package com.twitter.search.common.converter.earlybird;

import java.io.IOException;

import java.util.Date;

import java.util.List;

import java.util.Optional;

import javax.annotation.concurrent.NotThreadSafe;

import com.google.common.base.Preconditions;

import org.apache.commons.collections.CollectionUtils;

import org.joda.time.DateTime;

import org.joda.time.DateTimeZone;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.common\_internal.text.version.PenguinVersion;

import com.twitter.search.common.converter.earlybird.EncodedFeatureBuilder.TweetFeatureWithEncodeFeatures;

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

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

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

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

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

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

import com.twitter.search.common.partitioning.snowflakeparser.SnowflakeIdParser;

import com.twitter.search.common.relevance.entities.GeoObject;

import com.twitter.search.common.relevance.entities.TwitterMessage;

import com.twitter.search.common.relevance.entities.TwitterQuotedMessage;

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.EarlybirdEncodedFeatures;

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.thriftjava.ThriftDocument;

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

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

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

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

import com.twitter.tweetypie.thriftjava.ComposerSource;

/\*\*

\* Converts a TwitterMessage into a ThriftVersionedEvents. This is only responsible for data that

\* is available immediately when a Tweet is created. Some data, like URL data, isn't available

\* immediately, and so it is processed later, in the DelayedIndexingConverter and sent as an

\* update. In order to achieve this we create the document in 2 passes:

\*

\* 1. BasicIndexingConverter builds thriftVersionedEvents with the fields that do not require

\* external services.

\*

\* 2. DelayedIndexingConverter builds all the document fields depending on external services, once

\* those services have processed the relevant Tweet and we have retrieved that data.

\*/

@NotThreadSafe

public class BasicIndexingConverter {

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

private static final SearchCounter NUM\_NULLCAST\_FEATURE\_FLAG\_SET\_TWEETS =

SearchCounter.export("num\_nullcast\_feature\_flag\_set\_tweets");

private static final SearchCounter NUM\_NULLCAST\_TWEETS =

SearchCounter.export("num\_nullcast\_tweets");

private static final SearchCounter NUM\_NON\_NULLCAST\_TWEETS =

SearchCounter.export("num\_non\_nullcast\_tweets");

private static final SearchCounter ADJUSTED\_BAD\_CREATED\_AT\_COUNTER =

SearchCounter.export("adjusted\_incorrect\_created\_at\_timestamp");

private static final SearchCounter INCONSISTENT\_TWEET\_ID\_AND\_CREATED\_AT\_MS =

SearchCounter.export("inconsistent\_tweet\_id\_and\_created\_at\_ms");

private static final SearchCounter NUM\_SELF\_THREAD\_TWEETS =

SearchCounter.export("num\_self\_thread\_tweets");

private static final SearchCounter NUM\_EXCLUSIVE\_TWEETS =

SearchCounter.export("num\_exclusive\_tweets");

// If a tweet carries a timestamp smaller than this timestamp, we consider the timestamp invalid,

// because twitter does not even exist back then before: Sun, 01 Jan 2006 00:00:00 GMT

private static final long VALID\_CREATION\_TIME\_THRESHOLD\_MILLIS =

new DateTime(2006, 1, 1, 0, 0, 0, DateTimeZone.UTC).getMillis();

private final EncodedFeatureBuilder featureBuilder;

private final Schema schema;

private final EarlybirdCluster cluster;

public BasicIndexingConverter(Schema schema, EarlybirdCluster cluster) {

this.featureBuilder = new EncodedFeatureBuilder();

this.schema = schema;

this.cluster = cluster;

}

/\*\*

\* This function converts TwitterMessage to ThriftVersionedEvents, which is a generic data

\* structure that can be consumed by Earlybird directly.

