package com.twitter.search.common.schema;

import java.util.Map;

import java.util.Set;

import javax.annotation.Nullable;

import com.google.common.base.Preconditions;

import com.google.common.collect.ImmutableList;

import com.google.common.collect.Sets;

import com.twitter.common.text.util.CharSequenceTermAttributeSerializer;

import com.twitter.common.text.util.PositionIncrementAttributeSerializer;

import com.twitter.common.text.util.TokenStreamSerializer;

import com.twitter.common.text.util.TokenTypeAttributeSerializer;

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

public class SchemaBuilder {

public static final String CSF\_VIEW\_NAME\_SEPARATOR = ".";

protected final ThriftSchema schema = new ThriftSchema();

protected final FieldNameToIdMapping idMapping;

protected final int tokenStreamSerializerVersion;

// As of now, we do not allow two fields to share the same field name.

// This set is used to perform this check.

private final Set<String> fieldNameSet = Sets.newHashSet();

/\*\*

\* Construct a schema builder with the given FieldNameToIdMapper.

\* A SchemaBuilder is used to build a ThriftSchema incrementally.

\*/

public SchemaBuilder(FieldNameToIdMapping idMapping,

TokenStreamSerializer.Version tokenStreamSerializerVersion) {

this.idMapping = idMapping;

Preconditions.checkArgument(

tokenStreamSerializerVersion == TokenStreamSerializer.Version.VERSION\_2);

this.tokenStreamSerializerVersion = tokenStreamSerializerVersion.ordinal();

}

/\*\*

\* Build ThriftSchema using settings accumulated so far.

\*/

public final ThriftSchema build() {

return schema;

}

/\*\*

\* Uses fieldName also as facetName.

\*/

public final SchemaBuilder withFacetConfigs(String fieldName,

boolean storeSkipList,

boolean storeOffensiveCounters,

boolean useCSFForFacetCounting) {

return withFacetConfigs(

fieldName,

fieldName,

storeSkipList,

storeOffensiveCounters,

useCSFForFacetCounting);

}

/\*\*

\* Add facet field configuration.

\*/

public final SchemaBuilder withFacetConfigs(String fieldName,

String facetName,

boolean storeSkipList,

boolean storeOffensiveCounters,

boolean useCSFForFacetCounting) {

if (!shouldIncludeField(fieldName)) {

return this;

}

ThriftFacetFieldSettings facetSettings = new ThriftFacetFieldSettings();

// As of now, all our facet names are the same as field names

facetSettings.setFacetName(facetName);

facetSettings.setStoreSkiplist(storeSkipList);

facetSettings.setStoreOffensiveCounters(storeOffensiveCounters);

facetSettings.setUseCSFForFacetCounting(useCSFForFacetCounting);

int fieldId = idMapping.getFieldID(fieldName);

ThriftFieldConfiguration fieldConfiguration = schema.getFieldConfigs().get(fieldId);

Preconditions.checkNotNull(fieldConfiguration,

"In Earlybird, a facet field must be indexed. "

+ "No ThriftIndexedFieldSettings found for field " + fieldName);

fieldConfiguration.getSettings().setFacetFieldSettings(facetSettings);

return this;

}

/\*\*

\* Configure the given field ID to be used for partitioning.

\*/

public final SchemaBuilder withPartitionFieldId(int partitionFieldId) {

schema.setPartitionFieldId(partitionFieldId);

return this;

}

/\*\*

\* Add a column stride field into schema.

\*/

public final SchemaBuilder withColumnStrideField(String fieldName,

ThriftCSFType type,

int numValuesPerDoc,

boolean updatable,

boolean loadIntoRam) {

return withColumnStrideField(fieldName, type, numValuesPerDoc, updatable, loadIntoRam, null);

}

/\*\*

\* Add a column stride field into schema that is variable length.

