KIP-805: Add range and scan query over kv-store in IQv2

Status

Current state: *Adopted*

Discussion thread: https://lists.apache.org/thread/4clhz43yy9nk6kkggbcn0y3v61b05sp1

Voting thread: https://lists.apache.org/thread/fh9gnhk9zoqlt3fy883hwjwh47qjj2c5

JIRA: ⚠ Unable to render Jira issues macro, execution error.

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

Motivation

Provide an implementation of the `Query` interface, introduced in KIP-796: Interactive Query v2 , to support range and scan queries

Proposed Changes

The `RangeQuery` class will be used for both range and scan queries. A scan is performed when no lower and no upper bound is specified. A range query retrieves a set of keys, specified using an upper and/or lower bound, from the underlying KV store. A scan, on the other hand, retrieves all keys contained in the KV store.

RangeQuery.java

```
@Evolving
public class RangeQuery<K, V> implements Query<KeyValueIterator<K, V>> {
    private final Optional<K> lower;
    private final Optional<K> upper;

    private RangeQuery(final Optional<K> lower, final Optional<K> upper) {
        this.lower = lower;
        this.upper = upper;
    }

    public static <K, V> RangeQuery<K, V> withRange(final K lower, final K upper) {
        return new RangeQuery<>(Optional.of(lower), Optional.of(upper));
    }

    public static <K, V> RangeQuery<K, V> withUpperBound(final K upper) {
        return new RangeQuery<>(Optional.empty(), Optional.of(upper));
    }

    public static <K, V> RangeQuery<K, V> withLowerBound(final K lower) {
        return new RangeQuery<>(Optional.of(lower), Optional.empty());
    }

    public static <K, V> RangeQuery<K, V> withNoBounds() {
        return new RangeQuery<>(Optional.empty(), Optional.empty());
    }
```

```
    public Optional<K> getLowerBound() {
        return lower;
    }

    public Optional<K> getUpperBound() {
        return upper;
    }
}

// =====================================
// Range query example usage in IQv2:

Integer key1 = 1;
Integer key2 = 2;

// create the query parameters
final StateSerdes<Integer, ValueAndTimestamp<Integer>> serdes =
      kafkaStreams.serdesForStore("mystore")

StateQueryRequest<KeyValueIterator<Integer, Integer>> query =
  inStore("mystore")
  .withQuery(RangeQuery.withRange(key1, key2));

// run the query
StateQueryResult<KeyValueIterator<Integer, Integer>> result = kafkaStreams.query(query);

// Get the results from all partitions.
        final Map<Integer, QueryResult<KeyValueIterator<Integer, Integer>>> partitionResults =
            rangeResult.getPartitionResults();
        for (final Entry<Integer, QueryResult<KeyValueIterator<Integer, Integer>>> entry : partitionResults.
entrySet()) {
            try (final KeyValueIterator<Integer, Integer> keyValueIterator = entry.getValue().getResult()) {
                while (keyValueIterator.hasNext()) {
                    final KeyValue<Integer, Integer> next = keyValueIterator.next();
                    Integer key = next.key.get;
                                        Integer value = next.value;
                }
            }
        }


// =====================================
// Scan query example usage in IQv2:

// create the query parameters
StateQueryRequest<KeyValueIterator<Integer, Integer>> query =
  inStore("mystore")
  .withQuery(RangeQuery.withNoBounds());

// run the query
StateQueryResult<KeyValueIterator<Integer, Integer>> result = kafkaStreams.query(query);
```

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

Initially, we proposed to add also a `RawRangeQuery` typed with `<KeyValueIterator<Bytes, byte[]>` . After looking at the code, it seems that it doesn't provide us with many benefits (we save on one cast) which doesn't justify the cost of adding an extra query to the public interface.