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

import java.util.Set;

import javax.annotation.Nullable;

import com.google.common.base.Preconditions;

import com.google.common.collect.Sets;

import com.twitter.common.base.MorePreconditions;

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

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

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

// FeatureConfiguration is defined for all the column stride view fields.

public final class FeatureConfiguration {

private final String name;

private final int intIndex;

// Start position in the given int (0-31)

private final int bitStartPos;

// Length in bits of the feature

private final int bitLength;

// precomputed for reuse

private final int bitMask;

private final int inverseBitMask;

private final int maxValue;

private final ThriftCSFType type;

// This is the client seen feature type: if this is null, this field is unused.

@Nullable

private final ThriftCSFType outputType;

private final String baseField;

private final Set<FeatureConstraint> featureUpdateConstraints;

private final ThriftFeatureNormalizationType featureNormalizationType;

/\*\*

\* Creates a new FeatureConfiguration with a base field.

\*

\* @param intIndex which integer is the feature in (0 based).

\* @param bitStartPos at which bit does the feature start (0-31).

\* @param bitLength length in bits of the feature

\* @param baseField the CSF this feature is stored within.

\*/

private FeatureConfiguration(

String name,

ThriftCSFType type,

ThriftCSFType outputType,

int intIndex,

int bitStartPos,

int bitLength,

String baseField,

Set<FeatureConstraint> featureUpdateConstraints,

ThriftFeatureNormalizationType featureNormalizationType) {

Preconditions.checkState(bitStartPos + bitLength <= Integer.SIZE,

"Feature must not cross int boundary.");

this.name = MorePreconditions.checkNotBlank(name);

this.type = Preconditions.checkNotNull(type);

this.outputType = outputType;

this.intIndex = intIndex;

this.bitStartPos = bitStartPos;

this.bitLength = bitLength;

// Technically, int-sized features can use all 32 bits to store a positive value greater than

// Integer.MAX\_VALUE. But in practice, we will convert the values of those features to Java ints

// on the read side, so the max value for those features will still be Integer.MAX\_VALUE.

this.maxValue = (1 << Math.min(bitLength, Integer.SIZE - 1)) - 1;

this.bitMask = (int) (((1L << bitLength) - 1) << bitStartPos);

this.inverseBitMask = ~bitMask;

this.baseField = baseField;

this.featureUpdateConstraints = featureUpdateConstraints;

this.featureNormalizationType = Preconditions.checkNotNull(featureNormalizationType);

}

public String getName() {

return name;

}

public int getMaxValue() {

return maxValue;

}

@Override

public String toString() {

return new StringBuilder().append(name)

.append(" (").append(intIndex).append(", ")

.append(bitStartPos).append(", ")

.append(bitLength).append(") ").toString();

}

public int getValueIndex() {

return intIndex;

}

public int getBitStartPosition() {

return bitStartPos;

}

public int getBitLength() {

return bitLength;

}

public int getBitMask() {

return bitMask;

}

public int getInverseBitMask() {

return inverseBitMask;

}

public String getBaseField() {

return baseField;

}

public ThriftCSFType getType() {

return type;

}

@Nullable

public ThriftCSFType getOutputType() {

return outputType;

}

public ThriftFeatureNormalizationType getFeatureNormalizationType() {

return featureNormalizationType;

}

/\*\*

\* Returns the update constraint for the feature.

\*/

public Set<ThriftFeatureUpdateConstraint> getUpdateConstraints() {

if (featureUpdateConstraints == null) {

return null;

}

Set<ThriftFeatureUpdateConstraint> constraintSet = Sets.newHashSet();

for (FeatureConstraint constraint : featureUpdateConstraints) {

constraintSet.add(constraint.getType());

}

return constraintSet;

}

/\*\*

\* Returns true if the given update satisfies all feature update constraints.

