KIP-766: fetch/findSessions queries with open endpoints for SessionStore/WindowStore

Status

Current state: *"Accept"*

Discussion thread: *here*

JIRA: *here*

Motivation

This is a follow-up KIP for KIP-763: Range queries with open endpoints. In KIP-763, we focused on **ReadOnlyKeyValueStore**, in this KIP, we'll focus on **ReadOnlySessionStore** and **ReadOnlyWindowStore.**

Currently, **ReadOnlySessionStore**/**ReadOnlyWindowStore** in Kafka streams only support bounded ranged queries. More precisely, the fetch /backwardFetch, and findSessions/backwardFindSessions interfaces.

In the **ReadOnlyWindowStore** looks as follows:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
```

```
/**
 * Get all the key-value pairs in the given key range and time range from all the existing windows.
 * <p>
 * This iterator must be closed after use.
 *
 * @param keyFrom     the first key in the range
 * @param keyTo       the last key in the range
 * @param timeFrom time range start (inclusive), where iteration starts.
 * @param timeTo   time range end (inclusive), where iteration ends.
 * @return an iterator over windowed key-value pairs {@code <Windowed<K>, value>}, from beginning
to end of time.
 * @throws InvalidStateStoreException if the store is not initialized
 * @throws NullPointerException       if {@code null} is used for any key.
 * @throws IllegalArgumentException   if duration is negative or can't be represented as {@code lon
g milliseconds}
 */
KeyValueIterator<Windowed<K>, V> fetch(K keyFrom, K keyTo, Instant timeFrom, Instant timeTo)
    throws IllegalArgumentException;

