OTUS

Онлайн образование

otus.ru

Проверить, идет ли запись

Меня хорошо видно && слышно?

Тема вебинара

Kafka Streams

Непомнящий Евгений

Разработчик Java/Kotlin IT-Sense

@evgeniyN

Правила вебинара

Активно
участвуем

Off-topic обсуждаем
в TG

Задаем вопрос
в чат или **голосом**

Вопросы вижу в чате,
могу ответить не сразу

Условные обозначения

Индивидуально

Время, необходимое
на активность

Пишем в чат

Говорим голосом

Документ

Ответьте себе или
задайте вопрос

Маршрут вебинара

- Введение в проблему
- KStream
- KTable

Цели вебинара

После занятия вы сможете

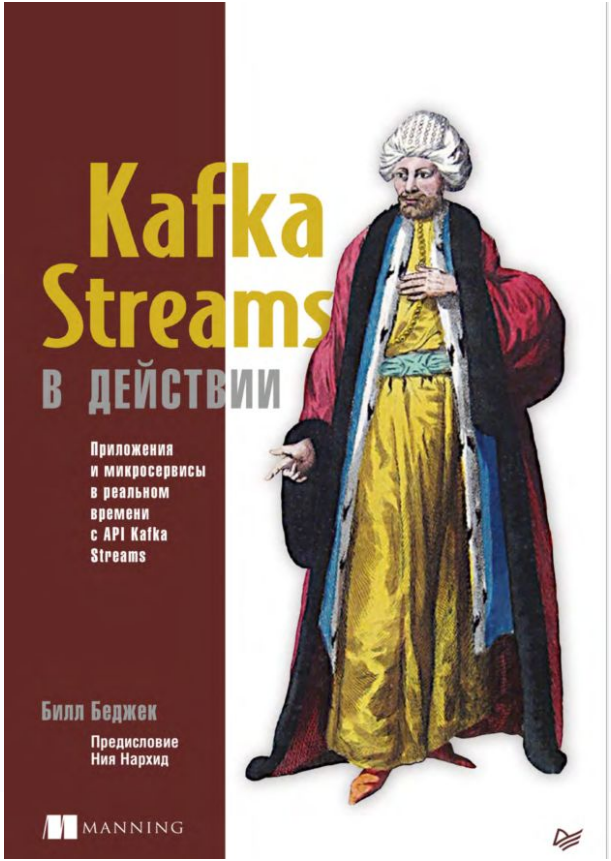
1. Рассмотреть Kafka Streams
2. Разработать потоковое приложение

Смысл

Зачем вам это уметь

1. Kafka Streams позволяет избавиться от ручной работы в некоторых ситуациях

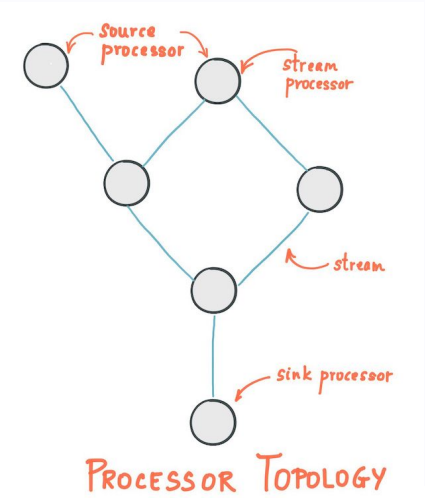
Литература

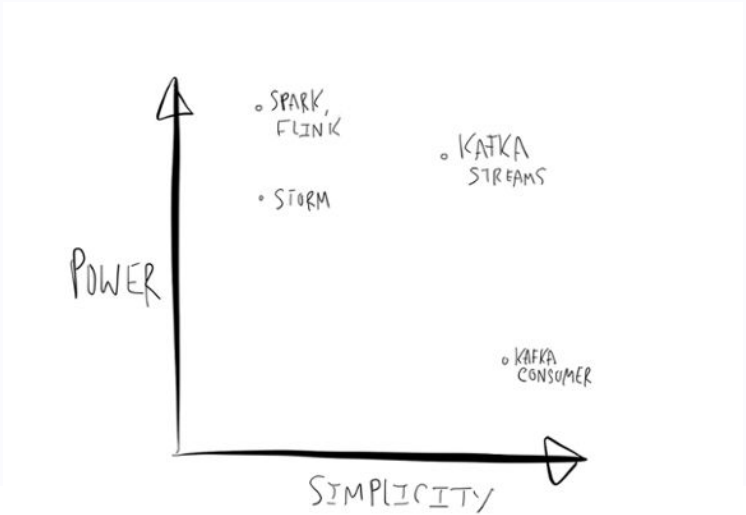

Kafka Streams
в действии
Приложения и микросервисы в реальном времени с API Kafka Streams
Билл Беджек
Предисловие Ния Нархид
MANNING