\*/

public ThriftVersionedEvents convertMessageToThrift(

TwitterMessage message,

boolean strict,

List<PenguinVersion> penguinVersions) throws IOException {

Preconditions.checkNotNull(message);

Preconditions.checkNotNull(penguinVersions);

ThriftVersionedEvents versionedEvents = new ThriftVersionedEvents()

.setId(message.getId());

ImmutableSchemaInterface schemaSnapshot = schema.getSchemaSnapshot();

for (PenguinVersion penguinVersion : penguinVersions) {

ThriftDocument document =

buildDocumentForPenguinVersion(schemaSnapshot, message, strict, penguinVersion);

ThriftIndexingEvent thriftIndexingEvent = new ThriftIndexingEvent()

.setDocument(document)

.setEventType(ThriftIndexingEventType.INSERT)

.setSortId(message.getId());

message.getFromUserTwitterId().map(thriftIndexingEvent::setUid);

versionedEvents.putToVersionedEvents(penguinVersion.getByteValue(), thriftIndexingEvent);

}

return versionedEvents;

}

private ThriftDocument buildDocumentForPenguinVersion(

ImmutableSchemaInterface schemaSnapshot,

TwitterMessage message,

boolean strict,

PenguinVersion penguinVersion) throws IOException {

TweetFeatureWithEncodeFeatures tweetFeature =

featureBuilder.createTweetFeaturesFromTwitterMessage(

message, penguinVersion, schemaSnapshot);

EarlybirdThriftDocumentBuilder builder =

buildBasicFields(message, schemaSnapshot, cluster, tweetFeature);

buildUserFields(builder, message, tweetFeature.versionedFeatures, penguinVersion);

buildGeoFields(builder, message, tweetFeature.versionedFeatures);

buildRetweetAndReplyFields(builder, message, strict);

buildQuotesFields(builder, message);

buildVersionedFeatureFields(builder, tweetFeature.versionedFeatures);

buildAnnotationFields(builder, message);

buildNormalizedMinEngagementFields(builder, tweetFeature.encodedFeatures, cluster);

buildDirectedAtFields(builder, message);

builder.withSpaceIdFields(message.getSpaceIds());

return builder.build();

}

/\*\*

\* Build the basic fields for a tweet.

\*/

public static EarlybirdThriftDocumentBuilder buildBasicFields(

TwitterMessage message,

ImmutableSchemaInterface schemaSnapshot,

EarlybirdCluster cluster,

TweetFeatureWithEncodeFeatures tweetFeature) {

EarlybirdEncodedFeatures extendedEncodedFeatures = tweetFeature.extendedEncodedFeatures;

if (extendedEncodedFeatures == null && EarlybirdCluster.isTwitterMemoryFormatCluster(cluster)) {

extendedEncodedFeatures = EarlybirdEncodedFeatures.newEncodedTweetFeatures(

schemaSnapshot, EarlybirdFieldConstant.EXTENDED\_ENCODED\_TWEET\_FEATURES\_FIELD);

}

EarlybirdThriftDocumentBuilder builder = new EarlybirdThriftDocumentBuilder(

tweetFeature.encodedFeatures,

extendedEncodedFeatures,

new EarlybirdFieldConstants(),

schemaSnapshot);

builder.withID(message.getId());

final Date createdAt = message.getDate();

long createdAtMs = createdAt == null ? 0L : createdAt.getTime();

createdAtMs = fixCreatedAtTimeStampIfNecessary(message.getId(), createdAtMs);

if (createdAtMs > 0L) {

builder.withCreatedAt((int) (createdAtMs / 1000));

}

builder.withTweetSignature(tweetFeature.versionedFeatures.getTweetSignature());

if (message.getConversationId() > 0) {

long conversationId = message.getConversationId();

builder.withLongField(

EarlybirdFieldConstant.CONVERSATION\_ID\_CSF.getFieldName(), conversationId);

// We only index conversation ID when it is different from the tweet ID.

if (message.getId() != conversationId) {

builder.withLongField(

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

}

}

if (message.getComposerSource().isPresent()) {

ComposerSource composerSource = message.getComposerSource().get();

builder.withIntField(

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

if (composerSource == ComposerSource.CAMERA) {

builder.withCameraComposerSourceFlag();

}

}

EarlybirdEncodedFeatures encodedFeatures = tweetFeature.encodedFeatures;

if (encodedFeatures.isFlagSet(EarlybirdFieldConstant.FROM\_VERIFIED\_ACCOUNT\_FLAG)) {

builder.addFilterInternalFieldTerm(EarlybirdFieldConstant.VERIFIED\_FILTER\_TERM);