/**
 * Get all the key-value pairs in the given key range and time range from all the existing windows
 * in backward order with respect to time (from end to beginning of time).
 * <p>
 * This iterator must be closed after use.
 *
 * @param keyFrom     the first key in the range
 * @param keyTo       the last key in the range
 * @param timeFrom time range start (inclusive), where iteration ends.
 * @param timeTo   time range end (inclusive), where iteration starts.
 * @return an iterator over windowed key-value pairs {@code <Windowed<K>, value>}, from end to
beginning of time.
 * @throws InvalidStateStoreException if the store is not initialized
 * @throws NullPointerException       if {@code null} is used for any key.
 * @throws IllegalArgumentException   if duration is negative or can't be represented as {@code lon
g milliseconds}
 */
default KeyValueIterator<Windowed<K>, V> backwardFetch(K keyFrom, K keyTo, Instant timeFrom,
Instant timeTo)
    throws IllegalArgumentException  {
    throw new UnsupportedOperationException();
}
```

In the **ReadOnlySessionStore** looks as follows:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
```

```
/**
 * Fetch any sessions in the given range of keys and the sessions end is &ge;
 * earliestSessionEndTime and the sessions start is &le; latestSessionStartTime iterating from
 * earliest to latest.
 * <p>
 * This iterator must be closed after use.
 *
 * @param keyFrom               The first key that could be in the range
 * @param keyTo                 The last key that could be in the range
 * @param earliestSessionEndTime the end timestamp of the earliest session to search for, where
 *                              iteration starts.
 * @param latestSessionStartTime the end timestamp of the latest session to search for, where
 *                              iteration ends.
 * @return iterator of sessions with the matching keys and aggregated values, from earliest to
 * latest session time.
 * @throws NullPointerException If null is used for any key.
 */
default KeyValueIterator<Windowed<K>, AGG> findSessions(final K keyFrom,
                                                        final K keyTo,
                                                        final long earliestSessionEndTime,
                                                        final long latestSessionStartTime) {
    throw new UnsupportedOperationException(
        "This API is not supported by this implementation of ReadOnlySessionStore.");
}

/**
 * Fetch any sessions in the given range of keys and the sessions end is &ge;
 * earliestSessionEndTime and the sessions start is &le; latestSessionStartTime iterating from
 * earliest to latest.
 * <p>
 * This iterator must be closed after use.
 *
```

```
     * @param keyFrom               The first key that could be in the range
     * @param keyTo                 The last key that could be in the range
     * @param earliestSessionEndTime the end timestamp of the earliest session to search for, where
     *                              iteration starts.
     * @param latestSessionStartTime the end timestamp of the latest session to search for, where
     *                              iteration ends.
     * @return iterator of sessions with the matching keys and aggregated values, from earliest to
     * latest session time.
     * @throws NullPointerException If null is used for any key.
     */
    default KeyValueIterator<Windowed<K>, AGG> findSessions(final K keyFrom,
                                                            final K keyTo,
                                                            final Instant earliestSessionEndTime,
                                                            final Instant latestSessionStartTime) {
        throw new UnsupportedOperationException(
            "This API is not supported by this implementation of ReadOnlySessionStore.");
    }

    /**
     * Fetch any sessions in the given range of keys and the sessions end is &ge;
     * earliestSessionEndTime and the sessions start is &le; latestSessionStartTime iterating from
     * latest to earliest.
     * <p>
     * This iterator must be closed after use.
     *
     * @param keyFrom               The first key that could be in the range
     * @param keyTo                 The last key that could be in the range
     * @param earliestSessionEndTime the end timestamp of the earliest session to search for, where
     *                              iteration ends.
     * @param latestSessionStartTime the end timestamp of the latest session to search for, where
     *                              iteration starts.
     * @return backward iterator of sessions with the matching keys and aggregated values, from
     * latest to earliest session time.
     * @throws NullPointerException If null is used for any key.
     */
    default KeyValueIterator<Windowed<K>, AGG> backwardFindSessions(final K keyFrom,
                                                                    final K keyTo,
                                                                    final long earliestSessionEndTim
    e,
                                                                    final long latestSessionStartTim
    e) {
        throw new UnsupportedOperationException(
            "This API is not supported by this implementation of ReadOnlySessionStore.");
    }

    /**
     * Fetch any sessions in the given range of keys and the sessions end is &ge;
     * earliestSessionEndTime and the sessions start is &le; latestSessionStartTime iterating from
     * latest to earliest.
     * <p>
     * This iterator must be closed after use.
     *
     * @param keyFrom               The first key that could be in the range
     * @param keyTo                 The last key that could be in the range
     * @param earliestSessionEndTime the end timestamp of the earliest session to search for, where
     *                              iteration ends.
     * @param latestSessionStartTime the end timestamp of the latest session to search for, where
     *                              iteration starts.
     * @return backward iterator of sessions with the matching keys and aggregated values, from
     * latest to earliest session time.
     * @throws NullPointerException If null is used for any key.
     */
    default KeyValueIterator<Windowed<K>, AGG> backwardFindSessions(final K keyFrom,
                                                                    final K keyTo,
                                                                    final Instant
    earliestSessionEndTime,
                                                                    final Instant
    latestSessionStartTime) {
        throw new UnsupportedOperationException(
            "This API is not supported by this implementation of ReadOnlySessionStore.");
    }
```

```
/**
 * Retrieve all aggregated sessions for the given range of keys. This iterator must be closed
 * after use.
 * <p>
 * For each key, the iterator guarantees ordering of sessions, starting from the oldest/earliest
 * available session to the newest/latest session.
 *
 * @param keyFrom first key in the range to find aggregated session values for
 * @param keyTo   last key in the range to find aggregated session values for
 * @return KeyValueIterator containing all sessions for the provided key, from oldest to newest
 * session.
 * @throws NullPointerException If null is used for any of the keys.
 */
KeyValueIterator<Windowed<K>, AGG> fetch(final K keyFrom, final K keyTo);

/**
 * Retrieve all aggregated sessions for the given range of keys. This iterator must be closed
 * after use.
 * <p>
 * For each key, the iterator guarantees ordering of sessions, starting from the newest/latest
 * available session to the oldest/earliest session.
 *
 * @param keyFrom first key in the range to find aggregated session values for
 * @param keyTo   last key in the range to find aggregated session values for
 * @return backward KeyValueIterator containing all sessions for the provided key, from newest
 * to oldest session.
 * @throws NullPointerException If null is used for any of the keys.
 */
default KeyValueIterator<Windowed<K>, AGG> backwardFetch(final K keyFrom, final K keyTo) {
    throw new UnsupportedOperationException(
        "This API is not supported by this implementation of ReadOnlySessionStore.");
}
```

