQuery = !negateQuery;

} else if (op.getOperand().equals(SearchOperatorConstants.ANTISOCIAL\_OFFENSIVE\_USER)) {

q = UserFlagsExcludeFilter.getUserFlagsExcludeFilter(userTable, true, true, false);

negateQuery = !negateQuery;

} else if (op.getOperand().equals(SearchOperatorConstants.PROTECTED)) {

q = UserFlagsExcludeFilter.getUserFlagsExcludeFilter(userTable, false, false, true);

negateQuery = !negateQuery;

} else if (op.getOperand().equals(SearchOperatorConstants.HAS\_ENGAGEMENT)) {

return buildHasEngagementsQuery();

} else if (op.getOperand().equals(SearchOperatorConstants.SAFE\_SEARCH\_FILTER)) {

BooleanQuery.Builder bqBuilder = new BooleanQuery.Builder();

bqBuilder.add(

createNoScoreTermQuery(

op,

EarlybirdFieldConstant.INTERNAL\_FIELD.getFieldName(),

EarlybirdFieldConstant.IS\_OFFENSIVE),

Occur.SHOULD);

// The following internal field \_\_filter\_sensitive\_content

// is not currently built by earlybird.

// This means the safe search filter soley operates on the is\_offensive bit

bqBuilder.add(

createNoScoreTermQuery(

op,

EarlybirdFieldConstant.INTERNAL\_FIELD.getFieldName(),

EarlybirdThriftDocumentUtil.formatFilter(SearchOperatorConstants.SENSITIVE\_CONTENT)),

Occur.SHOULD);

q = bqBuilder.build();

negateQuery = !negateQuery;

} else if (op.getOperand().equals(SearchOperatorConstants.RETWEETS)) {

// Special case for filter:retweets - we use the text field search "-rt"

// mostly for legacy reasons.

q = createSimpleTermQuery(

op,

EarlybirdFieldConstant.TEXT\_FIELD.getFieldName(),

EarlybirdThriftDocumentBuilder.RETWEET\_TERM);

} else if (schemaSnapshot.getFacetFieldByFacetName(op.getOperand()) != null) {

Schema.FieldInfo facetField = schemaSnapshot.getFacetFieldByFacetName(op.getOperand());

if (facetField.getFieldType().isStoreFacetSkiplist()) {

q = createSimpleTermQuery(

op,

EarlybirdFieldConstant.INTERNAL\_FIELD.getFieldName(),

EarlybirdFieldConstant.getFacetSkipFieldName(facetField.getName()));

} else {

// return empty BQ that doesn't match anything

q = new BooleanQuery.Builder().build();

}

} else if (op.getOperand().equals(SearchOperatorConstants.VINE\_LINK)) {

// Temporary special case for filter:vine\_link. The filter is called "vine\_link", but it

// should use the internal field "\_\_filter\_vine". We need this special case because otherwise

// it would look for the non-existing "\_\_filter\_vine\_link" field. See SEARCH-9390

q = createNoScoreTermQuery(

op,

EarlybirdFieldConstant.INTERNAL\_FIELD.getFieldName(),

EarlybirdThriftDocumentUtil.formatFilter("vine"));

} else {

// The default vanilla filters just uses the filter format string and the

// operand text.

q = createNoScoreTermQuery(

op,

EarlybirdFieldConstant.INTERNAL\_FIELD.getFieldName(),

EarlybirdThriftDocumentUtil.formatFilter(op.getOperand()));

}

// Double check: no filters should have any score contribution.

q = new BoostQuery(q, 0.0f);

return negateQuery ? negateQuery(q) : q;

}

private Query buildHasEngagementsQuery() {

if (earlybirdCluster == EarlybirdCluster.PROTECTED) {

// Engagements and engagement counts are not indexed on Earlybirds, so there is no need to

// traverse the entire segment with the MinFeatureValueFilter. See SEARCH-28120

return new MatchNoDocsQuery();

}

Query favFilter = MinFeatureValueFilter.getMinFeatureValueFilter(

EarlybirdFieldConstant.FAVORITE\_COUNT.getFieldName(), 1);

Query retweetFilter = MinFeatureValueFilter.getMinFeatureValueFilter(

EarlybirdFieldConstant.RETWEET\_COUNT.getFieldName(), 1);

Query replyFilter = MinFeatureValueFilter.getMinFeatureValueFilter(

EarlybirdFieldConstant.REPLY\_COUNT.getFieldName(), 1);

return new BooleanQuery.Builder()

.add(favFilter, Occur.SHOULD)

.add(retweetFilter, Occur.SHOULD)

.add(replyFilter, Occur.SHOULD)

.build();

}

protected Query visitLangOperator(SearchOperator op) throws QueryParserException {

return createNoScoreTermQuery(

op, EarlybirdFieldConstant.ISO\_LANGUAGE\_FIELD.getFieldName(), op.getOperand());

}

protected Query visitSourceOperator(SearchOperator op) throws QueryParserException {

return createNoScoreTermQuery(

op, EarlybirdFieldConstant.NORMALIZED\_SOURCE\_FIELD.getFieldName(), op.getOperand());

}

protected Query visitSmileyOperator(SearchOperator op) throws QueryParserException {

return createSimpleTermQuery(

op,

EarlybirdFieldConstant.INTERNAL\_FIELD.getFieldName(),

String.format(SMILEY\_FORMAT\_STRING, op.getOperand()));

}

protected Query visitDocValRangeFilterOperator(SearchOperator op) throws QueryParserException {

String csfFieldName = op.getOperands().get(0).toLowerCase();

ThriftCSFType csfFieldType = schemaSnapshot.getCSFFieldType(csfFieldName);

if (csfFieldType == null) {

throw new QueryParserException("invalid csf field name " + op.getOperands().get(0)

+ " used in " + op.serialize());

}

try {

if (csfFieldType == ThriftCSFType.DOUBLE

|| csfFieldType == ThriftCSFType.FLOAT) {

return DocValRangeFilter.getDocValRangeQuery(csfFieldName, csfFieldType,

Double.parseDouble(op.getOperands().get(1)),

Double.parseDouble(op.getOperands().get(2)));