}

if (encodedFeatures.isFlagSet(EarlybirdFieldConstant.FROM\_BLUE\_VERIFIED\_ACCOUNT\_FLAG)) {

builder.addFilterInternalFieldTerm(EarlybirdFieldConstant.BLUE\_VERIFIED\_FILTER\_TERM);

}

if (encodedFeatures.isFlagSet(EarlybirdFieldConstant.IS\_OFFENSIVE\_FLAG)) {

builder.withOffensiveFlag();

}

if (message.getNullcast()) {

NUM\_NULLCAST\_TWEETS.increment();

builder.addFilterInternalFieldTerm(EarlybirdFieldConstant.NULLCAST\_FILTER\_TERM);

} else {

NUM\_NON\_NULLCAST\_TWEETS.increment();

}

if (encodedFeatures.isFlagSet(EarlybirdFieldConstant.IS\_NULLCAST\_FLAG)) {

NUM\_NULLCAST\_FEATURE\_FLAG\_SET\_TWEETS.increment();

}

if (message.isSelfThread()) {

builder.addFilterInternalFieldTerm(

EarlybirdFieldConstant.SELF\_THREAD\_FILTER\_TERM);

NUM\_SELF\_THREAD\_TWEETS.increment();

}

if (message.isExclusive()) {

builder.addFilterInternalFieldTerm(EarlybirdFieldConstant.EXCLUSIVE\_FILTER\_TERM);

builder.withLongField(

EarlybirdFieldConstant.EXCLUSIVE\_CONVERSATION\_AUTHOR\_ID\_CSF.getFieldName(),

message.getExclusiveConversationAuthorId());

NUM\_EXCLUSIVE\_TWEETS.increment();

}

builder.withLanguageCodes(message.getLanguage(), message.getBCP47LanguageTag());

return builder;

}

/\*\*

\* Build the user fields.

\*/

public static void buildUserFields(

EarlybirdThriftDocumentBuilder builder,

TwitterMessage message,

VersionedTweetFeatures versionedTweetFeatures,

PenguinVersion penguinVersion) {

// 1. Set all the from user fields.

if (message.getFromUserTwitterId().isPresent()) {

builder.withLongField(EarlybirdFieldConstant.FROM\_USER\_ID\_FIELD.getFieldName(),

message.getFromUserTwitterId().get())

// CSF

.withLongField(EarlybirdFieldConstant.FROM\_USER\_ID\_CSF.getFieldName(),

message.getFromUserTwitterId().get());

} else {

LOG.warn("fromUserTwitterId is not set in TwitterMessage! Status id: " + message.getId());

}

if (message.getFromUserScreenName().isPresent()) {

String fromUser = message.getFromUserScreenName().get();

String normalizedFromUser =

NormalizerHelper.normalizeWithUnknownLocale(fromUser, penguinVersion);

builder

.withWhiteSpaceTokenizedScreenNameField(

EarlybirdFieldConstant.TOKENIZED\_FROM\_USER\_FIELD.getFieldName(),

normalizedFromUser)

.withStringField(EarlybirdFieldConstant.FROM\_USER\_FIELD.getFieldName(),

normalizedFromUser);

if (message.getTokenizedFromUserScreenName().isPresent()) {

builder.withCamelCaseTokenizedScreenNameField(

EarlybirdFieldConstant.CAMELCASE\_USER\_HANDLE\_FIELD.getFieldName(),

fromUser,

normalizedFromUser,

message.getTokenizedFromUserScreenName().get());

}

}

Optional<String> toUserScreenName = message.getToUserLowercasedScreenName();

if (toUserScreenName.isPresent() && !toUserScreenName.get().isEmpty()) {

builder.withStringField(

EarlybirdFieldConstant.TO\_USER\_FIELD.getFieldName(),

NormalizerHelper.normalizeWithUnknownLocale(toUserScreenName.get(), penguinVersion));

}

if (versionedTweetFeatures.isSetUserDisplayNameTokenStreamText()) {

builder.withTokenStreamField(EarlybirdFieldConstant.TOKENIZED\_USER\_NAME\_FIELD.getFieldName(),

versionedTweetFeatures.getUserDisplayNameTokenStreamText(),

versionedTweetFeatures.getUserDisplayNameTokenStream());

}

}

/\*\*

\* Build the geo fields.