Введение и простой пример

Kafka Streams

Kafka Streams - это про потоковую обработку событий. Вы получаете событие из одного или нескольких топиков, что-то с ними делаете и кладете в какие-то другие топики.
Kafka Streams позволяет работать не только с цепочкой обработчиков, а с DAG (направленный ациклический граф)





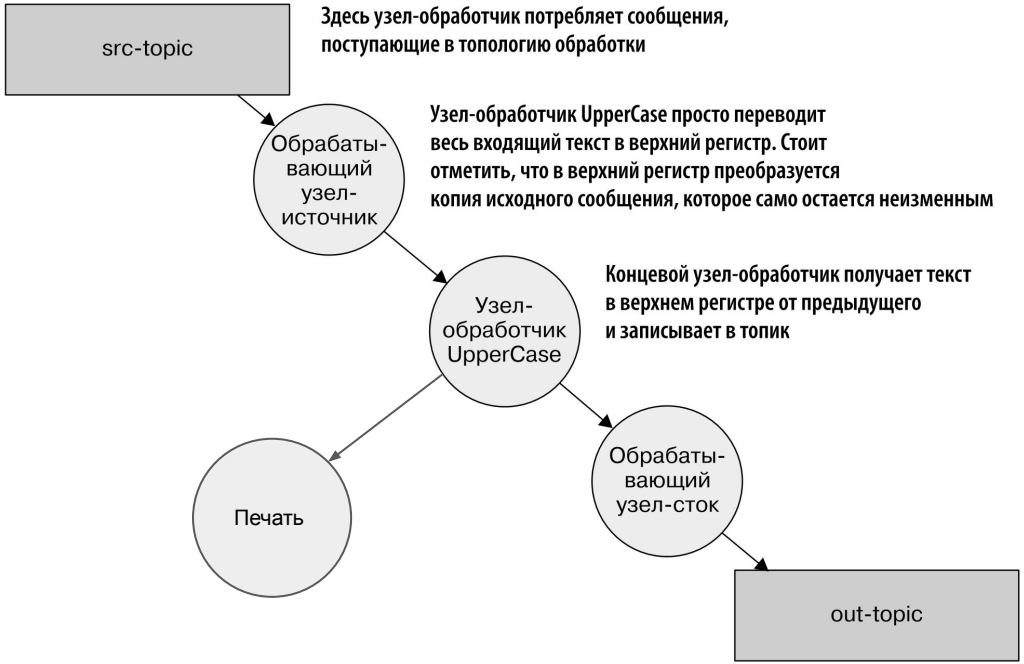
Kafka API

- Producer API - kafka-clients
- Consumer API - kafka-clients
- Streams API - kafka-streams
- Admin API - kafka-clients
- Connect API

Kafka Streams - это библиотека, которую вы используете в своем приложении. Это более высокий уровень по сравнению с ручным Producer / Consumer. Однако внутри там нет никакой “магии”, она основана на работе с Producer / Consumer.

Вводный пример

Ex1UpperCaseTransformer



Serde

Это пара из Serializer / Deserializer.
Serdes дает набор “стандартных” пар - для чисел, строк и т.п.

Более реальный пример

Обработка продажи товара

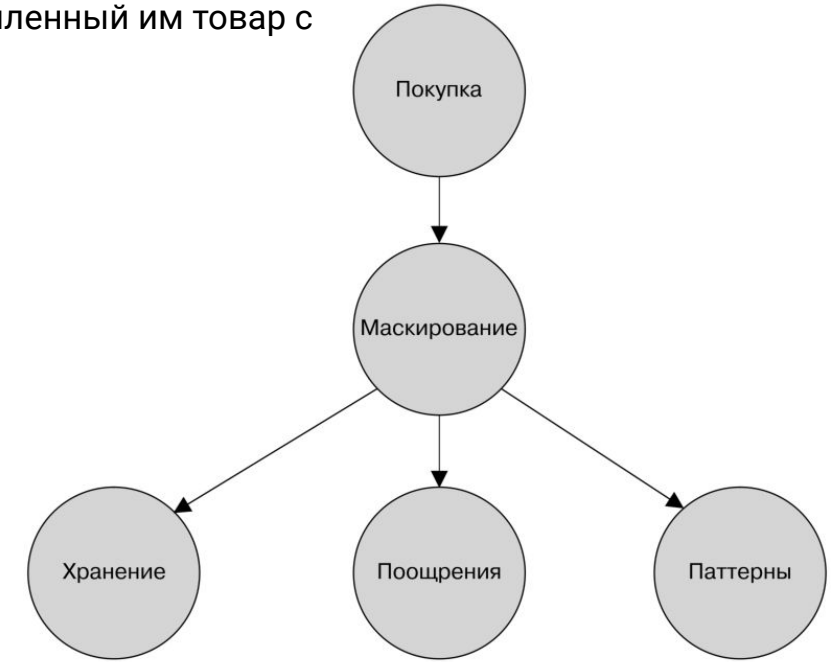
Клиент заходит в магазин и делает покупку. При этом он предъявляет дисконтную карту.
В результате возникает событие “Продажа” на каждый купленный им товар с кредитной картой, дисконтной картой и т.п.

