KIP-718: Make KTable Join on Foreign key unopinionated

Status

Current state: *"Voting in progress"*

Discussion thread: *here*

JIRA: *here*

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

Motivation

The current implementation of KIP-213 for Foreign Key Join between two KTables is *opinionated* in terms of intermediate (subscription) store persistence type.

Independently of the Materialization strategy provided in the method argument, it generates an intermediary RocksDB state store. Thus, even when the Materialization method provided is "in-memory", it will use RocksDB under-the-hood for this internal subscription-store.

A few problems that this behaviour causes:

- **IT Tests**: Having an implicit materialization method for state-store affects tests using foreign key state-stores. Windows based systems suffer from the bug described here. The bug is caused by the RocksDB filesystem failing on cleanup step. A work-around to avoid the bug is to use in-memory state-stores in integration tests (rather than exception swallowing). Having the intermediate RocksDB storage being created disregarding Materialization method forces any IT test to necessarily use the manual FS deletion with exception swallowing hack.
- **Short lived stream applications**: KTables (or the application running Kafka streams) can be short lived in a way that neither persistent storage nor change-logs creation are desired. The current implementation prevents this kind of behaviour.
- **Breach on dependency inversion design**: Forcing RocksDB intermediate storage seems to go against one of the goals of Kafka Stream on being agnostic of the persistence method - Kafka stream currently does provide just a few out-of-the-box persistence methods (e.g. RocksDB/in-memory) but seems to be designed to support multiple instead of necessarily favouring a given one - it seems that the KeyValueStore interface is declared exactly because of Interface Segregation Principle.
- **Lack of fine tuning capabilities:** Due to distinct access patterns between a subscription-store and a table-store, a user may want to apply different implementations or configurations to it.

Public Interfaces

The following public interface will be changed, adding 4 methods with extra arguments for join/left join on foreign key and deprecating the equivalent method calls with a single materialization option:

- org/apache/kafka/streams/kstream/KTable

Proposed Changes

As discussed in the mail list the change is to add new feature/API methods and deprecate old methods.

New feature:

The new feature proposal consists on **adding** the following methods on KTable interface (and implementing them in KtableImpl.java):

```
<VR, KO, VO> KTable<K, VR> leftJoin(final KTable<KO, VO> other,
                                    final Function<V, KO> foreignKeyExtractor,
                                    final ValueJoiner<V, VO, VR> joiner,
                                    final Named named,
```

final Materialized<K, VR, KeyValueStore<Bytes, byte[]>> materialized,

final Materialized<KO, V, KeyValueStore<Bytes,byte[]> subscriptionStoreMaterialization);

<VR, KO, VO> KTable<K, VR> leftJoin(final KTable<KO, VO> other,

final Function<V, KO> foreignKeyExtractor,

final ValueJoiner<V, VO, VR> joiner,

final Materialized<K, VR, KeyValueStore<Bytes, byte[]>> materialized,

final Materialized<KO, V, KeyValueStore<Bytes,byte[]> subscriptionStoreMaterialization);

<VR, KO, VO> KTable<K, VR> join(final KTable<KO, VO> other,

final Function<V, KO> foreignKeyExtractor,

final ValueJoiner<V, VO, VR> joiner,

final Named named,

final Materialized<K, VR, KeyValueStore<Bytes, byte[]>> materialized,

final Materialized<KO, V, KeyValueStore<Bytes,byte[]> subscriptionStoreMaterialization);

<VR, KO, VO> KTable<K, VR> join(final KTable<KO, VO> other,

final Function<V, KO> foreignKeyExtractor,

final ValueJoiner<V, VO, VR> joiner,

final Materialized<K, VR, KeyValueStore<Bytes, byte[]>> materialized,

final Materialized<KO, V, KeyValueStore<Bytes,byte[]> subscriptionStoreMaterialization);

As discussed in the mail-list, the reason to allow this kind of fine-grain tuning are:

1. **The number or size of records is very large, but the join cardinality is low:** In this scenario, some users may prefer to have an on-disk table-store with an in-memory subscription-store.
2. **The number or size of records in each partition of both tables is small(ish):** In this scenario, a user may prefer to have an in-memory subscription-store. The cardinality of a subscription-store keys is the cardinality of FK from the specified input table. In this case, even if the cardinality is high (on an all distinct 1-1 mapping) the number of entries in a subscription-store will still be small. One row will always map to only one Foreign Key.
3. **The user might want a different type (or differently configured) store for the subscription-store:** This would be for users fine-tuning the access pattern of the store.

As previously discussed in the mail list:

- Some fields from Materialized are not used by subscription-stores. These fields are Retention and Key/Value Serdes. The underlying store type from a subscription-store is "KeyValueStore<Bytes,byte[]>".
- Even though this fields are not used, in order to make the API surface smaller, it was decided to use Materialized class as a parameter for subscription-stores. Information about the field not being used in the case of subscription-stores is going to be added as part of the documentation.
- Documentation about subscription stores needs to be added to Kafka-streams docs.

API methods deprecation

The following method calls will be **deprecated**, since a user that is providing the materialization type of a table-store will likely also be deciding on the materialization type of a subscriptions-store

<VR, KO, VO> KTable<K, VR> leftJoin(final KTable<KO, VO> other,

final Function<V, KO> foreignKeyExtractor,

final ValueJoiner<V, VO, VR> joiner,

final Named named,

final Materialized<K, VR, KeyValueStore<Bytes, byte[]>> materialized);

```
<VR, KO, VO> KTable<K, VR> leftJoin(final KTable<KO, VO> other,
                                    final Function<V, KO> foreignKeyExtractor,
                                    final ValueJoiner<V, VO, VR> joiner,
                                    final Materialized<K, VR, KeyValueStore<Bytes, byte[]>> materialized);


<VR, KO, VO> KTable<K, VR> join(final KTable<KO, VO> other,
                                final Function<V, KO> foreignKeyExtractor,
                                final ValueJoiner<V, VO, VR> joiner,
                                final Named named,
                                final Materialized<K, VR, KeyValueStore<Bytes, byte[]>> materialized);


<VR, KO, VO> KTable<K, VR> join(final KTable<KO, VO> other,
                                final Function<V, KO> foreignKeyExtractor,
                                final ValueJoiner<V, VO, VR> joiner,
                                final Materialized<K, VR, KeyValueStore<Bytes, byte[]>> materialized);
```

Compatibility, Deprecation, and Migration Plan

Four API methods will be deprecated and replaced for four new equivalent methods.

After the deprecation period expires, to use newer versions of kafka-streams, users who use those deprecated methods need to update their code-base.

In the scenario where users of the old API method need to fine-tune the materialization of the subscription store, they will have to update their code-base to the new API instead. This will keep the behaviour for users of the old API consistent until deprecation.

Rejected Alternatives

- Only piggybacking on the Materialization provided by the materialized parameter. It would limit the desired customisation possibilities to the user.
- Creating a specific class for subscription-stores Materialization - it would increase the API footprint without bringing and code complexity without a good trade-off.
- Not deprecating the previous API. As agreed on the discussion, a user that is concerned about the materialization method of a table-store is likely concerned with the materialization method of subscription store.
- Applying Bug-fix to the old API. Since the old API call is being deprecated, users that would be interested on avoiding the opinionated storage should instead use the new API.