\*/

public static void buildGeoFields(

EarlybirdThriftDocumentBuilder builder,

TwitterMessage message,

VersionedTweetFeatures versionedTweetFeatures) {

double lat = GeoUtil.ILLEGAL\_LATLON;

double lon = GeoUtil.ILLEGAL\_LATLON;

if (message.getGeoLocation() != null) {

GeoObject location = message.getGeoLocation();

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

location.getLatitude(), location.getLongitude(), location.getAccuracy());

if (location.getSource() != null) {

builder.withStringField(EarlybirdFieldConstant.INTERNAL\_FIELD.getFieldName(),

EarlybirdFieldConstants.formatGeoType(location.getSource()));

}

if (GeoUtil.validateGeoCoordinates(location.getLatitude(), location.getLongitude())) {

lat = location.getLatitude();

lon = location.getLongitude();

}

}

// See SEARCH-14317 for investigation on how much space geo filed is used in archive cluster.

// In lucene archives, this CSF is needed regardless of whether geoLocation is set.

builder.withLatLonCSF(lat, lon);

if (versionedTweetFeatures.isSetTokenizedPlace()) {

Place place = versionedTweetFeatures.getTokenizedPlace();

Preconditions.checkArgument(place.isSetId(), "Place ID not set for tweet "

+ message.getId());

Preconditions.checkArgument(place.isSetFullName(),

"Place full name not set for tweet " + message.getId());

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

builder

.withStringField(EarlybirdFieldConstant.PLACE\_ID\_FIELD.getFieldName(), place.getId())

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

place.getFullName());

if (place.isSetCountryCode()) {

builder.withStringField(EarlybirdFieldConstant.PLACE\_COUNTRY\_CODE\_FIELD.getFieldName(),

place.getCountryCode());

}

}

if (versionedTweetFeatures.isSetTokenizedProfileGeoEnrichment()) {

ProfileGeoEnrichment profileGeoEnrichment =

versionedTweetFeatures.getTokenizedProfileGeoEnrichment();

Preconditions.checkArgument(

profileGeoEnrichment.isSetPotentialLocations(),

"ProfileGeoEnrichment.potentialLocations not set for tweet "

+ message.getId());

List<PotentialLocation> potentialLocations = profileGeoEnrichment.getPotentialLocations();

Preconditions.checkArgument(

!potentialLocations.isEmpty(),

"Found tweet with an empty ProfileGeoEnrichment.potentialLocations: "

+ message.getId());

builder.addFilterInternalFieldTerm(EarlybirdFieldConstant.PROFILE\_GEO\_FILTER\_TERM);

for (PotentialLocation potentialLocation : potentialLocations) {

if (potentialLocation.isSetCountryCode()) {

builder.withStringField(

EarlybirdFieldConstant.PROFILE\_GEO\_COUNTRY\_CODE\_FIELD.getFieldName(),

potentialLocation.getCountryCode());

}

if (potentialLocation.isSetRegion()) {

builder.withStringField(EarlybirdFieldConstant.PROFILE\_GEO\_REGION\_FIELD.getFieldName(),

potentialLocation.getRegion());

}

if (potentialLocation.isSetLocality()) {

builder.withStringField(EarlybirdFieldConstant.PROFILE\_GEO\_LOCALITY\_FIELD.getFieldName(),

potentialLocation.getLocality());

}

}

}

builder.withPlacesField(message.getPlaces());

}

/\*\*

\* Build the retweet and reply fields.