} else if (csfFieldType == ThriftCSFType.LONG

|| csfFieldType == ThriftCSFType.INT

|| csfFieldType == ThriftCSFType.BYTE) {

Query query = DocValRangeFilter.getDocValRangeQuery(csfFieldName, csfFieldType,

Long.parseLong(op.getOperands().get(1)),

Long.parseLong(op.getOperands().get(2)));

if (csfFieldName.equals(EarlybirdFieldConstant.LAT\_LON\_CSF\_FIELD.getFieldName())) {

return wrapQueryInUserScrubGeoFilter(query);

}

return query;

} else {

throw new QueryParserException("invalid ThriftCSFType. drop this op: " + op.serialize());

}

} catch (NumberFormatException e) {

throw new QueryParserException("invalid range numeric type used in " + op.serialize());

}

}

protected final Query visitCachedFilterOperator(SearchOperator op) throws QueryParserException {

try {

return CachedFilterQuery.getCachedFilterQuery(op.getOperand(), queryCacheManager);

} catch (CachedFilterQuery.NoSuchFilterException e) {

throw new QueryParserException(e.getMessage(), e);

}

}

protected final Query visitScoredFilterOperator(SearchOperator op) throws QueryParserException {

final List<String> operands = op.getOperands();

final String scoreFunction = operands.get(0);

ScoringFunctionProvider.NamedScoringFunctionProvider scoringFunctionProvider =

ScoringFunctionProvider.getScoringFunctionProviderByName(scoreFunction, schemaSnapshot);

if (scoringFunctionProvider == null) {

throw new QueryParserException("Unknown scoring function name [" + scoreFunction

+ " ] used as score\_filter's operand");

}

return ScoreFilterQuery.getScoreFilterQuery(

schemaSnapshot,

scoringFunctionProvider,

Float.parseFloat(operands.get(1)),

Float.parseFloat(operands.get(2)));

}

protected Query visitSinceTimeOperator(SearchOperator op) {

try {

return SinceUntilFilter.getSinceQuery(Integer.parseInt(op.getOperand()));

} catch (NumberFormatException e) {

LOG.warn("since time is not a valid integer, the date isn't reasonable. drop this op: "

+ op.serialize());

SINCE\_TIME\_INVALID\_INT\_COUNTER.increment();

return null;

}

}

protected Query visitUntilTimeOperator(SearchOperator op) {

try {

return SinceUntilFilter.getUntilQuery(Integer.parseInt(op.getOperand()));

} catch (NumberFormatException e) {

LOG.warn("until time is not a valid integer, the date isn't reasonable. drop this op: "

+ op.serialize());

UNTIL\_TIME\_INVALID\_INT\_COUNTER.increment();

return null;

}

}

protected Query visitSinceIDOperator(SearchOperator op) {

long id = Long.parseLong(op.getOperand());

return SinceMaxIDFilter.getSinceIDQuery(id);

}

protected Query visitMaxIDOperator(SearchOperator op) {

long id = Long.parseLong(op.getOperand());

return SinceMaxIDFilter.getMaxIDQuery(id);

}

protected Query visitGeoLocationTypeOperator(SearchOperator op) throws QueryParserException {

String operand = op.getOperand();

ThriftGeoLocationSource source = ThriftGeoLocationSource.valueOf(operand.toUpperCase());

// If necessary, this query will be wrapped by the UserScrubGeoFilter within

// the createSimpleTermQuery() helper method

return createNoScoreTermQuery(

op,

EarlybirdFieldConstant.INTERNAL\_FIELD.getFieldName(),

EarlybirdFieldConstants.formatGeoType(source));

}

protected Query visitGeocodeOperator(SearchOperator op) throws QueryParserException {

return visitGeocodeOrGeocodePrivateOperator(op);

}

protected Query visitGeoBoundingBoxOperator(SearchOperator op) throws QueryParserException {

Rectangle rectangle = boundingBoxFromSearchOperator(op);

return wrapQueryInUserScrubGeoFilter(

GeoQuadTreeQueryBuilder.buildGeoQuadTreeQuery(rectangle, terminationTracker));

}

protected Query visitPlaceOperator(SearchOperator op) throws QueryParserException {

// This query will be wrapped by the UserScrubGeoFilter within the createSimpleTermQuery()

// helper method

return createSimpleTermQuery(

op, EarlybirdFieldConstant.PLACE\_FIELD.getFieldName(), op.getOperand());

}

protected Query visitLinkOperator(SearchOperator op) throws QueryParserException {

// This should never be called - the Link visitor (visitor(Link link)) should be.

if (op instanceof Link) {

LOG.warn("Unexpected Link operator " + op.serialize());

return visit((Link) op);

} else {

throw new QueryParserException("Operator type set to " + op.getOperatorName()

+ " but it is not an instance of Link [" + op.toString() + "]");

}

}

protected Query visitEntityIdOperator(SearchOperator op) throws QueryParserException {

return createSimpleTermQuery(

op, EarlybirdFieldConstant.ENTITY\_ID\_FIELD.getFieldName(), op.getOperand());

}

protected Query visitFromUserIDOperator(SearchOperator op) {

return buildLongTermAttributeQuery(

op, EarlybirdFieldConstant.FROM\_USER\_ID\_FIELD.getFieldName());

}

protected Query visitInReplyToTweetIdOperator(SearchOperator op) {

return buildLongTermAttributeQuery(

op, EarlybirdFieldConstant.IN\_REPLY\_TO\_TWEET\_ID\_FIELD.getFieldName());

}

protected Query visitInReplyToUserIdOperator(SearchOperator op) {

return buildLongTermAttributeQuery(

op, EarlybirdFieldConstant.IN\_REPLY\_TO\_USER\_ID\_FIELD.getFieldName());

}

protected Query visitLikedByUserIdOperator(SearchOperator op) throws QueryParserException {

return buildLongTermAttributeQuery(op,

EarlybirdFieldConstant.LIKED\_BY\_USER\_ID\_FIELD.getFieldName());

}

protected Query visitRetweetedByUserIdOperator(SearchOperator op) throws QueryParserException {

return buildLongTermAttributeQuery(op,

EarlybirdFieldConstant.RETWEETED\_BY\_USER\_ID.getFieldName());