\*/

public final SchemaBuilder withBinaryColumnStrideField(String fieldName,

boolean loadIntoRam) {

if (!shouldIncludeField(fieldName)) {

return this;

}

ThriftCSFFieldSettings csfFieldSettings = new ThriftCSFFieldSettings();

csfFieldSettings.setCsfType(ThriftCSFType.BYTE)

.setVariableLength(true)

.setLoadIntoRAM(loadIntoRam);

ThriftFieldSettings fieldSettings =

new ThriftFieldSettings().setCsfFieldSettings(csfFieldSettings);

ThriftFieldConfiguration fieldConf =

new ThriftFieldConfiguration(fieldName).setSettings(fieldSettings);

putIntoFieldConfigs(idMapping.getFieldID(fieldName), fieldConf);

return this;

}

/\*\*

\* Add a column stride field into schema which has a default value.

\*/

public final SchemaBuilder withColumnStrideField(String fieldName,

ThriftCSFType type,

int numValuesPerDoc,

boolean updatable,

boolean loadIntoRam,

Long defaultValue) {

if (!shouldIncludeField(fieldName)) {

return this;

}

ThriftCSFFieldSettings csfFieldSettings = new ThriftCSFFieldSettings();

csfFieldSettings.setCsfType(type)

.setVariableLength(false)

.setFixedLengthSettings(

new ThriftFixedLengthCSFSettings()

.setNumValuesPerDoc(numValuesPerDoc)

.setUpdateable(updatable))

.setLoadIntoRAM(loadIntoRam);

if (defaultValue != null) {

csfFieldSettings.setDefaultValue(defaultValue);

}

ThriftFieldSettings fieldSettings =

new ThriftFieldSettings().setCsfFieldSettings(csfFieldSettings);

ThriftFieldConfiguration fieldConf =

new ThriftFieldConfiguration(fieldName).setSettings(fieldSettings);

putIntoFieldConfigs(idMapping.getFieldID(fieldName), fieldConf);

return this;

}

/\*\*

\* Add a CSF view into schema. A view is a portion of another CSF.

\*/

public final SchemaBuilder withColumnStrideFieldView(

String fieldName,

ThriftCSFType csfType,

ThriftCSFType outputCSFType,

String baseFieldName,

int valueIndex,

int bitStartPosition,

int bitLength,

ThriftFeatureNormalizationType featureNormalizationType,

@Nullable Set<ThriftFeatureUpdateConstraint> constraints) {

if (!shouldIncludeField(fieldName)) {

return this;

}

int baseFieldConfigID = idMapping.getFieldID(baseFieldName);

ThriftCSFViewSettings csfViewSettings = new ThriftCSFViewSettings()

.setBaseFieldConfigId(baseFieldConfigID)

.setCsfType(csfType)

.setValueIndex(valueIndex)

.setBitStartPosition(bitStartPosition)

.setBitLength(bitLength);

if (outputCSFType != null) {

csfViewSettings.setOutputCSFType(outputCSFType);

}

if (featureNormalizationType != ThriftFeatureNormalizationType.NONE) {

csfViewSettings.setNormalizationType(featureNormalizationType);

}

if (constraints != null) {

csfViewSettings.setFeatureUpdateConstraints(constraints);

}

ThriftFieldSettings fieldSettings = new ThriftFieldSettings()

.setCsfViewSettings(csfViewSettings);

ThriftFieldConfiguration fieldConf = new ThriftFieldConfiguration(fieldName)

.setSettings(fieldSettings);

Map<Integer, ThriftFieldConfiguration> fieldConfigs = schema.getFieldConfigs();

verifyCSFViewSettings(fieldConfigs, fieldConf);

putIntoFieldConfigs(idMapping.getFieldID(fieldName), fieldConf);

return this;

}

/\*\*

\* Sanity checks for CSF view settings.