\*/

public static void buildRetweetAndReplyFields(

EarlybirdThriftDocumentBuilder builder,

TwitterMessage message,

boolean strict) {

long retweetUserIdVal = -1;

long sharedStatusIdVal = -1;

if (message.getRetweetMessage() != null) {

if (message.getRetweetMessage().getSharedId() != null) {

sharedStatusIdVal = message.getRetweetMessage().getSharedId();

}

if (message.getRetweetMessage().hasSharedUserTwitterId()) {

retweetUserIdVal = message.getRetweetMessage().getSharedUserTwitterId();

}

}

long inReplyToStatusIdVal = -1;

long inReplyToUserIdVal = -1;

if (message.isReply()) {

if (message.getInReplyToStatusId().isPresent()) {

inReplyToStatusIdVal = message.getInReplyToStatusId().get();

}

if (message.getToUserTwitterId().isPresent()) {

inReplyToUserIdVal = message.getToUserTwitterId().get();

}

}

buildRetweetAndReplyFields(

retweetUserIdVal,

sharedStatusIdVal,

inReplyToStatusIdVal,

inReplyToUserIdVal,

strict,

builder);

}

/\*\*

\* Build the quotes fields.

\*/

public static void buildQuotesFields(

EarlybirdThriftDocumentBuilder builder,

TwitterMessage message) {

if (message.getQuotedMessage() != null) {

TwitterQuotedMessage quoted = message.getQuotedMessage();

if (quoted != null && quoted.getQuotedStatusId() > 0 && quoted.getQuotedUserId() > 0) {

builder.withQuote(quoted.getQuotedStatusId(), quoted.getQuotedUserId());

}

}

}

/\*\*

\* Build directed at field.

\*/

public static void buildDirectedAtFields(

EarlybirdThriftDocumentBuilder builder,

TwitterMessage message) {

if (message.getDirectedAtUserId().isPresent() && message.getDirectedAtUserId().get() > 0) {

builder.withDirectedAtUser(message.getDirectedAtUserId().get());

builder.addFilterInternalFieldTerm(EarlybirdFieldConstant.DIRECTED\_AT\_FILTER\_TERM);

}

}

/\*\*

\* Build the versioned features for a tweet.

\*/

public static void buildVersionedFeatureFields(

EarlybirdThriftDocumentBuilder builder,

VersionedTweetFeatures versionedTweetFeatures) {

builder

.withHashtagsField(versionedTweetFeatures.getHashtags())

.withMentionsField(versionedTweetFeatures.getMentions())

.withStocksFields(versionedTweetFeatures.getStocks())

.withResolvedLinksText(versionedTweetFeatures.getNormalizedResolvedUrlText())

.withTokenStreamField(EarlybirdFieldConstant.TEXT\_FIELD.getFieldName(),

versionedTweetFeatures.getTweetTokenStreamText(),

versionedTweetFeatures.isSetTweetTokenStream()

? versionedTweetFeatures.getTweetTokenStream() : null)

.withStringField(EarlybirdFieldConstant.SOURCE\_FIELD.getFieldName(),

versionedTweetFeatures.getSource())

.withStringField(EarlybirdFieldConstant.NORMALIZED\_SOURCE\_FIELD.getFieldName(),

versionedTweetFeatures.getNormalizedSource());

// Internal fields for smileys and question marks

if (versionedTweetFeatures.hasPositiveSmiley) {

builder.withStringField(

EarlybirdFieldConstant.INTERNAL\_FIELD.getFieldName(),

EarlybirdFieldConstant.HAS\_POSITIVE\_SMILEY);

}

if (versionedTweetFeatures.hasNegativeSmiley) {

builder.withStringField(

EarlybirdFieldConstant.INTERNAL\_FIELD.getFieldName(),

EarlybirdFieldConstant.HAS\_NEGATIVE\_SMILEY);

}

if (versionedTweetFeatures.hasQuestionMark) {

builder.withStringField(EarlybirdFieldConstant.TEXT\_FIELD.getFieldName(),

EarlybirdThriftDocumentBuilder.QUESTION\_MARK);

}

}

/\*\*

\* Build the escherbird annotations for a tweet.