}

protected Query visitRepliedToByUserIdOperator(SearchOperator op) throws QueryParserException {

return buildLongTermAttributeQuery(op,

EarlybirdFieldConstant.REPLIED\_TO\_BY\_USER\_ID.getFieldName());

}

protected Query visitQuotedUserIdOperator(SearchOperator op) throws QueryParserException {

return buildLongTermAttributeQuery(op,

EarlybirdFieldConstant.QUOTED\_USER\_ID\_FIELD.getFieldName());

}

protected Query visitQuotedTweetIdOperator(SearchOperator op) throws QueryParserException {

return buildLongTermAttributeQuery(op,

EarlybirdFieldConstant.QUOTED\_TWEET\_ID\_FIELD.getFieldName());

}

protected Query visitDirectedAtUserIdOperator(SearchOperator op) throws QueryParserException {

return buildLongTermAttributeQuery(op,

EarlybirdFieldConstant.DIRECTED\_AT\_USER\_ID\_FIELD.getFieldName());

}

protected Query visitConversationIdOperator(SearchOperator op) throws QueryParserException {

return buildLongTermAttributeQuery(

op, EarlybirdFieldConstant.CONVERSATION\_ID\_FIELD.getFieldName());

}

protected Query visitComposerSourceOperator(SearchOperator op) throws QueryParserException {

Preconditions.checkNotNull(op.getOperand(), "composer\_source requires operand");

try {

ComposerSource composerSource = ComposerSource.valueOf(op.getOperand().toUpperCase());

return buildNoScoreIntTermQuery(

op, EarlybirdFieldConstant.COMPOSER\_SOURCE, composerSource.getValue());

} catch (IllegalArgumentException e) {

throw new QueryParserException("Invalid operand for composer\_source: " + op.getOperand(), e);

}

}

protected Query visitRetweetsOfTweetIdOperator(SearchOperator op) {

return buildLongTermAttributeQuery(

op, EarlybirdFieldConstant.RETWEET\_SOURCE\_TWEET\_ID\_FIELD.getFieldName());

}

protected Query visitRetweetsOfUserIdOperator(SearchOperator op) {

return buildLongTermAttributeQuery(

op, EarlybirdFieldConstant.RETWEET\_SOURCE\_USER\_ID\_FIELD.getFieldName());

}

protected Query visitLinkCategoryOperator(SearchOperator op) {

int linkCategory;

try {

linkCategory = LinkCategory.valueOf(op.getOperand()).getValue();

} catch (IllegalArgumentException e) {

linkCategory = Integer.parseInt(op.getOperand());

}

String fieldName = EarlybirdFieldConstant.LINK\_CATEGORY\_FIELD.getFieldName();

org.apache.lucene.index.Term term = new org.apache.lucene.index.Term(

fieldName, IntTermAttributeImpl.copyIntoNewBytesRef(linkCategory));

return wrapQuery(

new TermQueryWithSafeToString(term, Integer.toString(linkCategory)), op, fieldName);

}

protected Query visitCardNameOperator(SearchOperator op) throws QueryParserException {

return createNoScoreTermQuery(

op, EarlybirdFieldConstant.CARD\_NAME\_FIELD.getFieldName(), op.getOperand());

}

protected Query visitCardDomainOperator(SearchOperator op) throws QueryParserException {

return createNoScoreTermQuery(

op, EarlybirdFieldConstant.CARD\_DOMAIN\_FIELD.getFieldName(), op.getOperand());

}

protected Query visitCardLangOperator(SearchOperator op) throws QueryParserException {

return createNoScoreTermQuery(

op, EarlybirdFieldConstant.CARD\_LANG.getFieldName(), op.getOperand());

}

protected Query visitHFTermPairOperator(SearchOperator op) throws QueryParserException {

final List<String> operands = op.getOperands();

String termPair = HighFrequencyTermPairs.createPair(op.getOperands().get(0),

op.getOperands().get(1));

Query q = createSimpleTermQuery(op, ImmutableSchema.HF\_TERM\_PAIRS\_FIELD, termPair);

float boost = Float.parseFloat(operands.get(2));

if (boost >= 0) {

q = BoostUtils.maybeWrapInBoostQuery(q, boost);

}

return q;

}

protected Query visitHFTermPhrasePairOperator(SearchOperator op) throws QueryParserException {

final List<String> operands = op.getOperands();

String termPair = HighFrequencyTermPairs.createPhrasePair(op.getOperands().get(0),

op.getOperands().get(1));

Query q = createSimpleTermQuery(op, ImmutableSchema.HF\_PHRASE\_PAIRS\_FIELD, termPair);

float boost = Float.parseFloat(operands.get(2));

if (boost >= 0) {

q = BoostUtils.maybeWrapInBoostQuery(q, boost);

}

return q;

}

private Query logAndThrowQueryParserException(String message) throws QueryParserException {

LOG.error(message);

throw new QueryParserException(message);

}

private Query logMissingEntriesAndThrowQueryParserException(String field, SearchOperator op)

throws QueryParserException {

return logAndThrowQueryParserException(

String.format("Missing required %s entries for %s", field, op.serialize()));

}

// previous implementation of this operator allowed insertion of

// operands from the thrift search query. This was reverted to ensure simplicity

// of the api, and to keep the serialized query self contained.

protected final Query visitMultiTermDisjunction(SearchOperator op) throws QueryParserException {

final List<String> operands = op.getOperands();

final String field = operands.get(0);

if (isUserIdField(field)) {

List<Long> ids = Lists.newArrayList();

parseLongArgs(operands.subList(1, operands.size()), ids, op);

if (ids.size() > 0) {

// Try to get ranks for ids if exist from hitAttributeHelper.

// Otherwise just pass in a empty list.