\*/

public static void verifyCSFViewSettings(Map<Integer, ThriftFieldConfiguration> fieldConfigs,

ThriftFieldConfiguration fieldConf) {

Preconditions.checkNotNull(fieldConf.getSettings());

Preconditions.checkNotNull(fieldConf.getSettings().getCsfViewSettings());

ThriftCSFViewSettings csfViewSettings = fieldConf.getSettings().getCsfViewSettings();

if (fieldConfigs != null) {

ThriftFieldConfiguration baseFieldConfig = fieldConfigs.get(

csfViewSettings.getBaseFieldConfigId());

if (baseFieldConfig != null) {

String baseFieldName = baseFieldConfig.getFieldName();

String expectedViewNamePrefix = baseFieldName + CSF\_VIEW\_NAME\_SEPARATOR;

if (fieldConf.getFieldName().startsWith(expectedViewNamePrefix)) {

ThriftFieldSettings baseFieldSettings = baseFieldConfig.getSettings();

ThriftCSFFieldSettings baseFieldCSFSettings = baseFieldSettings.getCsfFieldSettings();

if (baseFieldCSFSettings != null) {

if (!baseFieldCSFSettings.isVariableLength()

&& baseFieldCSFSettings.getFixedLengthSettings() != null) {

ThriftCSFType baseCSFType = baseFieldCSFSettings.getCsfType();

switch (baseCSFType) {

case BYTE:

checkCSFViewPositions(baseFieldCSFSettings, 8, csfViewSettings);

break;

case INT:

checkCSFViewPositions(baseFieldCSFSettings, 32, csfViewSettings);

break;

default:

throw new IllegalStateException("Base field: " + baseFieldName

+ " is of a non-supported CSFType: " + baseCSFType);

}

} else {

throw new IllegalStateException("Base field: " + baseFieldName

+ " must be a fixed-length CSF field");

}

} else {

throw new IllegalStateException("Base field: " + baseFieldName + " is not a CSF field");

}

} else {

throw new IllegalStateException("View field name for baseFieldConfigID: "

+ csfViewSettings.getBaseFieldConfigId() + " must start with: '"

+ expectedViewNamePrefix + "'");

}

} else {

throw new IllegalStateException("Can't add a view, no field defined for base fieldID: "

+ csfViewSettings.getBaseFieldConfigId());

}

} else {

throw new IllegalStateException("Can't add a view, no field configs defined.");

}

}

private static void checkCSFViewPositions(ThriftCSFFieldSettings baseFieldCSFSettings,

int bitsPerValue,

ThriftCSFViewSettings csfViewSettings) {

ThriftFixedLengthCSFSettings fixedLengthCSFSettings =

baseFieldCSFSettings.getFixedLengthSettings();

Preconditions.checkNotNull(fixedLengthCSFSettings);

int numValues = fixedLengthCSFSettings.getNumValuesPerDoc();

Preconditions.checkState(csfViewSettings.getValueIndex() >= 0,

"value index must be positive: " + csfViewSettings.getValueIndex());

Preconditions.checkState(csfViewSettings.getValueIndex() < numValues, "value index "

+ csfViewSettings.getValueIndex() + " must be less than numValues: " + numValues);

Preconditions.checkState(csfViewSettings.getBitStartPosition() >= 0,

"bitStartPosition must be positive: " + csfViewSettings.getBitStartPosition());

Preconditions.checkState(csfViewSettings.getBitStartPosition() < bitsPerValue,

"bitStartPosition " + csfViewSettings.getBitStartPosition()

+ " must be less than bitsPerValue " + bitsPerValue);

Preconditions.checkState(csfViewSettings.getBitLength() >= 1,

"bitLength must be positive: " + csfViewSettings.getBitLength());

Preconditions.checkState(

csfViewSettings.getBitStartPosition() + csfViewSettings.getBitLength() <= bitsPerValue,

String.format("bitStartPosition (%d) + bitLength (%d) must be less than bitsPerValue (%d)",

csfViewSettings.getBitStartPosition(), csfViewSettings.getBitLength(), bitsPerValue));

}

// No position; no freq; not pretokenized; not tokenized.

/\*\*

\* Norm is disabled as default. Like Lucene string field, or int/long fields.

\*/

public final SchemaBuilder withIndexedNotTokenizedField(String fieldName) {

return withIndexedNotTokenizedField(fieldName, false);

}

/\*\*

\* Add an indexed but not tokenized field. This is similar to Lucene's StringField.