The requested fetch/findSessions is bounded because both the *keyFrom* and the *keyTo* 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.

Public Interfaces

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 *keyFrom* and *keyTo* would change, thus, requiring a modifications to the JavaDoc method description. The proposed change is shown below:

In the **ReadOnlyWindowStore** looks as follows:

```
/**
 * Get all the key-value pairs in the given key range and time range from all the existing windows.
 * <p>
 * This iterator must be closed after use.
 *
 * @param keyFrom     the first key in the range
 * A null value indicates a starting position from the first element in the store <-- new added
 * @param keyTo the last key in the range
 * A null value indicates that the range ends with the last element in the store. <-- new added
 * @param timeFrom time range start (inclusive), where iteration starts.
 * @param timeTo time range end (inclusive), where iteration ends.
 * @return an iterator over windowed key-value pairs {@code <Windowed<K>, value>}, from beginning
to end of time.
 * @throws InvalidStateStoreException if the store is not initialized
 * @throws IllegalArgumentException if duration is negative or can't be represented as {@code long
milliseconds}
 */
KeyValueIterator<Windowed<K>, V> fetch(K keyFrom, K keyTo, Instant timeFrom, Instant timeTo)
throws IllegalArgumentException;

/**
 * Get all the key-value pairs in the given key range and time range from all the existing windows
 * in backward order with respect to time (from end to beginning of time).
 * <p>
 * This iterator must be closed after use.
 *
 * @param keyFrom the first key in the range
* A null value indicates a starting position from the first element in the store <-- new added
 * @param keyTo the last key in the range
* A null value indicates that the range ends with the last element in the store. <-- new added
 * @param timeFrom time range start (inclusive), where iteration ends.
 * @param timeTo time range end (inclusive), where iteration starts.
 * @return an iterator over windowed key-value pairs {@code <Windowed<K>, value>}, from end to
beginning of time.
 * @throws InvalidStateStoreException if the store is not initialized
 * @throws IllegalArgumentException if duration is negative or can't be represented as {@code long
milliseconds}
 */
default KeyValueIterator<Windowed<K>, V> backwardFetch(K keyFrom, K keyTo, Instant timeFrom,
Instant timeTo)
throws IllegalArgumentException {
throw new UnsupportedOperationException();
}
```