Надо

- замаскировать код кредитной карты (`mapValues`)
- начислить бонусы за покупку (`mapValues, to`)
- отследить факт продажи товара (`mapValues, to`)
- сохранить замаскированную транзакцию для последующей обработки (`to`)

Еще используем: `print, peek`

Ex2PurchaseSimple

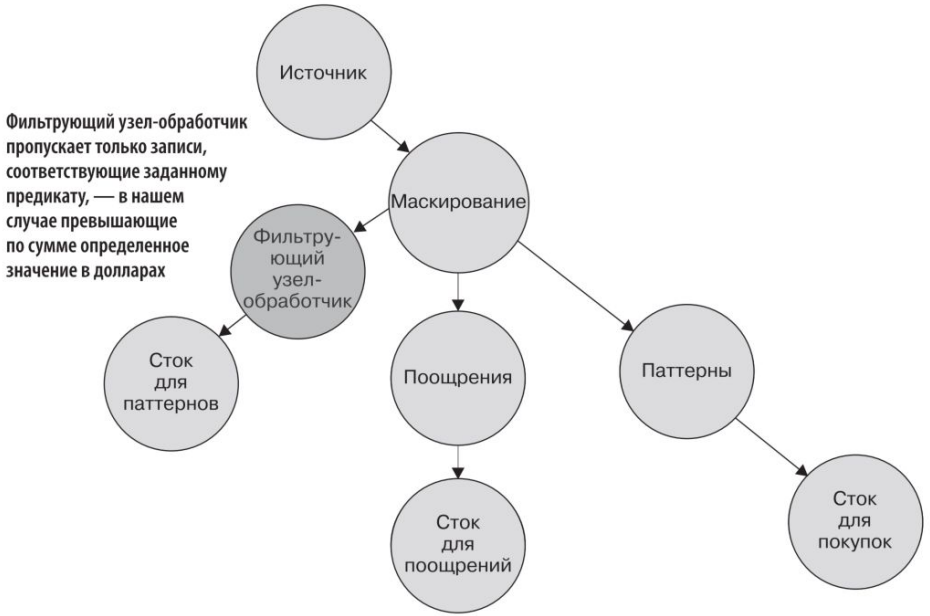


Фильтрация покупок

Предположим нам надо хранить только дорогие покупки.

- `filter`
- selectKey

Ex3Purchase (** 1)