\*/

public final SchemaBuilder withIndexedNotTokenizedField(String fieldName,

boolean supportOutOfOrderAppends) {

return withIndexedNotTokenizedField(fieldName, supportOutOfOrderAppends, true);

}

private final SchemaBuilder withIndexedNotTokenizedField(String fieldName,

boolean supportOutOfOrderAppends,

boolean omitNorms) {

if (!shouldIncludeField(fieldName)) {

return this;

}

ThriftFieldSettings settings = getNoPositionNoFreqSettings(supportOutOfOrderAppends);

settings.getIndexedFieldSettings().setOmitNorms(omitNorms);

ThriftFieldConfiguration config = new ThriftFieldConfiguration(fieldName)

.setSettings(settings);

putIntoFieldConfigs(idMapping.getFieldID(fieldName), config);

return this;

}

/\*\* Makes the given field searchable by default, with the given weight. \*/

public final SchemaBuilder withSearchFieldByDefault(

String fieldName, float textSearchableFieldWeight) {

if (!shouldIncludeField(fieldName)) {

return this;

}

ThriftFieldSettings settings =

schema.getFieldConfigs().get(idMapping.getFieldID(fieldName)).getSettings();

settings.setSearchFieldSettings(

new ThriftSearchFieldSettings()

.setTextSearchableFieldWeight(textSearchableFieldWeight)

.setTextDefaultSearchable(true));

return this;

}

/\*\*

\* Similar to Lucene's TextField. The string is analyzed using the default/override analyzer.

\* @param fieldName

\* @param addHfPairIfHfFieldsArePresent Add hfPair fields if they exists in the schema.

\* For certain text fields, adding hfPair fields are usually preferred, but they may

\* not exist in the schema, in which case the hfPair fields will not be added.

\*/

public final SchemaBuilder withTextField(String fieldName,

boolean addHfPairIfHfFieldsArePresent) {

if (!shouldIncludeField(fieldName)) {

return this;

}

ThriftFieldConfiguration config = new ThriftFieldConfiguration(fieldName).setSettings(

getDefaultSettings(ThriftIndexOptions.DOCS\_AND\_FREQS\_AND\_POSITIONS));

if (addHfPairIfHfFieldsArePresent) {

// Add hfPair fields only if they exist in the schema for the cluster

boolean hfPair = shouldIncludeField(ImmutableSchema.HF\_TERM\_PAIRS\_FIELD)

&& shouldIncludeField(ImmutableSchema.HF\_PHRASE\_PAIRS\_FIELD);

config.getSettings().getIndexedFieldSettings().setIndexHighFreqTermPairs(hfPair);

}

config.getSettings().getIndexedFieldSettings().setTokenized(true);

putIntoFieldConfigs(idMapping.getFieldID(fieldName), config);

return this;

}

/\*\*

\* Marked the given field as having per position payload.

\*/

public final SchemaBuilder withPerPositionPayload(String fieldName, int defaultPayloadLength) {

if (!shouldIncludeField(fieldName)) {

return this;

}

ThriftFieldSettings settings =

schema.getFieldConfigs().get(idMapping.getFieldID(fieldName)).getSettings();

settings.getIndexedFieldSettings().setStorePerPositionPayloads(true);

settings.getIndexedFieldSettings().setDefaultPerPositionPayloadLength(defaultPayloadLength);

return this;

}

/\*\*

\* Add field into schema that is pre-tokenized and does not have position.

\* E.g. hashtags / stocks / card\_domain

\*/

public final SchemaBuilder withPretokenizedNoPositionField(String fieldName) {

if (!shouldIncludeField(fieldName)) {

return this;

}

ThriftFieldConfiguration config = new ThriftFieldConfiguration(fieldName)

.setSettings(getPretokenizedNoPositionFieldSetting());

// Add hfPair fields only if they exist in the schema for the cluster

boolean hfPair = shouldIncludeField(ImmutableSchema.HF\_TERM\_PAIRS\_FIELD)

&& shouldIncludeField(ImmutableSchema.HF\_PHRASE\_PAIRS\_FIELD);

config.getSettings().getIndexedFieldSettings().setIndexHighFreqTermPairs(hfPair);

putIntoFieldConfigs(idMapping.getFieldID(fieldName), config);

return this;

}

/\*\*

\* Mark the field to have ordered term dictionary.