List<Integer> ranks;

if (hitAttributeHelper != null

&& hitAttributeHelper.getExpandedNodeToRankMap().containsKey(op)) {

ranks = hitAttributeHelper.getExpandedNodeToRankMap().get(op);

} else {

ranks = Lists.newArrayList();

}

return UserIdMultiSegmentQuery.createIdDisjunctionQuery(

"multi\_term\_disjunction\_" + field,

ids,

field,

schemaSnapshot,

multiSegmentTermDictionaryManager,

decider,

earlybirdCluster,

ranks,

hitAttributeHelper,

queryTimeout);

} else {

return logMissingEntriesAndThrowQueryParserException(field, op);

}

} else if (EarlybirdFieldConstant.ID\_FIELD.getFieldName().equals(field)) {

List<Long> ids = Lists.newArrayList();

parseLongArgs(operands.subList(1, operands.size()), ids, op);

if (ids.size() > 0) {

return RequiredStatusIDsFilter.getRequiredStatusIDsQuery(ids);

} else {

return logMissingEntriesAndThrowQueryParserException(field, op);

}

} else if (isTweetIdField(field)) {

List<Long> ids = Lists.newArrayList();

parseLongArgs(operands.subList(1, operands.size()), ids, op);

if (ids.size() > 0) {

BooleanQuery.Builder bqBuilder = new BooleanQuery.Builder();

int numClauses = 0;

for (long id : ids) {

if (numClauses >= BooleanQuery.getMaxClauseCount()) {

BooleanQuery saved = bqBuilder.build();

bqBuilder = new BooleanQuery.Builder();

bqBuilder.add(saved, BooleanClause.Occur.SHOULD);

numClauses = 1;

}

bqBuilder.add(buildLongTermAttributeQuery(op, field, id), Occur.SHOULD);

++numClauses;

}

return bqBuilder.build();

} else {

return logMissingEntriesAndThrowQueryParserException(field, op);

}

} else {

return createUnsupportedOperatorQuery(op);

}

}

protected final Query visitCSFDisjunctionFilter(SearchOperator op)

throws QueryParserException {

List<String> operands = op.getOperands();

String field = operands.get(0);

ThriftCSFType csfType = schemaSnapshot.getCSFFieldType(field);

if (csfType == null) {

throw new QueryParserException("Field must be a CSF");

}

if (csfType != ThriftCSFType.LONG) {

throw new QueryParserException("csf\_disjunction\_filter only works with long fields");

}

Set<Long> values = new HashSet<>();

parseLongArgs(operands.subList(1, operands.size()), values, op);

Query query = CSFDisjunctionFilter.getCSFDisjunctionFilter(field, values);

if (field.equals(EarlybirdFieldConstant.LAT\_LON\_CSF\_FIELD.getFieldName())) {

return wrapQueryInUserScrubGeoFilter(query);

}

return query;

}

protected Query visitSafetyExclude(SearchOperator op) throws QueryParserException {

// We do not allow negating safety\_exclude operator. Note the operator is internal so if we

// get here, it means there's a bug in the query construction side.

if (isParentNegated(op) || nodeIsNegated(op)) {

throw new QueryParserException("Negating safety\_exclude operator is not allowed: " + op);

}

// Convert the safety filter to other operators depending on cluster setting

// The safety filter is interpreted differently on archive because the underlying safety labels

// in extended encoded field are not available on archive.

if (EarlybirdCluster.isArchive(earlybirdCluster)) {

return visit(OPERATOR\_CACHED\_EXCLUDE\_ANTISOCIAL\_AND\_NATIVERETWEETS);

} else {

List<com.twitter.search.queryparser.query.Query> children = Lists.newArrayList();

for (String filterName : op.getOperands()) {

children.addAll(

OPERATORS\_BY\_SAFE\_EXCLUDE\_OPERAND.getOrDefault(filterName, ImmutableList.of()));

}

return visit(new Conjunction(children));

}

}

protected Query visitNamedEntity(SearchOperator op) throws QueryParserException {

List<String> operands = op.getOperands();

Preconditions.checkState(operands.size() == 1,

"named\_entity: wrong number of operands");

return createDisjunction(

operands.get(0).toLowerCase(),

op,

EarlybirdFieldConstant.NAMED\_ENTITY\_FROM\_TEXT\_FIELD,

EarlybirdFieldConstant.NAMED\_ENTITY\_FROM\_URL\_FIELD);

}

protected Query visitSpaceId(SearchOperator op) throws QueryParserException {

List<String> operands = op.getOperands();

Preconditions.checkState(operands.size() == 1,

"space\_id: wrong number of operands");

return createSimpleTermQuery(

op,

EarlybirdFieldConstant.SPACE\_ID\_FIELD.getFieldName(),

op.getOperand()

);

}

protected Query visitNamedEntityWithType(SearchOperator op) throws QueryParserException {

List<String> operands = op.getOperands();

Preconditions.checkState(operands.size() == 2,

"named\_entity\_with\_type: wrong number of operands");

String name = operands.get(0);

String type = operands.get(1);

return createDisjunction(

String.format("%s:%s", name, type).toLowerCase(),

op,

EarlybirdFieldConstant.NAMED\_ENTITY\_WITH\_TYPE\_FROM\_TEXT\_FIELD,

EarlybirdFieldConstant.NAMED\_ENTITY\_WITH\_TYPE\_FROM\_URL\_FIELD);

}

// Create a disjunction query for a given value in one of the given fields

private Query createDisjunction(

String value, SearchOperator operator, EarlybirdFieldConstant... fields)

throws QueryParserException {

BooleanQuery.Builder booleanQueryBuilder = new BooleanQuery.Builder();

for (EarlybirdFieldConstant field : fields) {

booleanQueryBuilder.add(

createSimpleTermQuery(operator, field.getFieldName(), value), Occur.SHOULD);

}

return booleanQueryBuilder.build();

}

protected Query visitMinFeatureValueOperator(SearchOperator.Type type, SearchOperator op) {

final List<String> operands = op.getOperands();

String featureName;

switch (type) {

case MIN\_FAVES:

featureName = EarlybirdFieldConstant.FAVORITE\_COUNT.getFieldName();

break;

case MIN\_QUALITY\_SCORE:

featureName = EarlybirdFieldConstant.PARUS\_SCORE.getFieldName();

break;

case MIN\_REPLIES:

featureName = EarlybirdFieldConstant.REPLY\_COUNT.getFieldName();

break;

case MIN\_REPUTATION:

featureName = EarlybirdFieldConstant.USER\_REPUTATION.getFieldName();

break;

case MIN\_RETWEETS:

featureName = EarlybirdFieldConstant.RETWEET\_COUNT.getFieldName();

break;

default:

throw new IllegalArgumentException("Unknown min feature type " + type);

}

double operand = Double.parseDouble(operands.get(0));

// SEARCH-16751: Because we use QueryCacheConstants.HAS\_ENGAGEMENT as a driving query below, we

// won't return tweets with 0 engagements when we handle a query with a [min\_X 0] filter (e.g.