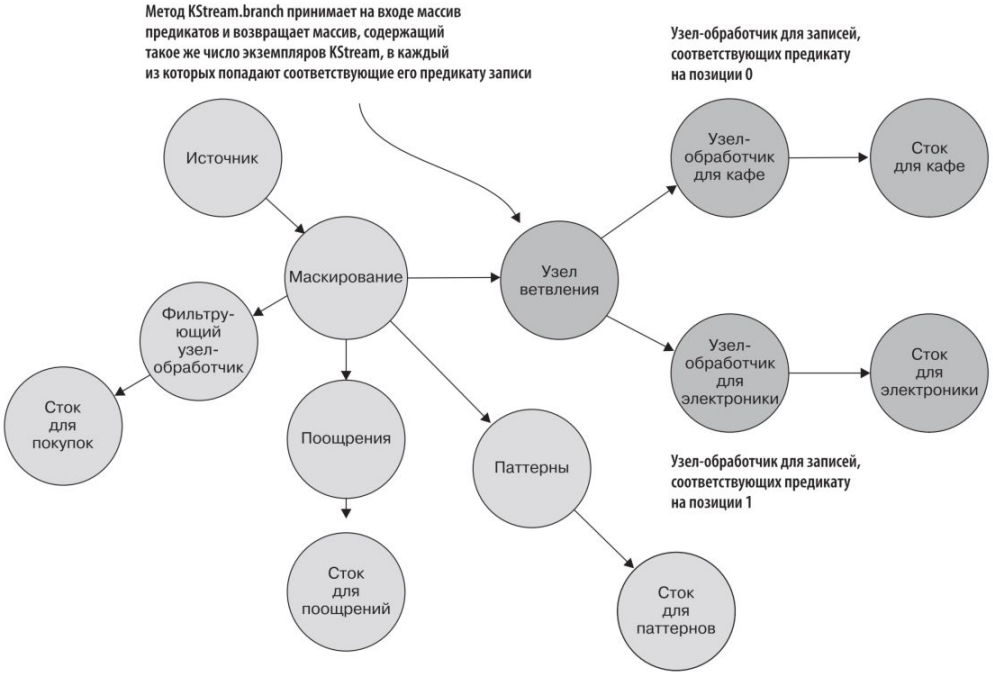


Копирование в разные топики

Предположим, мы хотим выделить продажи в кафе и продажи электроники отдельно.

- `split (branch)`

Ex3Purchase (** 2)

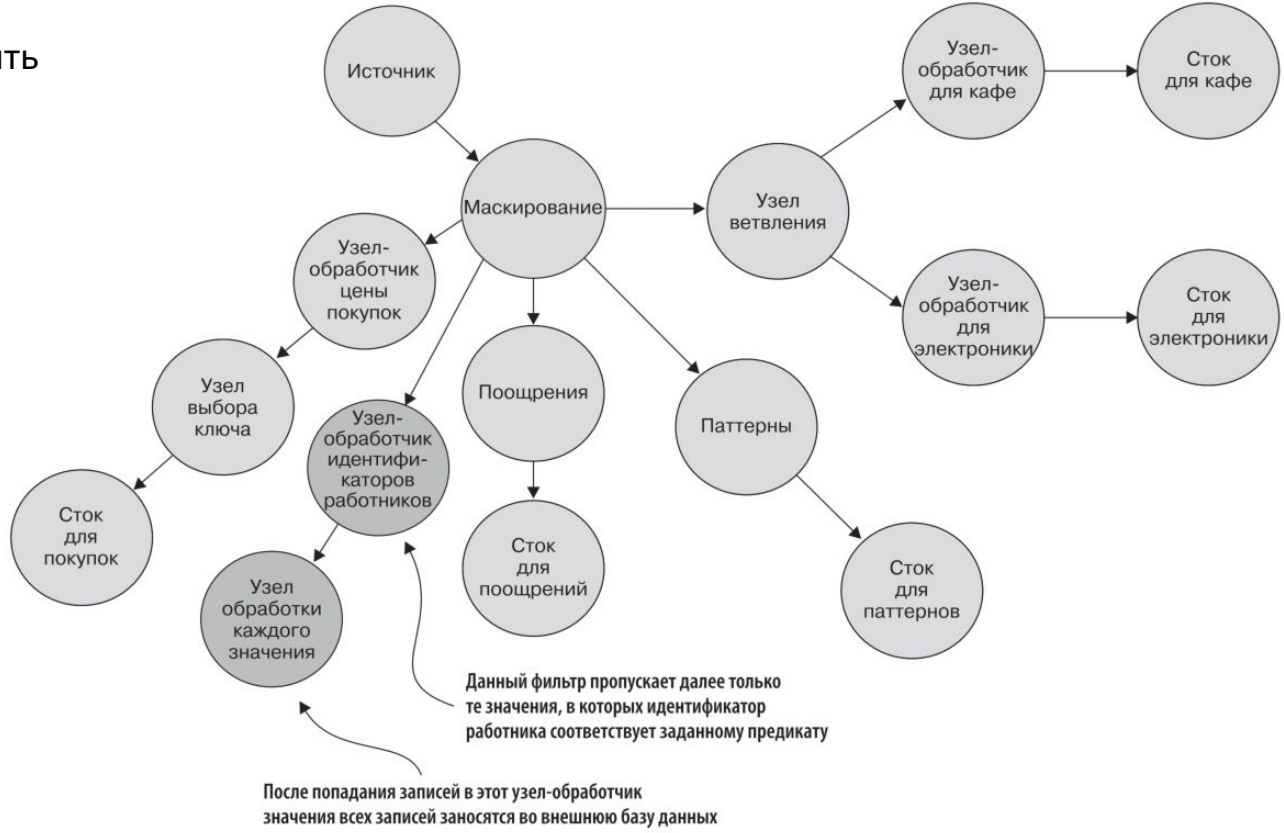


Сохранение записи во внешнее хранилище

Предположим, мы хотим для конкретного продавца сохранять его продажи отдельно в БД.