\* In Lucene, term dictionary is sorted. In Earlybird, term dictionary order is not

\* guaranteed unless this is turned on.

\*/

public final SchemaBuilder withOrderedTerms(String fieldName) {

if (!shouldIncludeField(fieldName)) {

return this;

}

ThriftFieldSettings settings =

schema.getFieldConfigs().get(idMapping.getFieldID(fieldName)).getSettings();

settings.getIndexedFieldSettings().setSupportOrderedTerms(true);

return this;

}

/\*\*

\* Support lookup of term text by term id in the term dictionary.

\*/

public final SchemaBuilder withTermTextLookup(String fieldName) {

if (!shouldIncludeField(fieldName)) {

return this;

}

ThriftFieldSettings settings =

schema.getFieldConfigs().get(idMapping.getFieldID(fieldName)).getSettings();

settings.getIndexedFieldSettings().setSupportTermTextLookup(true);

return this;

}

/\*\*

\* Add a text field that is pre-tokenized, so not analyzed again in the index (e.g. Earlybird).

\*

\* Note that the token streams MUST be created using the attributes defined in

\* {@link com.twitter.search.common.util.text.TweetTokenStreamSerializer}.

\*/

public final SchemaBuilder withPretokenizedTextField(

String fieldName,

boolean addHfPairIfHfFieldsArePresent) {

if (!shouldIncludeField(fieldName)) {

return this;

}

ThriftFieldConfiguration config = new ThriftFieldConfiguration(fieldName)

.setSettings(getDefaultPretokenizedSettings(

ThriftIndexOptions.DOCS\_AND\_FREQS\_AND\_POSITIONS));

putIntoFieldConfigs(idMapping.getFieldID(fieldName), config);

// Add hfPair fields only if they exist in the schema for the cluster

if (addHfPairIfHfFieldsArePresent) {

// Add hfPair fields only if they exist in the schema for the cluster

boolean hfPair = shouldIncludeField(ImmutableSchema.HF\_TERM\_PAIRS\_FIELD)

&& shouldIncludeField(ImmutableSchema.HF\_PHRASE\_PAIRS\_FIELD);

config.getSettings().getIndexedFieldSettings().setIndexHighFreqTermPairs(hfPair);

}

return this;

}

/\*\*

\* Add a feature configuration

\*/

public final SchemaBuilder withFeatureConfiguration(String baseFieldName, String viewName,

FeatureConfiguration featureConfiguration) {

return withColumnStrideFieldView(

viewName,

// Defaulting all encoded tweet features to int since the underlying encoded tweet features

// are ints.

ThriftCSFType.INT,

featureConfiguration.getOutputType(),

baseFieldName,

featureConfiguration.getValueIndex(),

featureConfiguration.getBitStartPosition(),

featureConfiguration.getBitLength(),

featureConfiguration.getFeatureNormalizationType(),

featureConfiguration.getUpdateConstraints()

);

}

/\*\*

\* Add a long field in schema. This field uses LongTermAttribute.

\*/

private SchemaBuilder addLongTermField(String fieldName, boolean useSortableEncoding) {

if (!shouldIncludeField(fieldName)) {

return this;

}

ThriftFieldSettings longTermSettings = getEarlybirdNumericFieldSettings();

ThriftTokenStreamSerializer tokenStreamSerializer =

new ThriftTokenStreamSerializer(tokenStreamSerializerVersion);

tokenStreamSerializer.setAttributeSerializerClassNames(

ImmutableList.<String>of(LongTermAttributeSerializer.class.getName()));

longTermSettings.getIndexedFieldSettings().setTokenStreamSerializer(tokenStreamSerializer);