// (\* cat [min\_faves 0] ). Thus we need to return a MatchAllDocsQuery in that case.

if (operand == 0) {

return new MatchAllDocsQuery();

}

// Only perform the rewrite if the operator is a min engagement operator.

if (isOperatorTypeEngagementFilter(type)) {

return buildQueryForEngagementOperator(op, operands, featureName);

}

if (type == SearchOperator.Type.MIN\_REPUTATION) {

return buildQueryForMinReputationOperator(operands, featureName);

}

return MinFeatureValueFilter.getMinFeatureValueFilter(

featureName, Double.parseDouble(operands.get(0)));

}

protected Query visitFeatureValueInAcceptListOrUnsetFilterOperator(SearchOperator op)

throws QueryParserException {

final List<String> operands = op.getOperands();

final String field = operands.get(0);

if (isIdCSFField(field)) {

Set<Long> ids = Sets.newHashSet();

parseLongArgs(operands.subList(1, operands.size()), ids, op);

return FeatureValueInAcceptListOrUnsetFilter.getFeatureValueInAcceptListOrUnsetFilter(

field, ids);

} else {

return logAndThrowQueryParserException(

"Invalid CSF field passed to operator " + op.toString());

}

}

/\*\*

\* Creates a Lucene query for an operator that's not supported by the search service.

\*

\* NOTE: Developer, if you are writing a class to extends this class, make sure the

\* behaviour of this function makes sense for your search service.

\*

\* @param op The operator that's not supported by the search service.

\* @return The Lucene query for this operator

\*/

protected Query createUnsupportedOperatorQuery(SearchOperator op) throws QueryParserException {

SearchCounter

.export(UNSUPPORTED\_OPERATOR\_PREFIX + op.getOperatorType().getOperatorName())

.increment();

return visit(op.toPhrase());

}

private Query buildNoScoreIntTermQuery(

SearchOperator op,

EarlybirdFieldConstant field,

int termValue) {

org.apache.lucene.index.Term term = new org.apache.lucene.index.Term(

field.getFieldName(), IntTermAttributeImpl.copyIntoNewBytesRef(termValue));

return wrapQuery(

new TermQueryWithSafeToString(term, Integer.toString(termValue)), op, field.getFieldName());

}

private Query buildQueryForMinReputationOperator(List<String> operands, String featureName) {

int operand = (int) Double.parseDouble(operands.get(0));

// Driving by MinFeatureValueFilter's DocIdSetIterator is very slow, because we have to

// perform an expensive check for all doc IDs in the segment, so we use a cached result to

// drive the query, and use MinFeatureValueFilter as a secondary filter.

String queryCacheFilterName;

if (operand >= 50) {

queryCacheFilterName = QueryCacheConstants.MIN\_REPUTATION\_50;

} else if (operand >= 36) {

queryCacheFilterName = QueryCacheConstants.MIN\_REPUTATION\_36;

} else if (operand >= 30) {

queryCacheFilterName = QueryCacheConstants.MIN\_REPUTATION\_30;

} else {

return MinFeatureValueFilter.getMinFeatureValueFilter(featureName, operand);

}

try {

Query drivingQuery = CachedFilterQuery.getCachedFilterQuery(

queryCacheFilterName, queryCacheManager);

return new FilteredQuery(

drivingQuery, MinFeatureValueFilter.getDocIdFilterFactory(featureName, operand));

} catch (Exception e) {

// If the filter is not found, that's OK, it might be our first time running the query cache,

// or there may be no tweets with that high reputation.

return MinFeatureValueFilter.getMinFeatureValueFilter(featureName, operand);

}

}

private Query buildQueryForEngagementOperator(

SearchOperator op, List<String> operands, String featureName) {

// Engagements and engagement counts are not indexed on Protected Earlybirds, so there is no

// need to traverse the entire segment with the MinFeatureValueFilter. SEARCH-28120

if (earlybirdCluster == EarlybirdCluster.PROTECTED) {

return new MatchNoDocsQuery();

}

EarlybirdFieldConstant field =

EarlybirdFieldConstants.CSF\_NAME\_TO\_MIN\_ENGAGEMENT\_FIELD\_MAP.get(featureName);

if (field == null) {

throw new IllegalArgumentException(String.format("Expected the feature to be "

+ "FAVORITE\_COUNT, REPLY\_COUNT, or RETWEET\_COUNT. Got %s.", featureName));

}

int operand = (int) Double.parseDouble(operands.get(0));

ByteNormalizer normalizer = MinFeatureValueFilter.getMinFeatureValueNormalizer(featureName);

int minValue = normalizer.unsignedByteToInt(normalizer.normalize(operand));

// We default to the old behavior of filtering posts instead of consulting the min engagement

// field if the operand is less than some threshold value because it seems, empirically, that

// the old method results in lower query latencies for lower values of the filter operand.