\*/

public static void buildAnnotationFields(

EarlybirdThriftDocumentBuilder builder,

TwitterMessage message) {

List<TwitterMessage.EscherbirdAnnotation> escherbirdAnnotations =

message.getEscherbirdAnnotations();

if (CollectionUtils.isEmpty(escherbirdAnnotations)) {

return;

}

builder.addFacetSkipList(EarlybirdFieldConstant.ENTITY\_ID\_FIELD.getFieldName());

for (TwitterMessage.EscherbirdAnnotation annotation : escherbirdAnnotations) {

String groupDomainEntity = String.format("%d.%d.%d",

annotation.groupId, annotation.domainId, annotation.entityId);

String domainEntity = String.format("%d.%d", annotation.domainId, annotation.entityId);

String entity = String.format("%d", annotation.entityId);

builder.withStringField(EarlybirdFieldConstant.ENTITY\_ID\_FIELD.getFieldName(),

groupDomainEntity);

builder.withStringField(EarlybirdFieldConstant.ENTITY\_ID\_FIELD.getFieldName(),

domainEntity);

builder.withStringField(EarlybirdFieldConstant.ENTITY\_ID\_FIELD.getFieldName(),

entity);

}

}

/\*\*

\* Build the correct ThriftIndexingEvent's fields based on retweet and reply status.

\*/

public static void buildRetweetAndReplyFields(

long retweetUserIdVal,

long sharedStatusIdVal,

long inReplyToStatusIdVal,

long inReplyToUserIdVal,

boolean strict,

EarlybirdThriftDocumentBuilder builder) {

Optional<Long> retweetUserId = Optional.of(retweetUserIdVal).filter(x -> x > 0);

Optional<Long> sharedStatusId = Optional.of(sharedStatusIdVal).filter(x -> x > 0);

Optional<Long> inReplyToUserId = Optional.of(inReplyToUserIdVal).filter(x -> x > 0);

Optional<Long> inReplyToStatusId = Optional.of(inReplyToStatusIdVal).filter(x -> x > 0);

// We have six combinations here. A Tweet can be

// 1) a reply to another tweet (then it has both in-reply-to-user-id and

// in-reply-to-status-id set),

// 2) directed-at a user (then it only has in-reply-to-user-id set),

// 3) not a reply at all.

// Additionally, it may or may not be a Retweet (if it is, then it has retweet-user-id and

// retweet-status-id set).

//

// We want to set some fields unconditionally, and some fields (reference-author-id and

// shared-status-id) depending on the reply/retweet combination.

//

// 1. Normal tweet (not a reply, not a retweet). None of the fields should be set.

//

// 2. Reply to a tweet (both in-reply-to-user-id and in-reply-to-status-id set).

// IN\_REPLY\_TO\_USER\_ID\_FIELD should be set to in-reply-to-user-id

// SHARED\_STATUS\_ID\_CSF should be set to in-reply-to-status-id

// IS\_REPLY\_FLAG should be set

//

// 3. Directed-at a user (only in-reply-to-user-id is set).

// IN\_REPLY\_TO\_USER\_ID\_FIELD should be set to in-reply-to-user-id

// IS\_REPLY\_FLAG should be set

//

// 4. Retweet of a normal tweet (retweet-user-id and retweet-status-id are set).

// RETWEET\_SOURCE\_USER\_ID\_FIELD should be set to retweet-user-id

// SHARED\_STATUS\_ID\_CSF should be set to retweet-status-id

// IS\_RETWEET\_FLAG should be set

//

// 5. Retweet of a reply (both in-reply-to-user-id and in-reply-to-status-id set,

// retweet-user-id and retweet-status-id are set).

// RETWEET\_SOURCE\_USER\_ID\_FIELD should be set to retweet-user-id

// SHARED\_STATUS\_ID\_CSF should be set to retweet-status-id (retweet beats reply!)

// IS\_RETWEET\_FLAG should be set

// IN\_REPLY\_TO\_USER\_ID\_FIELD should be set to in-reply-to-user-id

// IS\_REPLY\_FLAG should NOT be set

//

// 6. Retweet of a directed-at tweet (only in-reply-to-user-id is set,

// retweet-user-id and retweet-status-id are set).

// RETWEET\_SOURCE\_USER\_ID\_FIELD should be set to retweet-user-id

// SHARED\_STATUS\_ID\_CSF should be set to retweet-status-id

// IS\_RETWEET\_FLAG should be set

// IN\_REPLY\_TO\_USER\_ID\_FIELD should be set to in-reply-to-user-id

// IS\_REPLY\_FLAG should NOT be set

//

// In other words:

// SHARED\_STATUS\_ID\_CSF logic: if this is a retweet SHARED\_STATUS\_ID\_CSF should be set to

// retweet-status-id, otherwise if it's a reply to a tweet, it should be set to

// in-reply-to-status-id.

