KIP-968: Support single-key_multi-timestamp interactive queries (IQv2) for versioned state stores

This page is meant as a template for writing a KIP. To create a KIP choose Tools->Copy on this page and modify with your content and replace the heading with the next KIP number and a description of your issue. Replace anything in italics with your own description.

Status

Current state: Accepted

Discussion thread: here

JIRA: KAFKA-15347

Please keep the discussion on the mailing list rather than commenting on the wiki (wiki discussions get unwieldy fast).

Motivation

The main goal is supporting interactive queries in presence of versioned state stores (KIP-889) in AK. This KIP is the successor of KIP-960 and discusses single-key, multi-timestamp queries. Other types of IQs are explained in the following KIP (KIP-969)

Key Queries with multiple timestamps:

1. single-key query with upper bound timestamp
2. single-key query with lower bound timestamp
3. single-key query with timestamp range
4. single-key all versions query

Public Interfaces

In this KIP,

- we propose a public class, **MultiVersionedKeyQuery**
- and a public enum **ResultOrder**
- Moreover, the public interface **VersionedRecordIterator** is added to iterate over different values that are returned from a single-key query (each value corresponds to a timestamp).
- In addition, a new method is added to the **VersionedKeyValueStore** interface to support single-key_multi-timestamp queries.
- Finally, a field called *validTo* is added to the **VersionedRecord** class to enable us representing tombstones as well.

Proposed Changes

To be able to list the tombstones, the *validTo* Optional field is added to the **VersionedRecord** class. The default value of *validTo* is *Optional.empty()* which means the record is still valid.

VersionedRecord.java

```
package org.apache.kafka.streams.state;

public final class VersionedRecord<V> {

         /**
     * Create a new {@link VersionedRecord} instance. {@code value} cannot be {@code null}.
     *
     * @param value      the value
     * @param timestamp  the timestamp
     */
    public VersionedRecord(final V value, final long timestamp) {
        this.value = Objects.requireNonNull(value);
        this.timestamp = timestamp;
        this.validTo = Optional.empty();
    }

        /**
     * Create a new {@link VersionedRecord} instance. {@code value} cannot be {@code null}.
     *
     * @param value      The value
     * @param timestamp  The timestamp
     * @param validTo    The exclusive upper bound of the validity interval
     */
     public VersionedRecord(final V value, final long timestamp, final Optional<Long> validTo);


    /**
     * Returns the {@code validTo}
     */
     public Optional<Long> validTo();
}
```

For single-key queries, **MultiVersionedKeyQuery** and **VersionedRecordIterator** classes will be used.

VersionedRecordIterator.java

```
package org.apache.kafka.streams.state;

/**
 * Iterator interface of {@link V}.
 * <p>
 * Users must call its {@code close} method explicitly upon completeness to release resources,
 * or use try-with-resources statement (available since JDK7) for this {@link Closeable} class.
 * Note that {@code remove()} is not supported.
 *
 * @param <V> Type of values
 */
public interface VersionedRecordIterator<V> extends Iterator<VersionedRecord<V>>, Closeable {

    @Override
    void close();
}
```

ResultOrder enum
It helps with specifying the order of the returned results by the query.

ResultOrder

```
package org.apache.kafka.streams.query;

public enum ResultOrder {
    ANY,
    ASCENDING,
    DESCENDING
}
```

MultiVersionedKeyQuery class

- The methods are composable. The *fromTime(Instant fromTime)* and *toTime(Instant toTime)* methods specify the time range.
    - If a user applies the same time limit multiple times such as MultiVersionedKeyQuery.withKey(k).fromTime(t1).fromTime(t2), then the last one wins (it will be translated to MultiVersionedKeyQuery.withKey(k).fromTime(t2)).
    - Defining a query with time range (empty, t1] will be translated into [0, t1] (calling only the *toTime(t1)* method).
    - Defining a query with time range [t1, empty) will be translated into [t1, MAX) (calling only the *fromTime(t1)* method).
    - A query with no specified time range will be translated into [0, MAX). It means that the query will return all the versions of the records with specified key.
- As explained in the javadocs, the query returns all valid records within the specified time range.
    - The *fromTime* specifies the starting point. There can be records which have been inserted before the *fromTime* and are valid in the time range. For example, if the record (k,v) has been inserted at time=0, it will be returned by the multi versioned key queries with key=k and fromTime>=0. Obviously, if the record (k,v) becomes tombstone at time=2, then the multi versioned key queries with key=k and fromTime>=2 will not return it any more. In this case, the multi versioned key queries with key=k and fromTime<2 will return the record (k, v) validTo=2.
    - The *toTime* specifies the ending point. Records that have been inserted at *toTime* are returned by the query as well.
- No ordering is guaranteed for the results, but the results can be sorted by timestamp (in ascending or descending order) by calling the corresponding defined methods (*withAscendingTimestamps()* and *withDescendingTimestamps()* respectively).
    -