// This threshold can be controlled by the "use\_min\_engagement\_field\_threshold" decider. The

// current default value is 90. SEARCH-16102

int useMinEngagementFieldThreshold = decider.getAvailability(

"use\_min\_engagement\_field\_threshold").getOrElse(() -> 0);

if (operand >= useMinEngagementFieldThreshold) {

NUM\_QUERIES\_ABOVE\_MIN\_ENGAGEMENT\_THRESHOLD.increment();

} else {

NUM\_QUERIES\_BELOW\_MIN\_ENGAGEMENT\_THRESHOLD.increment();

}

if (schemaHasField(field) && operand >= useMinEngagementFieldThreshold) {

return buildNoScoreIntTermQuery(op, field, minValue);

}

// Driving by MinFeatureValueFilter's DocIdSetIterator is very slow, because we have to

// perform an expensive check for all doc IDs in the segment, so we use a cached result to

// drive the query, and use MinFeatureValueFilter as a secondary filter.

try {

Query drivingQuery = minEngagmentsDrivingQuery(op, operand);

return new FilteredQuery(

drivingQuery, MinFeatureValueFilter.getDocIdFilterFactory(featureName, operand));

} catch (Exception e) {

// If the filter is not found, that's OK, it might be our first time running the query cache,

// or there may be no Tweets with that many engagements (we would only expect this in tests).

return MinFeatureValueFilter.getMinFeatureValueFilter(featureName, operand);

}

}

private Query minEngagmentsDrivingQuery(SearchOperator operator, int minValue)

throws CachedFilterQuery.NoSuchFilterException, QueryParserException {

// If the min engagements value is large, then many of the hits that have engagement will still

// not match the query, leading to extremely slow queries. Therefore, if there is more than 100

// engagements, we drive by a more restricted filter. See SEARCH-33740

String filter;

if (minValue < 100) {

filter = QueryCacheConstants.HAS\_ENGAGEMENT;

} else if (operator.getOperatorType() == SearchOperator.Type.MIN\_FAVES) {

filter = QueryCacheConstants.MIN\_FAVES\_100;

} else if (operator.getOperatorType() == SearchOperator.Type.MIN\_REPLIES) {

filter = QueryCacheConstants.MIN\_REPLIES\_100;

} else if (operator.getOperatorType() == SearchOperator.Type.MIN\_RETWEETS) {

filter = QueryCacheConstants.MIN\_RETWEETS\_100;

} else {

throw new QueryParserException("Missing engagement filter.");

}

return CachedFilterQuery.getCachedFilterQuery(filter, queryCacheManager);

}

private boolean isOperatorTypeEngagementFilter(SearchOperator.Type type) {

return type == SearchOperator.Type.MIN\_FAVES

|| type == SearchOperator.Type.MIN\_RETWEETS

|| type == SearchOperator.Type.MIN\_REPLIES;

}

private boolean schemaHasField(EarlybirdFieldConstant field) {

return schemaSnapshot.hasField(field.getFieldId());

}

// Helper functions

private Query createSimpleTermQuery(

com.twitter.search.queryparser.query.Query node, String field, String text)

throws QueryParserException {

Query baseQuery = new TermQuery(createTerm(field, text));

if (isGeoFieldThatShouldBeScrubbed(field, text)) {

baseQuery = wrapQueryInUserScrubGeoFilter(baseQuery);

}

return wrapQuery(baseQuery, node, field);

}

private boolean isGeoFieldThatShouldBeScrubbed(String field, String text) {

if (field.equals(EarlybirdFieldConstant.INTERNAL\_FIELD.getFieldName())) {

// the internal field is used for the place id filter and the geo location type filters, some

// of which should be scrubbed

return GEO\_FILTERS\_TO\_BE\_SCRUBBED.contains(text);

}

return GEO\_FIELDS\_TO\_BE\_SCRUBBED.contains(field);

}

// Like above, but sets boost to 0 to disable scoring component. This should be used

// for filters that do not impact scoring (such as filter:images).

private Query createNoScoreTermQuery(com.twitter.search.queryparser.query.Query node,

String field, String text)

throws QueryParserException {

Query query = createSimpleTermQuery(node, field, text);

return new BoostQuery(query, 0.0f); // No score contribution.

}

private Query createNormalizedTermQuery(com.twitter.search.queryparser.query.Query node,

String field, String text)

throws QueryParserException {

return createSimpleTermQuery(

node,

field,

NormalizerHelper.normalizeWithUnknownLocale(text, EarlybirdConfig.getPenguinVersion()));

}

/\*\*

\* Get the boost from the annotation list of a query node.

\* Right now this is very simple, we simple extract the value of some annotations and ignore all

\* others, also, if there are multiple annotations that have values, we only use the first one we

\* see in the list (although the rewritten query EB receives should have this).

\* NOTE: we use simple weight selection logic here based on the assumption that the annotator

\* and rewriter will not produce ambiguous weight information. There should always be only one

\* weight-bearing annotation for a specific node.

\*

\* @param annotations The list of annotations of the query node.

\* @return The boost for this query node, 0 if there is no boost, in which case you shouldn't

\* apply it at all.

\*/

private static double getBoostFromAnnotations(List<Annotation> annotations) {

if (annotations != null) {

for (Annotation anno : annotations) {

switch (anno.getType()) {

case VARIANT:

case SPELLING:

case WEIGHT:

case OPTIONAL:

return ((FloatAnnotation) anno).getValue();

default:

}

}

}

return -1;

}

private static double getPhraseProximityFromAnnotations(List<Annotation> annotations) {

if (annotations != null) {

for (Annotation anno : annotations) {

if (anno.getType() == Annotation.Type.PROXIMITY) {

return ((FloatAnnotation) anno).getValue();

}

}

}

return -1;

}

private static boolean isOptional(com.twitter.search.queryparser.query.Query node) {

return node.hasAnnotationType(Annotation.Type.OPTIONAL);

}

private static boolean isProximityGroup(com.twitter.search.queryparser.query.Query node) {

if (node.isTypeOf(com.twitter.search.queryparser.query.Query.QueryType.OPERATOR)) {

SearchOperator op = (SearchOperator) node;

if (op.getOperatorType() == SearchOperator.Type.PROXIMITY\_GROUP) {

return true;

}

}

return false;

}

private final Query simplifyBooleanQuery(BooleanQuery q) {

if (q.clauses() == null || q.clauses().size() != 1) {

return q;

}

return q.clauses().get(0).getQuery();

}

private Query visit(final Phrase phrase, boolean sloppy) throws QueryParserException {

Optional<Annotation> fieldOpt = phrase.getAnnotationOf(Annotation.Type.FIELD);

if (fieldOpt.isPresent()) {

String field = fieldOpt.get().valueToString();

Schema.FieldInfo fieldInfo = schemaSnapshot.getFieldInfo(field);

if (fieldInfo != null && !fieldInfo.getFieldType().hasPositions()) {

throw new QueryParserException(String.format("Field %s does not support phrase queries "