Preconditions.checkState(retweetUserId.isPresent() == sharedStatusId.isPresent());

if (retweetUserId.isPresent()) {

builder.withNativeRetweet(retweetUserId.get(), sharedStatusId.get());

if (inReplyToUserId.isPresent()) {

// Set IN\_REPLY\_TO\_USER\_ID\_FIELD even if this is a retweet of a reply.

builder.withInReplyToUserID(inReplyToUserId.get());

}

} else {

// If this is a retweet of a reply, we don't want to mark it as a reply, or override fields

// set by the retweet logic.

// If we are in this branch, this is not a retweet. Potentially, we set the reply flag,

// and override shared-status-id and reference-author-id.

if (inReplyToStatusId.isPresent()) {

if (strict) {

// Enforcing that if this is a reply to a tweet, then it also has a replied-to user.

Preconditions.checkState(inReplyToUserId.isPresent());

}

builder.withReplyFlag();

builder.withLongField(

EarlybirdFieldConstant.SHARED\_STATUS\_ID\_CSF.getFieldName(),

inReplyToStatusId.get());

builder.withLongField(

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

inReplyToStatusId.get());

}

if (inReplyToUserId.isPresent()) {

builder.withReplyFlag();

builder.withInReplyToUserID(inReplyToUserId.get());

}

}

}

/\*\*

\* Build the engagement fields.

\*/

public static void buildNormalizedMinEngagementFields(

EarlybirdThriftDocumentBuilder builder,

EarlybirdEncodedFeatures encodedFeatures,

EarlybirdCluster cluster) throws IOException {

if (EarlybirdCluster.isArchive(cluster)) {

int favoriteCount = encodedFeatures.getFeatureValue(EarlybirdFieldConstant.FAVORITE\_COUNT);

int retweetCount = encodedFeatures.getFeatureValue(EarlybirdFieldConstant.RETWEET\_COUNT);

int replyCount = encodedFeatures.getFeatureValue(EarlybirdFieldConstant.REPLY\_COUNT);

builder

.withNormalizedMinEngagementField(

EarlybirdFieldConstant.NORMALIZED\_FAVORITE\_COUNT\_GREATER\_THAN\_OR\_EQUAL\_TO\_FIELD

.getFieldName(),

favoriteCount);

builder

.withNormalizedMinEngagementField(

EarlybirdFieldConstant.NORMALIZED\_RETWEET\_COUNT\_GREATER\_THAN\_OR\_EQUAL\_TO\_FIELD

.getFieldName(),

retweetCount);

builder

.withNormalizedMinEngagementField(

EarlybirdFieldConstant.NORMALIZED\_REPLY\_COUNT\_GREATER\_THAN\_OR\_EQUAL\_TO\_FIELD

.getFieldName(),

replyCount);

}

}

/\*\*

\* As seen in SEARCH-5617, we sometimes have incorrect createdAt. This method tries to fix them

\* by extracting creation time from snowflake when possible.

\*/

public static long fixCreatedAtTimeStampIfNecessary(long id, long createdAtMs) {

if (createdAtMs < VALID\_CREATION\_TIME\_THRESHOLD\_MILLIS

&& id > SnowflakeIdParser.SNOWFLAKE\_ID\_LOWER\_BOUND) {

// This tweet has a snowflake ID, and we can extract timestamp from the ID.

ADJUSTED\_BAD\_CREATED\_AT\_COUNTER.increment();

return SnowflakeIdParser.getTimestampFromTweetId(id);

} else if (!SnowflakeIdParser.isTweetIDAndCreatedAtConsistent(id, createdAtMs)) {

LOG.error(

"Found inconsistent tweet ID and created at timestamp: [statusID={}], [createdAtMs={}]",

id, createdAtMs);

INCONSISTENT\_TWEET\_ID\_AND\_CREATED\_AT\_MS.increment();

}

return createdAtMs;

}

}