

KIP-997: update WindowRangeQuery and unify WindowKeyQuery and WindowRangeQuery

- [Status](#)
- [Motivation](#)
- [Proposed Changes](#)
- [Compatibility, Deprecation, and Migration Plan](#)
- [Examples](#)
- [Rejected Alternatives](#)

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: *Under Discussion*

Discussion thread: <https://lists.apache.org/thread/klbjwb0906y1r05dp3wbklowm3bq41zr>

JIRA: [KAFKA-15795](#) - Getting issue details...

STATUS

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

Motivation

We aim to enhance the `WindowRangeQuery` by supporting a new method: `fetch(keyFrom, keyTo, from, to)`. Currently, `WindowRangeQuery` utilizes `KeyValueIterator<Windowed<K>, V> fetchAll(Instant timeFrom, Instant timeTo)` for retrieving all key-value pairs within a specified time range. However, we propose to use `KeyValueIterator<Windowed<K>, V> fetch(K keyFrom, K keyTo, Instant timeFrom, Instant timeTo)` instead. This new method will provide a `KeyValueIterator<Windowed<K>, V>` that allows users to iterate over windowed key-value pairs `{<Windowed<K>, value>}`, spanning the entire time range.

With this new method, users can retrieve window sessions for specific keys, rather than all keys, which enables a more targeted query. This is an improvement over the `fetchAll` method, which only allows retrieval of all key's window sessions without the ability to specify a range of keys.

Additionally, this enhancement also allows the `WindowRangeQuery` to support `WindowKeyQuery` functionality. Users seeking to query window sessions for a specific key can do so by setting `keyFrom` and `keyTo` to be equal. This dual functionality provides more flexibility and efficiency in querying windowed keys.

Proposed Changes

We need to introduce several additional arguments, `upper`, `key`, `newTimeFrom`, `newTimeTo`, in the `WindowRangeQuery` class, and subsequently update the methods within this class to accommodate these new parameters.

We also add several new method with `allKey()`, `fromTime()`, `toTime()`, `withKeyRange()` to this class.

WindowRangeQuery

```
public class WindowRangeQuery<K, V> implements Query<KeyValueIterator<Windowed<K>, V>> {

    // newly added
    public static <K, V> WindowRangeQuery<K, V> withAllKey()

    // newly added
    public WindowRangeQuery<K, V> fromTime(final Instant timeFrom)

    // newly added
    public WindowRangeQuery<K, V> toTime(final Instant timeTo)

    // newly added
    public static <K, V> WindowRangeQuery<K, V> withKeyRange(final K lower, final K upper)

    // @Deprecated
    public Optional<Instant> getOldTimeFrom()

    // @Deprecated
    public Optional<Instant> getOldTimeTo()

    // newly added
    public Optional<Instant> timeFrom()

    // newly added
    public Optional<Instant> timeTo()

    // newly added
    public Optional<K> key()

}
```

Compatibility, Deprecation, and Migration Plan

- Utilizing the existing WindowRangeQuery class, we can make some modifications to realize the concepts of KeyValueIterator<Windowed<K>, V> fetch(K keyFrom, K keyTo, Instant timeFrom, Instant timeTo).
- We want to deprecate WindowKeyQuery class

Examples

The following example illustrates the use of the WindowRangeQuery class to query a kv-store or ts kv-store.

withWindowKeyRange(2, 2, time=2023-01-01T10:05:00.00Z, time=2023-01-01T10:10:00.00Z), the result is (4, 5)

shouldHandleWindowKeyQuery

```
public <V> void shouldHandleWindowRangeQuery(
    final Integer keyFrom,
    final Integer keyTo,
    final Instant timeFrom,
    final Instant timeTo,
    final Function<V, Integer> valueExtractor,
    final Set<Integer> expectedValues) {

    final WindowRangeQuery<Integer, V> query = WindowRangeQuery.withKeyRange(keyFrom, keyTo).fromTime
(timeFrom).toTime(timeTo);

    final StateQueryRequest<KeyValueIterator<Windowed<Integer>, V>> request =
        inStore(STORE_NAME)
            .withQuery(query)
            .withPartitions(mkSet(0, 1))
            .withPositionBound(PositionBound.at(INPUT_POSITION));

    final StateQueryResult<KeyValueIterator<Windowed<Integer>, V>> result =
        IntegrationTestUtils.iqv2WaitForResult(kafkaStreams, request);

    if (result.getGlobalResult() != null) {
        fail("global tables aren't implemented");
    } else {
        final Set<Integer> actualValues = new HashSet<>();
        final Map<Integer, QueryResult<KeyValueIterator<Windowed<Integer>, V>>> queryResult = result.
getPartitionResults();
        for (final int partition : queryResult.keySet()) {
            final boolean failure = queryResult.get(partition).isFailure();
            if (failure) {
                throw new AssertionError(queryResult.toString());
            }
            assertThat(queryResult.get(partition).isSuccess(), is(true));

            assertThrows(
                IllegalArgumentException.class,
                queryResult.get(partition)::getFailureReason
            );
            assertThrows(
                IllegalArgumentException.class,
                queryResult.get(partition)::getFailureMessage
            );

            try (final KeyValueIterator<Windowed<Integer>, V> iterator = queryResult.get(partition).
getResult()) {
                while (iterator.hasNext()) {
                    actualValues.add(valueExtractor.apply(iterator.next().value));
                }
                assertThat(queryResult.get(partition).getExecutionInfo(), is(empty()));
            }
            assertThat("Result:" + result, actualValues, is(expectedValues));
            assertThat("Result:" + result, result.getPosition(), is(INPUT_POSITION));
        }
    }
}
```

Test Plan

With the introduction of the new `keyTo` argument to the `WindowRangeQuery` class, it is necessary to update all associated tests. Since `SessionRangeQuery` and `WindowKeyQuery` also utilize methods from `WindowRangeQuery`, we must ensure that the tests covering `SessionRangeQuery` and `WindowKeyQuery` are revised accordingly.

Rejected Alternatives

We will introduce a new class, `WindowRangeQuery2`, designed to facilitate the `KeyValueIterator<Windowed<K>, V> fetch(K keyFrom, K keyTo, Instant timeFrom, Instant timeTo)` method. This addition will be an expansion to the `IQv2` codebase, ensuring that no existing code is modified; we are merely supplementing with new code and methods to support this enhanced query type.