+ "because it does not have position information.", field));

}

}

BooleanQuery.Builder queryBuilder = new BooleanQuery.Builder();

Map<String, Float> actualFieldWeights = getFieldWeightMapForNode(phrase);

for (Map.Entry<String, Float> entry : actualFieldWeights.entrySet()) {

PhraseQuery.Builder phraseQueryBuilder = new PhraseQuery.Builder();

int curPos = 0;

for (String term : phrase.getTerms()) {

if (!term.equals(PHRASE\_WILDCARD)) {

phraseQueryBuilder.add(createTerm(entry.getKey(), term), curPos);

curPos++;

} else if (curPos != 0) { //"\*" at the beggining of a phrase has no effect/meaning

curPos++;

}

}

// No actual terms added to query

if (curPos == 0) {

break;

}

int annotatedSloppiness = (int) getPhraseProximityFromAnnotations(phrase.getAnnotations());

if (annotatedSloppiness > 0) {

phraseQueryBuilder.setSlop(annotatedSloppiness);

} else if (sloppy) {

phraseQueryBuilder.setSlop(proximityPhraseSlop);

}

float fieldWeight = entry.getValue();

float boost = (float) getBoostFromAnnotations(phrase.getAnnotations());

Query query = phraseQueryBuilder.build();

if (boost >= 0) {

query = BoostUtils.maybeWrapInBoostQuery(query, boost \* fieldWeight);

} else if (fieldWeight != DEFAULT\_FIELD\_WEIGHT) {

query = BoostUtils.maybeWrapInBoostQuery(query, fieldWeight);

} else {

query = BoostUtils.maybeWrapInBoostQuery(query, proximityPhraseWeight);

}

Occur occur = actualFieldWeights.size() > 1 ? Occur.SHOULD : Occur.MUST;

queryBuilder.add(wrapQuery(query, phrase, entry.getKey()), occur);

}

Query q = simplifyBooleanQuery(queryBuilder.build());

return negateQueryIfNodeNegated(phrase, q);

}

private Query wrapQuery(

org.apache.lucene.search.Query query,

com.twitter.search.queryparser.query.Query node,

String fieldName) {

return EarlybirdQueryHelper.maybeWrapWithTimeout(

EarlybirdQueryHelper.maybeWrapWithHitAttributionCollector(

query, node, schemaSnapshot.getFieldInfo(fieldName), hitAttributeHelper),

node, queryTimeout);

}

private final boolean nodeIsNegated(com.twitter.search.queryparser.query.Query node) {

if (isParentNegated(node)) {

return !node.mustNotOccur();

} else {

return node.mustNotOccur();

}

}

private final Query negateQuery(Query q) {

return new BooleanQuery.Builder()

.add(q, Occur.MUST\_NOT)

.add(new MatchAllDocsQuery(), Occur.MUST)

.build();

}

// Simple helper to examine node, and negate the lucene query if necessary.

private final Query negateQueryIfNodeNegated(com.twitter.search.queryparser.query.Query node,

Query query) {

if (query == null) {

return null;

}

return nodeIsNegated(node) ? negateQuery(query) : query;

}

private boolean isParentNegated(com.twitter.search.queryparser.query.Query query) {

return parentNegatedQueries.contains(query);

}

private org.apache.lucene.index.Term createTerm(String field, String text)

throws QueryParserException {

Schema.FieldInfo fieldInfo = schemaSnapshot.getFieldInfo(field);

if (fieldInfo == null) {

throw new QueryParserException("Unknown field: " + field);

}

queriedFields.add(field);

try {

return new org.apache.lucene.index.Term(field, SchemaUtil.toBytesRef(fieldInfo, text));

} catch (UnsupportedOperationException e) {

throw new QueryParserException(e.getMessage(), e.getCause());

}

}

/\*\*

\* Get field weight map for a node, combing default values and its annotations.

\*/

private Map<String, Float> getFieldWeightMapForNode(

com.twitter.search.queryparser.query.Query query) throws QueryParserException {

return FieldWeightUtil.combineDefaultWithAnnotation(

query,

defaultFieldWeightMap,

enabledFieldWeightMap,

Functions.<String>identity(),

mappableFieldMap,

Functions.<String>identity());

}

private boolean addQuery(

BooleanQuery.Builder bqBuilder,

com.twitter.search.queryparser.query.Query child) throws QueryParserException {

Occur occur = Occur.MUST;

if (child.mustNotOccur()) {

// To build a conjunction, we will not rely on the negation in the child visitor.

// Instead we will add the term as MUST\_NOT occur.

// Store this in parentNegatedQueries so the child visitor can do the right thing.

occur = Occur.MUST\_NOT;

parentNegatedQueries.add(child);

} else if (isOptional(child) || isProximityGroup(child)) {

occur = Occur.SHOULD;

}

Query q = child.accept(this);

if (q != null) {

bqBuilder.add(q, occur);

return true;

}

return false;

}

/\*\*

\* Constructs a BooleanQuery from a queryparser Query node.

\* Adds fields as configured in the fieldWeightMap and specified by termQueryDisjunctionType

\* - TermQueryDisjunctionType.ONLY\_OPTIONALIZED adds optional fields

\* (only resolved\_links\_text for now),

\* - TermQueryDisjunctionType.DROP\_OPTIONALIZED adds all other valid fields expect

\* resolved\_links\_text (for now),

\* - TermQueryDisjunctionType.NORMAL adds all valid fields

\* @param query an instance of com.twitter.search.queryparser.query.Query or

\* com.twitter.search.queryparser.query.Term

\* @return a BooleanQuery consists of fields from query

\*/

private BooleanQuery createTermQueryDisjunction(

com.twitter.search.queryparser.query.Query query) throws QueryParserException {

String normTerm = query.isTypeOf(com.twitter.search.queryparser.query.Query.QueryType.TERM)

? ((Term) query).getValue() : query.toString(false);

BooleanQuery.Builder booleanQueryBuilder = new BooleanQuery.Builder();

Map<String, Float> actualFieldWeightMap = getFieldWeightMapForNode(query);