In the **ReadOnlySessionStore** looks as follows:

```
/**
 * Fetch any sessions in the given range of keys and the sessions end is &ge;
 * earliestSessionEndTime and the sessions start is &le; latestSessionStartTime iterating from
 * earliest to latest.
 * <p>
 * This iterator must be closed after use.
 *
 * @param keyFrom               The first key that could be in the range
 * A null value indicates a starting position from the first element in the store <-- new added
 * @param keyTo                 The last key that could be in the range
 * A null value indicates that the range ends with the last element in the store. <-- new added
 * @param earliestSessionEndTime the end timestamp of the earliest session to search for, where
 * iteration starts.
 * @param latestSessionStartTime the end timestamp of the latest session to search for, where
 * iteration ends.
 * @return iterator of sessions with the matching keys and aggregated values, from earliest to
 * latest session time.
 */
default KeyValueIterator<Windowed<K>, AGG> findSessions(final K keyFrom,
final K keyTo,
final long earliestSessionEndTime,
final long latestSessionStartTime) {
throw new UnsupportedOperationException(
"This API is not supported by this implementation of ReadOnlySessionStore.");
}

/**
 * Fetch any sessions in the given range of keys and the sessions end is &ge;
 * earliestSessionEndTime and the sessions start is &le; latestSessionStartTime iterating from
 * earliest to latest.
```

```
 * <p>
 * This iterator must be closed after use.
 *
 * @param keyFrom The first key that could be in the range
* A null value indicates a starting position from the first element in the store <-- new added
 * @param keyTo The last key that could be in the range
* A null value indicates that the range ends with the last element in the store. <-- new added
 * @param earliestSessionEndTime the end timestamp of the earliest session to search for, where
 * iteration starts.
 * @param latestSessionStartTime the end timestamp of the latest session to search for, where
 * iteration ends.
 * @return iterator of sessions with the matching keys and aggregated values, from earliest to
 * latest session time.
 */
default KeyValueIterator<Windowed<K>, AGG> findSessions(final K keyFrom,
final K keyTo,
final Instant earliestSessionEndTime,
final Instant latestSessionStartTime) {
throw new UnsupportedOperationException(
"This API is not supported by this implementation of ReadOnlySessionStore.");
}

/**
 * Fetch any sessions in the given range of keys and the sessions end is &ge;
 * earliestSessionEndTime and the sessions start is &le; latestSessionStartTime iterating from
 * latest to earliest.
 * <p>
 * This iterator must be closed after use.
 *
 * @param keyFrom The first key that could be in the range
* A null value indicates a starting position from the first element in the store <-- new added
 * @param keyTo The last key that could be in the range
* A null value indicates that the range ends with the last element in the store. <-- new added
 * @param earliestSessionEndTime the end timestamp of the earliest session to search for, where
 * iteration ends.
 * @param latestSessionStartTime the end timestamp of the latest session to search for, where
 * iteration starts.
 * @return backward iterator of sessions with the matching keys and aggregated values, from
 * latest to earliest session time.
 */
default KeyValueIterator<Windowed<K>, AGG> backwardFindSessions(final K keyFrom,
final K keyTo,
final long earliestSessionEndTime,
final long latestSessionStartTime) {
throw new UnsupportedOperationException(
"This API is not supported by this implementation of ReadOnlySessionStore.");
}

/**
 * Fetch any sessions in the given range of keys and the sessions end is &ge;
 * earliestSessionEndTime and the sessions start is &le; latestSessionStartTime iterating from
 * latest to earliest.
 * <p>
 * This iterator must be closed after use.
 *
 * @param keyFrom The first key that could be in the range
* A null value indicates a starting position from the first element in the store <-- new added
 * @param keyTo The last key that could be in the range
* A null value indicates that the range ends with the last element in the store. <-- new added
 * @param earliestSessionEndTime the end timestamp of the earliest session to search for, where
 * iteration ends.
 * @param latestSessionStartTime the end timestamp of the latest session to search for, where
 * iteration starts.
 * @return backward iterator of sessions with the matching keys and aggregated values, from
 * latest to earliest session time.
 */
default KeyValueIterator<Windowed<K>, AGG> backwardFindSessions(final K keyFrom,
final K keyTo,
final Instant earliestSessionEndTime,
final Instant latestSessionStartTime) {
throw new UnsupportedOperationException(
"This API is not supported by this implementation of ReadOnlySessionStore.");
}
```

```
/**
 * Retrieve all aggregated sessions for the given range of keys. This iterator must be closed
 * after use.
 * <p>
 * For each key, the iterator guarantees ordering of sessions, starting from the oldest/earliest
 * available session to the newest/latest session.
 *
 * @param keyFrom first key in the range to find aggregated session values for
 * A null value indicates a starting position from the first element in the store <-- new added
 * @param keyTo   last key in the range to find aggregated session values for
 * A null value indicates that the range ends with the last element in the store. <-- new added
 * @return KeyValueIterator containing all sessions for the provided key, from oldest to newest
 * session.
 */
KeyValueIterator<Windowed<K>, AGG> fetch(final K keyFrom, final K keyTo);

/**
 * Retrieve all aggregated sessions for the given range of keys. This iterator must be closed
 * after use.
 * <p>
 * For each key, the iterator guarantees ordering of sessions, starting from the newest/latest
 * available session to the oldest/earliest session.
 *
 * @param keyFrom first key in the range to find aggregated session values for
 * A null value indicates a starting position from the first element in the store <-- new added
 * @param keyTo   last key in the range to find aggregated session values for
 * A null value indicates that the range ends with the last element in the store. <-- new added
 * @return backward KeyValueIterator containing all sessions for the provided key, from newest
 * to oldest session.
 */
default KeyValueIterator<Windowed<K>, AGG> backwardFetch(final K keyFrom, final K keyTo) {
    throw new UnsupportedOperationException(
        "This API is not supported by this implementation of ReadOnlySessionStore.");
}
```

