package com.twitter.search.common.schema.base;

import java.util.LinkedHashMap;

import java.util.Map;

import com.google.common.collect.ImmutableMap;

import com.google.common.collect.Maps;

import static com.google.common.base.Preconditions.checkNotNull;

/\*\*

\* Records whether a field's enabled for search by default and its default weight. Note that these

\* two are decoupled -- a field can have a default weight but not enabled for search by default.

\* In a query it can be enabled by an annotation that does not specify a weight (e.g., ":f:foo"),

\* which would then use the default weight.

\*

\* Instances are mutable.

\*/

public class FieldWeightDefault {

private final boolean enabled;

private final float weight;

public FieldWeightDefault(boolean enabled, float weight) {

this.enabled = enabled;

this.weight = weight;

}

public static FieldWeightDefault fromSignedWeight(float signedValue) {

return new FieldWeightDefault(signedValue >= 0, Math.abs(signedValue));

}

/\*\*

\* Returns an immutable map from field name to default field weights for only enabled fields.

\* Fields that are not enabled for search by default will not be included.

\*/

public static <T> ImmutableMap<T, Float> getOnlyEnabled(

Map<T, FieldWeightDefault> map) {

ImmutableMap.Builder<T, Float> builder = ImmutableMap.builder();

for (Map.Entry<T, FieldWeightDefault> entry : map.entrySet()) {

if (entry.getValue().isEnabled()) {

builder.put(entry.getKey(), entry.getValue().getWeight());

}

}

return builder.build();

}

public boolean isEnabled() {

return enabled;

}

public float getWeight() {

return weight;

}

/\*\*

\* Overlays the base field-weight map with the given one. Since it is an overlay, a

\* field that does not exist in the base map will never be added. Also, negative value means

\* the field is not enabled for search by default, but if it is, the absolute value would serve as

\* the default.

\*/

public static ImmutableMap<String, FieldWeightDefault> overrideFieldWeightMap(

Map<String, FieldWeightDefault> base,

Map<String, Double> fieldWeightMapOverride) {

checkNotNull(base);

if (fieldWeightMapOverride == null) {

return ImmutableMap.copyOf(base);

}

LinkedHashMap<String, FieldWeightDefault> map = Maps.newLinkedHashMap(base);

for (Map.Entry<String, Double> entry : fieldWeightMapOverride.entrySet()) {

if (base.containsKey(entry.getKey())

&& entry.getValue() >= -Float.MAX\_VALUE

&& entry.getValue() <= Float.MAX\_VALUE) {

map.put(

entry.getKey(),

FieldWeightDefault.fromSignedWeight(entry.getValue().floatValue()));

}

}

return ImmutableMap.copyOf(map);

}

/\*\*

\* Creates a field-to-FieldWeightDefault map from the given field-to-weight map, where negative

\* weight means the the field is not enabled for search by default, but if it is (e.g.,

\* by annotation), the absolute value of the weight shall be used.

\*/

public static <T> ImmutableMap<T, FieldWeightDefault> fromSignedWeightMap(

Map<T, ? extends Number> signedWeightMap) {

ImmutableMap.Builder<T, FieldWeightDefault> builder = ImmutableMap.builder();

for (Map.Entry<T, ? extends Number> entry : signedWeightMap.entrySet()) {

// If double to float conversion failed, we will get a float infinity.

// See http://stackoverflow.com/a/10075093/716468

float floatValue = entry.getValue().floatValue();

if (floatValue != Float.NEGATIVE\_INFINITY

&& floatValue != Float.POSITIVE\_INFINITY) {

builder.put(

entry.getKey(),

FieldWeightDefault.fromSignedWeight(floatValue));

}

}

return builder.build();

}

}