KIP-763: Range queries with open endpoints

Status

Current state: Implemented (for 3.1.0)

Discussion thread: *here*

JIRA: KAFKA-4063

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

Motivation

Currently, state stores in Kafka streams only support bounded ranged queries. More precisely, the range (and reverseRange) interface in the ReadOnlyKeyValueStore look as follows:

```
/**
 * Get an iterator over a given range of keys. This iterator must be closed after use.
 * The returned iterator must be safe from {@link java.util.ConcurrentModificationException}s
 * and must not return null values.
 * Order is not guaranteed as bytes lexicographical ordering might not represent key order.
 *
 * @param from The first key that could be in the range, where iteration starts from.
 * @param to   The last key that could be in the range, where iteration ends.
 * @return The iterator for this range, from smallest to largest bytes.
 * @throws NullPointerException       If null is used for from or to.
 * @throws InvalidStateStoreException if the store is not initialized
 */
KeyValueIterator<K, V> range(K from, K to);

/**
 * Get a reverse iterator over a given range of keys. This iterator must be closed after use.
 * The returned iterator must be safe from {@link java.util.ConcurrentModificationException}s
 * and must not return null values.
 * Order is not guaranteed as bytes lexicographical ordering might not represent key order.
 *
 * @param from The first key that could be in the range, where iteration ends.
 * @param to   The last key that could be in the range, where iteration starts from.
 * @return The reverse iterator for this range, from largest to smallest key bytes.
 * @throws NullPointerException       If null is used for from or to.
 * @throws InvalidStateStoreException if the store is not initialized
 */
default KeyValueIterator<K, V> reverseRange(K from, K to) {
    throw new UnsupportedOperationException();
}
```

The requested range is bounded because both the *from* and the *to* parameter needs to be a valid object of key type K and must not be null.

Unlike bounded queries, unbounded queries are currently not supported in an efficient manner. For instance, the only way to retrieve all the records with a key smaller or greater a certain constant is to query the full range range (e.g., using the all() method) and then filter out elements greater or smaller the given value. In this KIP we propose to a solution to directly query unbounded ranges of records from the store store, without the need for extra filtering.

Proposed Changes

The proposed solution is a small change to the semantics of the existing range interface in the ReadOnlyKeyValueStore. Specifically, we propose to allow the use of *null* values as a way to represent unbounded ranges. With this change, the range interface signature will remain the same, only the set of accepted values for the parameters *from* and *to* would change, thus, requiring a modifications to the JavaDoc method description. The proposed change is shown below:

```
/**
 * Get an iterator over a given range of keys.
 * @param from The first key that could be in the range, where iteration starts from.
 * A null value indicates a starting position from the first element in the store.
 * @param to   The last key that could be in the range, where iteration ends.
 * A null value indicates that the range ends with the last element in the store.
 * @return The iterator for this range, from smallest to largest bytes.
 */
KeyValueIterator<K, V> range(K from, K to);

/**
 * Get a reverse iterator over a given range of keys. This iterator must be closed after use.
 * The returned iterator must be safe from {@link java.util.ConcurrentModificationException}s
 * and must not return null values.
 * Order is not guaranteed as bytes lexicographical ordering might not represent key order.
 *
 * @param from The first key that could be in the range, where iteration ends.
 * A null value indicates a starting position from the first element in the store.
 * @param to   The last key that could be in the range, where iteration starts from.
 * A null value indicates that the range ends with the last element in the store.
 * @return The reverse iterator for this range, from largest to smallest key bytes.
 * @throws InvalidStateStoreException if the store is not initialized
 */
default KeyValueIterator<K, V> reverseRange(K from, K to) {
    throw new UnsupportedOperationException();
}
```

The following code snippet illustrates how unbounded range queries can be issued after those changes:

```
//return an iterator for all the records with a key greater (or equal) than "key-x"
//starting with the smallest record
KeyValueIterator<String, String> iter = store.range("key-x", null);

//return an iterator for all the records with a key less (or equal) than "key-x"
//starting with the smallest record
KeyValueIterator<String, String> iter = store.range(null, "key-x");

//return an iterator for all the records with a key greater (or equal) than "key-x"
//starting with the largest record
KeyValueIterator<String, String> iter = store.reverseRange("key-x", null);

//return all the records in the store, equivalent to store.all()
KeyValueIterator<String, String> iter = store.range(null, null);
```

Public Interfaces

- JavaDoc description of org.apache.kafka.streams.state.ReadOnlyKeyValueStore::range
- JavaDoc description of org.apache.kafka.streams.state.ReadOnlyKeyValueStore::reverseRange

Compatibility, Deprecation, and Migration Plan

The proposed changes do not affect existing bounded range queries, therefore no migration is necessary. The proposed changes are also backward compatible because the use of null values has previously been disallowed explicitly.

Rejected Alternatives

Adding new method methods for unbounded range queries

One alternative to the proposed solution is to add new methods for querying unbounded ranges:

```
/**
 * Get an iterator over all keys less than or equal to the key passed in.
 * @param to The last key that could be in the range
 * @return The iterator for this range.
 * @throws NullPointerException If null is used for to.
 */
default KeyValueIterator<K, V> rangeUntil(K to) { throw new UnsupportedOperationException(); }

/**
 * Get an iterator over all keys greater than or equal to the key passed in.
 * @param from The first key that could be in the range
 * @return The iterator for this range.
 * @throws NullPointerException If null is used for from.
 */
default KeyValueIterator<K, V> rangeFrom(K from) { throw new UnsupportedOperationException();
```

We rejected this solution because of two reasons. First, because of the nested architecture of the Kafka streams state stores, adding new methods to the state store interface requires modifications to more than a dozen of interfaces which increases the potential for bugs and makes code maintenance and testing more difficult. These issues are further amplified if in the future we want to add variations to those range calls, such as the ability to configure one of the bounds to be inclusive or exclusive.

Modify existing range interface

Yet another solution would be replace the existing range interface with something more powerful that can support different types of range queries with a single method. An example of such an interface is given blow.

```
void range(Range<K> range)

class Range<K> {
  K from;
  K to;
  ...
  static <K> from(K key) ...
  static <K> until(K key) ..
  // etc
}
```

While a solution like this would be the cleanest, it would require that we deprecate the old range interface and migrate existing code to the new interface. Compared to the solution proposed in this KIP, which does not require modifications to the interface and no migration is needed, we felt that making bigger changes to the range interface is not necessary at this point.

Expanding idea of open endpoint range interfaces to other stores (e.g., SessionStore, WindowStore)

There are several other APIs that offer similar range based interfaces. Examples are the findSession interface in ReadOnlySessionStore or the fetch interface in ReadOnlyWindowStore. One could apply similar ideas of applying open endpoint semantics to those interfaces. At this point, however, we decided not to expand the scope of this KIP and leave it to follow-up KIPs to propose changes to those interfaces.