ThriftIndexedNumericFieldSettings numericFieldSettings =

new ThriftIndexedNumericFieldSettings(true);

numericFieldSettings.setNumericType(ThriftNumericType.LONG);

numericFieldSettings.setUseSortableEncoding(useSortableEncoding);

longTermSettings.getIndexedFieldSettings().setNumericFieldSettings(numericFieldSettings);

putIntoFieldConfigs(idMapping.getFieldID(fieldName),

new ThriftFieldConfiguration(fieldName).setSettings(longTermSettings));

return this;

}

public final SchemaBuilder withSortableLongTermField(String fieldName) {

return addLongTermField(fieldName, true);

}

public final SchemaBuilder withLongTermField(String fieldName) {

return addLongTermField(fieldName, false);

}

/\*\*

\* Add an int field in schema. This field uses IntTermAttribute.

\*/

public final SchemaBuilder withIntTermField(String fieldName) {

if (!shouldIncludeField(fieldName)) {

return this;

}

ThriftFieldSettings intTermSettings = getEarlybirdNumericFieldSettings();

ThriftTokenStreamSerializer attributeSerializer =

new ThriftTokenStreamSerializer(tokenStreamSerializerVersion);

attributeSerializer.setAttributeSerializerClassNames(

ImmutableList.<String>of(IntTermAttributeSerializer.class.getName()));

intTermSettings.getIndexedFieldSettings().setTokenStreamSerializer(attributeSerializer);

ThriftIndexedNumericFieldSettings numericFieldSettings =

new ThriftIndexedNumericFieldSettings(true);

numericFieldSettings.setNumericType(ThriftNumericType.INT);

intTermSettings.getIndexedFieldSettings().setNumericFieldSettings(numericFieldSettings);

putIntoFieldConfigs(idMapping.getFieldID(fieldName),

new ThriftFieldConfiguration(fieldName).setSettings(intTermSettings));

return this;

}

/\*\*

\* Timeline and ExpertSearch uses

\* {@link com.twitter.search.common.util.analysis.PayloadWeightedTokenizer} to store weighted

\* values.

\*

\* E.g. for the PRODUCED\_LANGUAGES and CONSUMED\_LANGUAGES fields, they contain not a single,

\* value, but instead a list of values with a weight associated with each value.

\*

\* This method adds an indexed field that uses

\* {@link com.twitter.search.common.util.analysis.PayloadWeightedTokenizer}.

\*/

public final SchemaBuilder withCharTermPayloadWeightedField(String fieldName) {

ThriftFieldConfiguration config = new ThriftFieldConfiguration(fieldName)

.setSettings(getPayloadWeightedSettings(ThriftIndexOptions.DOCS\_AND\_FREQS\_AND\_POSITIONS));

putIntoFieldConfigs(idMapping.getFieldID(fieldName), config);

return this;

}

/\*\*

\* Set the version and description of this schema.

\*/

public final SchemaBuilder withSchemaVersion(

int majorVersionNumber,

int minorVersionNumber,

String versionDesc,

boolean isOfficial) {

schema.setMajorVersionNumber(majorVersionNumber);

schema.setMinorVersionNumber(minorVersionNumber);

schema.setVersion(majorVersionNumber + ":" + versionDesc);

schema.setVersionIsOfficial(isOfficial);

return this;

}

public final SchemaBuilder withSchemaVersion(

int majorVersionNumber,

String versionDesc,

boolean isOfficial) {

return withSchemaVersion(majorVersionNumber, 0, versionDesc, isOfficial);

}

protected void putIntoFieldConfigs(int id, ThriftFieldConfiguration config) {

if (schema.getFieldConfigs() != null && schema.getFieldConfigs().containsKey(id)) {

throw new IllegalStateException("Already have a ThriftFieldConfiguration for field id " + id);

}

if (fieldNameSet.contains(config.getFieldName())) {

throw new IllegalStateException("Already have a ThriftFieldConfiguration for field "

+ config.getFieldName());