Proposed Changes

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 *keyFrom* and *keyTo* would change.

Note: we didn't change the time parameters because currently user can always fetch/findSessions from/to the earliest/latest time by setting the time as ***0L /Long.MAX_VALUE*** or ***`Instant.ofEpochMilli(0)`/`Instant.ofEpochMilli(Long.MAX_VALUE)`*** to achieve it. So, we focus on the key parameters here.

For example, in **ReadOnlyWindowStore**, the following code snippet illustrates how unbounded range queries can be issued after those changes:

```
1  //return an iterator for all the records with a key greater (or equal) than "key-x"
2  //within time range from all the existing windows. starting with the smallest record
3  KeyValueIterator<String, String> iter = windowStore.fetch("key-x", null, Instant.ofEpochMilli(start_time), In
4  stant.ofEpochMilli(end_time));
5
6  //return an iterator for all the records with a key less (or equal) than "key-x"
7  //within time range from all the existing windows. starting with the smallest record
8  KeyValueIterator<String, String> iter = windowStore.fetch(null, "key-x", Instant.ofEpochMilli(start_time), In
9  stant.ofEpochMilli(end_time));
10
11 //return an iterator for all the records with a key greater (or equal) than "key-x"
   //within time range from all the existing windows. starting with the largest record
12 KeyValueIterator<String, String> iter = windowStore.backwardFetch(null, "key-x", Instant.ofEpochMilli
   (start_time), Instant.ofEpochMilli(end_time));
13
14 //return all the records in the store, equivalent to windowStore.all()
   KeyValueIterator<String, String> iter = windowStore.fetch(null, null);
```

in **ReadOnlySessionStore**, the following code snippet illustrates how unbounded range queries can be issued after those changes:

```
//return an iterator for all the sessions with a key greater (or equal) than "key-x"
//within time range from all the existing windows. starting with the smallest record
KeyValueIterator<String, String> iter = sessionStore.findSessions("key-x", null, Instant.ofEpochMilli
(start_time), Instant.ofEpochMilli(end_time));

KeyValueIterator<String, String> iter = sessionStore.findSessions("key-x", null, start_time_ms, end_time_ms);

//return an iterator for all the sessions with a key less (or equal) than "key-x"
//within time range from all the existing windows. starting with the smallest record
KeyValueIterator<String, String> iter = sessionStore.findSessions(null, "key-x"Instant.ofEpochMilli
(start_time), Instant.ofEpochMilli(end_time));

KeyValueIterator<String, String> iter = sessionStore.findSessions(null, "key-x", start_time_ms, end_time_ms);

//return an iterator for all the sessions with a key greater (or equal) than "key-x"
//within time range from all the existing windows. starting with the largest record
KeyValueIterator<String, String> iter = sessionStore.backwardFindSessions(null, "key-x"Instant.ofEpochMilli
(start_time), Instant.ofEpochMilli(end_time));
```

Compatibility, Deprecation, and Migration Plan

This change is completely backward compatible.

Rejected Alternatives

Extended from KIP-763: Range queries with open endpoints

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.