\*/

public boolean validateFeatureUpdate(final Number oldValue, final Number newValue) {

if (featureUpdateConstraints != null) {

for (FeatureConstraint contraint : featureUpdateConstraints) {

if (!contraint.apply(oldValue, newValue)) {

return false;

}

}

}

return true;

}

@Override

public int hashCode() {

return (name == null ? 0 : name.hashCode())

+ intIndex \* 7

+ bitStartPos \* 13

+ bitLength \* 23

+ bitMask \* 31

+ inverseBitMask \* 43

+ (int) maxValue \* 53

+ (type == null ? 0 : type.hashCode()) \* 61

+ (outputType == null ? 0 : outputType.hashCode()) \* 71

+ (baseField == null ? 0 : baseField.hashCode()) \* 83

+ (featureUpdateConstraints == null ? 0 : featureUpdateConstraints.hashCode()) \* 87

+ (featureNormalizationType == null ? 0 : featureNormalizationType.hashCode()) \* 97;

}

@Override

public boolean equals(Object obj) {

if (!(obj instanceof FeatureConfiguration)) {

return false;

}

FeatureConfiguration featureConfiguration = FeatureConfiguration.class.cast(obj);

return (name == featureConfiguration.name)

&& (bitStartPos == featureConfiguration.bitStartPos)

&& (bitLength == featureConfiguration.bitLength)

&& (bitMask == featureConfiguration.bitMask)

&& (inverseBitMask == featureConfiguration.inverseBitMask)

&& (maxValue == featureConfiguration.maxValue)

&& (type == featureConfiguration.type)

&& (outputType == featureConfiguration.outputType)

&& (baseField == featureConfiguration.baseField)

&& (featureUpdateConstraints == null

? featureConfiguration.featureUpdateConstraints == null

: featureUpdateConstraints.equals(featureConfiguration.featureUpdateConstraints))

&& (featureNormalizationType == null

? featureConfiguration.featureNormalizationType == null

: featureNormalizationType.equals(featureConfiguration.featureNormalizationType));

}

private interface FeatureConstraint {

boolean apply(Number oldValue, Number newValue);

ThriftFeatureUpdateConstraint getType();

}

public static Builder builder() {

return new Builder();

}

public static final class Builder {

private String name;

private ThriftCSFType type;

private ThriftCSFType outputType;

private int intIndex;

// Start position in the given int (0-31)

private int bitStartPos;

// Length in bits of the feature

private int bitLength;

private String baseField;

private Set<FeatureConstraint> featureUpdateConstraints;

private ThriftFeatureNormalizationType featureNormalizationType =

ThriftFeatureNormalizationType.NONE;

public FeatureConfiguration build() {

return new FeatureConfiguration(name, type, outputType, intIndex, bitStartPos, bitLength,

baseField, featureUpdateConstraints, featureNormalizationType);

}

public Builder withName(String n) {

this.name = n;

return this;

}

public Builder withType(ThriftCSFType featureType) {

this.type = featureType;

return this;

}

public Builder withOutputType(ThriftCSFType featureFeatureType) {

this.outputType = featureFeatureType;

return this;

}

public Builder withFeatureNormalizationType(

ThriftFeatureNormalizationType normalizationType) {

this.featureNormalizationType = Preconditions.checkNotNull(normalizationType);

return this;

}

/\*\*

\* Sets the bit range at the given intIndex, startPos and length.

\*/

public Builder withBitRange(int index, int startPos, int length) {

this.intIndex = index;

this.bitStartPos = startPos;

this.bitLength = length;

return this;

}

public Builder withBaseField(String baseFieldName) {

this.baseField = baseFieldName;

return this;

}

/\*\*

\* Adds a feature update constraint.

\*/

public Builder withFeatureUpdateConstraint(final ThriftFeatureUpdateConstraint constraint) {

if (featureUpdateConstraints == null) {

featureUpdateConstraints = Sets.newHashSet();

}

switch (constraint) {

case IMMUTABLE:

featureUpdateConstraints.add(new FeatureConstraint() {

@Override public boolean apply(Number oldValue, Number newValue) {

return false;

}

@Override public ThriftFeatureUpdateConstraint getType() {

return ThriftFeatureUpdateConstraint.IMMUTABLE;

}

});

break;

case INC\_ONLY:

featureUpdateConstraints.add(new FeatureConstraint() {

@Override public boolean apply(Number oldValue, Number newValue) {

return newValue.intValue() > oldValue.intValue();

}

@Override public ThriftFeatureUpdateConstraint getType() {

return ThriftFeatureUpdateConstraint.INC\_ONLY;

}

});

break;

case POSITIVE:

featureUpdateConstraints.add(new FeatureConstraint() {

@Override public boolean apply(Number oldValue, Number newValue) {

return newValue.intValue() >= 0;

}

@Override public ThriftFeatureUpdateConstraint getType() {

return ThriftFeatureUpdateConstraint.POSITIVE;

}

});

break;

default:

}

return this;

}

private Builder() {

}

}

}