}

fieldNameSet.add(config.getFieldName());

schema.putToFieldConfigs(id, config);

}

// Default field settings. Most field settings are similar to this.

protected ThriftFieldSettings getDefaultSettings(ThriftIndexOptions indexOption) {

return getDefaultSettings(indexOption, false);

}

protected ThriftFieldSettings getDefaultSettings(ThriftIndexOptions indexOption,

boolean supportOutOfOrderAppends) {

ThriftFieldSettings fieldSettings = new ThriftFieldSettings();

ThriftIndexedFieldSettings indexedFieldSettings = new ThriftIndexedFieldSettings();

indexedFieldSettings

.setIndexed(true)

.setStored(false)

.setTokenized(false)

.setStoreTermVectors(false)

.setStoreTermVectorOffsets(false)

.setStoreTermVectorPayloads(false)

.setStoreTermVectorPositions(false)

.setSupportOutOfOrderAppends(supportOutOfOrderAppends)

.setIndexOptions(indexOption)

.setOmitNorms(true); // All Earlybird fields omit norms.

fieldSettings.setIndexedFieldSettings(indexedFieldSettings);

return fieldSettings;

}

/\*\*

\* Default field settings for fields that are pretokenized

\*

\* The fields that use these settings will need to be tokenized using a serializer with the

\* attributes defined in {@link com.twitter.search.common.util.text.TweetTokenStreamSerializer}.

\*/

protected final ThriftFieldSettings getDefaultPretokenizedSettings(

ThriftIndexOptions indexOption) {

ThriftFieldSettings fieldSettings = getDefaultSettings(indexOption);

fieldSettings.getIndexedFieldSettings().setTokenized(true);

ThriftTokenStreamSerializer attributeSerializer =

new ThriftTokenStreamSerializer(tokenStreamSerializerVersion);

attributeSerializer.setAttributeSerializerClassNames(

ImmutableList.<String>of(

CharSequenceTermAttributeSerializer.class.getName(),

PositionIncrementAttributeSerializer.class.getName(),

TokenTypeAttributeSerializer.class.getName()));

fieldSettings.getIndexedFieldSettings().setTokenStreamSerializer(attributeSerializer);

return fieldSettings;

}

protected final ThriftFieldSettings getPretokenizedNoPositionFieldSetting() {

return getDefaultPretokenizedSettings(ThriftIndexOptions.DOCS\_AND\_FREQS);

}

protected final ThriftFieldSettings getNoPositionNoFreqSettings() {

return getNoPositionNoFreqSettings(false);

}

protected final ThriftFieldSettings getNoPositionNoFreqSettings(

boolean supportOutOfOrderAppends) {

return getDefaultSettings(ThriftIndexOptions.DOCS\_ONLY, supportOutOfOrderAppends);

}

protected final ThriftFieldSettings getEarlybirdNumericFieldSettings() {

// Supposedly numeric fields are not tokenized.

// However, Earlybird uses SingleTokenTokenStream to handle int/long fields.

// So we need to set indexed to true for these fields.

ThriftFieldSettings settings = getNoPositionNoFreqSettings();

settings.getIndexedFieldSettings().setTokenized(true);

return settings;

}

private ThriftFieldSettings getPayloadWeightedSettings(ThriftIndexOptions indexOption) {

ThriftFieldSettings fieldSettings = getDefaultSettings(indexOption);

fieldSettings.getIndexedFieldSettings().setTokenized(true);

ThriftTokenStreamSerializer attributeSerializer =

new ThriftTokenStreamSerializer(tokenStreamSerializerVersion);

attributeSerializer.setAttributeSerializerClassNames(

ImmutableList.<String>of(CharTermAttributeSerializer.class.getName(),

PositionIncrementAttributeSerializer.class.getName(),

PayloadAttributeSerializer.class.getName()));

fieldSettings.getIndexedFieldSettings().setTokenStreamSerializer(attributeSerializer);

return fieldSettings;

}

protected boolean shouldIncludeField(String fieldName) {

return true;

}

}