MultiVersionedKeyQuery.java

```
package org.apache.kafka.streams.query;

/**
 * Interactive query for retrieving a set of records with the same specified key and different timestamps
within the specified time range.
 * No ordering is guaranteed for the results, but the results can be sorted by timestamp (in ascending or
descending order) by calling the corresponding defined methods.
 *
 *  @param <K> The type of the key.
 *  @param <V> The type of the result returned by this query.
 */

@Evolving
public final class MultiVersionedKeyQuery<K, V> implements Query<VersionedRecordIterator<V>> {

  private final K key;
  private final Optional<Instant> fromTime;
  private final Optional<Instant> toTime;
  private final ResultOrder order;

  private MultiVersionedKeyQuery(
      final K key,
      final Optional<Instant> fromTime,
      final Optional<Instant> toTime,
      final ResultOrder order) {
    this.key = Objects.requireNonNull(key);
    this.fromTime = fromTime;
    this.toTime = toTime;
    this.order = order;
  }


  /**
   * Creates a query that will retrieve the set of records identified by {@code key} if any exists
   * (or {@code null} otherwise).
   *
   * <p>
```

```
   * While the query by default returns the all the record versions of the specified {@code key}, setting
   * the {@code fromTimestamp} (by calling the {@link #fromTime(Instant)} method), and the {@code toTimestamp}
   * (by calling the {@link #toTime(Instant)} method) makes the query to return the record versions associated
   * to the specified time range.
   *
   * @param key The specified key by the query
   * @param <K> The type of the key
   * @param <V> The type of the value that will be retrieved
   * @throws NullPointerException if @param key is null
   */

  public static <K, V> MultiVersionedKeyQuery<K, V> withKey(final K key);

  /**
   * Specifies the starting time point for the key query.
   * <p>
   * The key query returns all the records that are still existing in the time range starting from the
timestamp {@code fromTime}. There can
   * be records which have been inserted before the {@code fromTime} and are still valid in the query specified
time range (the whole time range
   * or even partially). The key query in fact returns all the records that have NOT become tombstone at or
after {@code fromTime}.
   *
   * @param fromTime The starting time point
   * If {@code fromTime} is null, will be considered as negative infinity, ie, no lower bound
   */
  public MultiVersionedKeyQuery<K, V> fromTime(final Instant fromTime);

  /**
   * Specifies the ending time point for the key query.
   * The key query returns all the records that have timestamp <= toTime.
   *
   * @param toTime The ending time point
   * If @param toTime is null, will be considered as positive infinity, ie, no upper bound
   */
  public MultiVersionedKeyQuery<K, V> toTime(final Instant toTime);

  /**
   * Specifies the order of the returned records by the query as descending by timestamp.
   */
  public MultiVersionedKeyQuery<K, V> withDescendingTimestamps();

  /**
   * Specifies the order of the returned records by the query as ascending by timestamp.
   */
  public MultiVersionedKeyQuery<K, V> withAscendingTimestamps();

   /**
   * The key that was specified for this query.
   * The specified {@code key} of the query.
   */
  public K key();

  /**
   * The starting time point of the query, if specified
   * @return The specified {@code fromTime} of the query.
   */
  public Optional<Instant> fromTime();

  /**
   * The ending time point of the query, if specified
   * @return The specified {@code toTime} of the query.
   */
  public Optional<Instant> toTime();

 /**
  * The order of the returned records by timestamp.
  * @return UNORDERED, ASCENDING, or DESCENDING if the query returns records in an unordered, ascending, or
descending order of timestamps.
  */
```

```
  public ResultOrder resultOrder();
}
```