Set<String> fieldsToUse = Sets.newLinkedHashSet(actualFieldWeightMap.keySet());

Occur occur = fieldsToUse.size() > 1 ? Occur.SHOULD : Occur.MUST;

for (String field : fieldsToUse) {

addTermQueryWithField(booleanQueryBuilder, query, normTerm, field, occur,

actualFieldWeightMap.get(field));

}

return booleanQueryBuilder.build();

}

private void addTermQueryWithField(

BooleanQuery.Builder bqBuilder,

com.twitter.search.queryparser.query.Query term,

String normTerm,

String fieldName,

Occur occur,

float fieldWeight) throws QueryParserException {

float boost = (float) getBoostFromAnnotations(term.getAnnotations());

Query query = createSimpleTermQuery(term, fieldName, normTerm);

if (boost >= 0) {

query = BoostUtils.maybeWrapInBoostQuery(query, boost \* fieldWeight);

} else {

query = BoostUtils.maybeWrapInBoostQuery(query, fieldWeight);

}

bqBuilder.add(query, occur);

}

private Query finalizeQuery(BooleanQuery bq, Term term) {

Query q = simplifyBooleanQuery(bq);

return negateQueryIfNodeNegated(term, q);

}

private Rectangle boundingBoxFromSearchOperator(SearchOperator op) throws QueryParserException {

Preconditions.checkArgument(op.getOperatorType() == SearchOperator.Type.GEO\_BOUNDING\_BOX);

Preconditions.checkNotNull(op.getOperands());

Preconditions.checkState(op.getOperands().size() == 4);

List<String> operands = op.getOperands();

try {

// Unfortunately, we store coordinates as floats in our index, which causes a lot of precision

// loss. On the query side, we have to cast into floats to match.

float minLat = (float) Double.parseDouble(operands.get(0));

float minLon = (float) Double.parseDouble(operands.get(1));

float maxLat = (float) Double.parseDouble(operands.get(2));

float maxLon = (float) Double.parseDouble(operands.get(3));

Point lowerLeft = new PointImpl(minLon, minLat, GeohashChunkImpl.getSpatialContext());

Point upperRight = new PointImpl(maxLon, maxLat, GeohashChunkImpl.getSpatialContext());

return new RectangleImpl(lowerLeft, upperRight, GeohashChunkImpl.getSpatialContext());

} catch (NumberFormatException e) {

// consider operator invalid if any of the coordinate cannot be parsed.

throw new QueryParserException("Malformed bounding box operator." + op.serialize());

}

}

private Query visitGeocodeOrGeocodePrivateOperator(SearchOperator op)

throws QueryParserException {

GeoCode geoCode = GeoCode.fromOperator(op);

if (geoCode == null) {

throw new QueryParserException("Invalid GeoCode operator:" + op.serialize());

}

return wrapQueryInUserScrubGeoFilter(

GeoQuadTreeQueryBuilder.buildGeoQuadTreeQuery(geoCode, terminationTracker));

}

private Query wrapQueryInUserScrubGeoFilter(Query baseQuery) {

if (DeciderUtil.isAvailableForRandomRecipient(

decider, "filter\_out\_geo\_scrubbed\_tweets\_" + earlybirdCluster.getNameForStats())) {

return new FilteredQuery(

baseQuery,

UserScrubGeoFilter.getDocIdFilterFactory(userScrubGeoMap));

} else {

return baseQuery;

}

}

private Query buildLongTermAttributeQuery(SearchOperator op, String fieldName) {

return buildLongTermAttributeQuery(op, fieldName, Long.parseLong(op.getOperand()));

}

private Query buildLongTermAttributeQuery(SearchOperator op, String fieldName, long argValue) {

org.apache.lucene.index.Term term = new org.apache.lucene.index.Term(

fieldName, LongTermAttributeImpl.copyIntoNewBytesRef(argValue));

return wrapQuery(new TermQueryWithSafeToString(term, Long.toString(argValue)), op, fieldName);

}

private static void parseLongArgs(List<String> operands,

Collection<Long> arguments,

SearchOperator op) throws QueryParserException {

for (String operand : operands) {

try {

arguments.add(Long.parseLong(operand));

} catch (NumberFormatException e) {

throw new QueryParserException("Invalid long operand in " + op.serialize(), e);

}

}

}

private static boolean isUserIdField(String field) {

return EarlybirdFieldConstant.FROM\_USER\_ID\_FIELD.getFieldName().equals(field)

|| EarlybirdFieldConstant.IN\_REPLY\_TO\_USER\_ID\_FIELD.getFieldName().equals(field)

|| EarlybirdFieldConstant.RETWEET\_SOURCE\_USER\_ID\_FIELD.getFieldName().equals(field)

|| EarlybirdFieldConstant.LIKED\_BY\_USER\_ID\_FIELD.getFieldName().equals(field)

|| EarlybirdFieldConstant.RETWEETED\_BY\_USER\_ID.getFieldName().equals(field)

|| EarlybirdFieldConstant.REPLIED\_TO\_BY\_USER\_ID.getFieldName().equals(field)

|| EarlybirdFieldConstant.QUOTED\_USER\_ID\_FIELD.getFieldName().equals(field)

|| EarlybirdFieldConstant.DIRECTED\_AT\_USER\_ID\_FIELD.getFieldName().equals(field);

}

private static boolean isTweetIdField(String field) {

return EarlybirdFieldConstant.IN\_REPLY\_TO\_TWEET\_ID\_FIELD.getFieldName().equals(field)

|| EarlybirdFieldConstant.RETWEET\_SOURCE\_TWEET\_ID\_FIELD.getFieldName().equals(field)

|| EarlybirdFieldConstant.QUOTED\_TWEET\_ID\_FIELD.getFieldName().equals(field)

|| EarlybirdFieldConstant.CONVERSATION\_ID\_FIELD.getFieldName().equals(field);

}

private static boolean isIdCSFField(String field) {

return EarlybirdFieldConstant.DIRECTED\_AT\_USER\_ID\_CSF.getFieldName().equals(field);

}

public Set<String> getQueriedFields() {

return queriedFields;

}

}