Examples

The following example illustrates the use of the VersionedKeyQuery class to query a versioned state store.
Imagine we have the following records

put(1, 1, time=2023-01-**01**T10:00:00.00Z)

put(1, null, time=2023-01-**05**T10:00:00.00Z)

put(1, null, time=2023-01-**10**T10:00:00.00Z)

put(1, 2, time=2023-01-**15**T10:00:00.00Z)

put(1, 3, time=2023-01-**20**T10:00:00.00Z)

```
// example 1: MultiVersionedKeyQuery without specifying any time bound will be interpreted as all versions
final MultiVersionedKeyQuery<Integer, Integer> query1 = MultiVersionedKeyQuery.withKey(1);

final StateQueryRequest<VersionedRecordIterator<Integer>> request1 = StateQueryRequest.inStore("my_store").
withQuery(query1);

final StateQueryResult<VersionedRecordIterator<Integer>> versionedKeyResult1 = kafkaStreams.query(request1);

// Get the results from all partitions
final Map<Integer, QueryResult<VersionedRecordIterator<Integer>>> partitionResults1 = versionedKeyResult1.
getPartitionResults();
for (final Entry<Integer, QueryResult<VersionedRecordIterator<Integer>>> entry : partitionResults1.entrySet()) {
        try (final VersionedRecordIterator<Integer> iterator = entry.getValue().getResult()) {
        while (iterator.hasNext()) {
          final VersionedRecord<Integer> record = iterator.next();
          Long timestamp = record.timestamp();
          Long validTo = record.validTo();
          Integer value = record.value();
                  System.out.println ("value: " + value + ", timestamp: " + Instant.ofEpochSecond(timestamp)+
", valid till: " + Instant.ofEpochSecond(validTo));
        }
     }
}
/* the printed output will be
        value: 1, timestamp: 2023-01-01T10:00:00.00Z, valid till: 2023-01-05T10:00:00.00Z
        value: 2, timestamp: 2023-01-15T10:00:00.00Z, valid till: 2023-01-20T10:00:00.00Z
        value: 3, timestamp: 2023-01-20T10:00:00.00Z, valid till: now
*/

// example 2: The value of the record with key=1 from 2023-01-17 Time: 10:00:00.00Z till 2023-01-25 T10:00:00.00
Z

MultiVersionedKeyQuery<Integer, Integer> query2 = MultiVersionedKeyQuery.withKey(1);
query2 = query2.fromTime(Instant.parse("2023-01-17T10:00:00.00Z")).toTime(Instant.parse("2023-01-25T10:00:00.00
Z"))

final StateQueryRequest<VersionedRecordIterator<Integer>> request2 = StateQueryRequest.inStore("my_store").
withQuery(query2);

final StateQueryResult<VersionedRecordIterator<Integer>> versionedKeyResult2 = kafkaStreams.query(request2);

// Get the results from all partitions
final Map<Integer, QueryResult<VersionedRecordIterator<Integer>>> partitionResults2 = versionedKeyResult2.
getPartitionResults();
for (final Entry<Integer, QueryResult<VersionedRecordIterator<Integer>>> entry : partitionResults2.entrySet()) {
        try (final VersionedRecordIterator<Integer> iterator = entry.getValue().getResult()) {
        while (iterator.hasNext()) {
          final VersionedRecord<Integer> record = iterator.next();
          Long timestamp = record.timestamp();
          Long validTo = record.validTo();
          Integer value = record.value();
                  System.out.println ("value: " + value + ", timestamp: " + Instant.ofEpochSecond(timestamp)+
", valid till: " + Instant.ofEpochSecond(validTo));
        }
     }
}
/* the printed output will be
        value: 2, timestamp: 2023-01-15T10:00:00.00Z, valid till: 2023-01-20T10:00:00.00Z
        value: 3, timestamp: 2023-01-20T10:00:00.00Z, valid till: now
*/
```

Compatibility, Deprecation, and Migration Plan

- Since this is a completely new set of APIs, no backward compatibility concerns are anticipated.
- Since nothing is deprecated in this KIP, users have no need to migrate unless they want to.

Rejected Alternatives

In order to be able to retrieve the consecutive tombstones, we can have a method or flag (disabled by default) to allow users to get all tombstones. If it is a real use case for the users, we will add it later.

Test Plan

The single-key_multi-timestamp interactive queries will be tested in versioned stored IQv2 integration test (like non-versioned key queries). Moreover , there will be unit tests where ever needed.