- `foreach`

Ex3Purchase (** 3)



Разные методы KStream

- filter, filterNot
- map, mapValues
- flatMap, flatMapValues
- print
- foreach
- peek
- split
- merge - объединение с другим стримом

Хранение состояния

Состояние

До сих пор мы не хранили состояние обработки.
Просто каждое событие последовательно обрабатывалось.

Когда может потребоваться состояние? Например мы накапливаем сумму трат, чтобы иметь пороговые накопительные скидки. Или мы сливаем (join) вместе два потока.

В чем тут проблема? У нас может быть несколько инстансов приложения - как тут будет работать состояние? Как оно будет шариться между экземплярами?

Накапливание бонусов

Давайте при начислении бонусов указывать еще и сумму накопленного (totalRewardPoints)

Т.е. нам нужно как-то хранить кол-во накопленных бонусов по кастомеру.

- `process, processValues`
- `StreamBuilder.addStateStore`
- `Stores.keyValueStoreBuilder`

Для проверки - в консоли ищем BonusByCustomer, а потом пару кастомеров в топике reward-topic

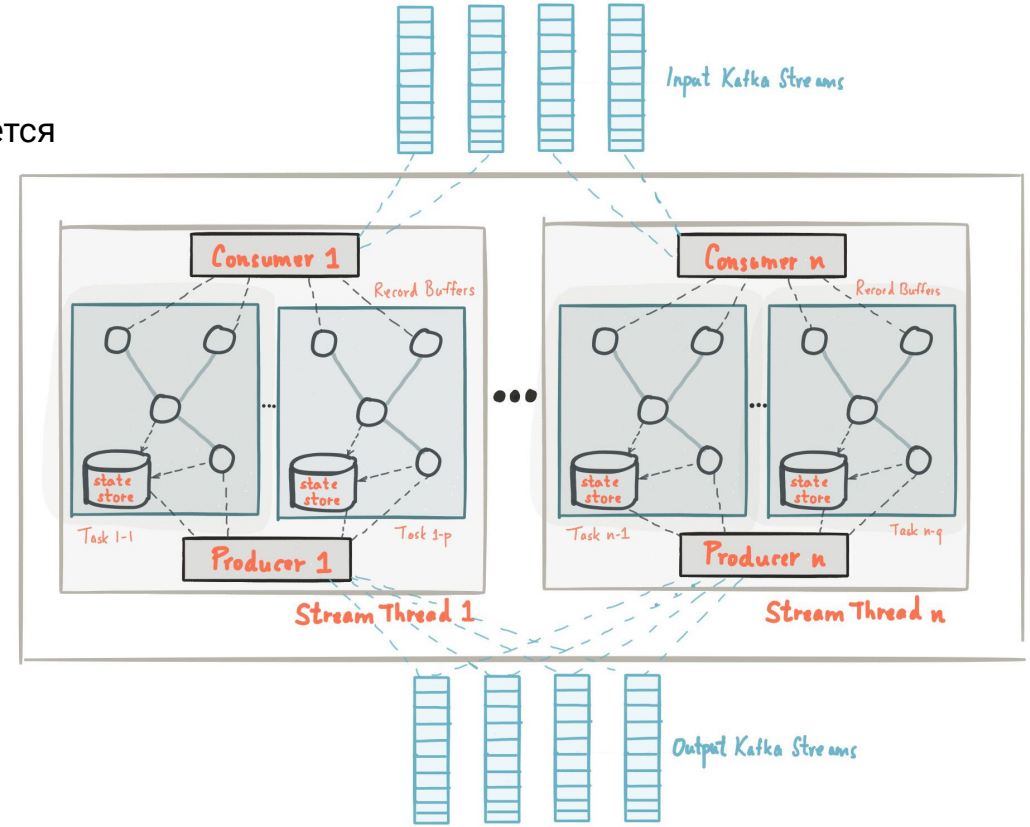
А еще хранилище дублируется в кафку - см ex4-reward-store-name-changelog

Ex4Reward

А что если сделать 2 партиции?

Ex4Reward (**1)
Почему оно сломалось?

На каждую партицию создается свой Task.
Каждый Task имеет свое хранилище (которое мапится на соотв. партицию служебного топика).
Это означает, что то, что участвует в вычислении состояния, должно быть в одной партиции



Но у нас нет ключей!

У нас во входных данных нет ключей, поэтому они распределяются между партициями рандомно.
Даже если бы ключи были, не факт, что они были бы нужными (customerId).

В таком случае нам поможет репартиционирование - перекладка сообщений в служебный топик в нужные нам партиции, а потом чтение оттуда.

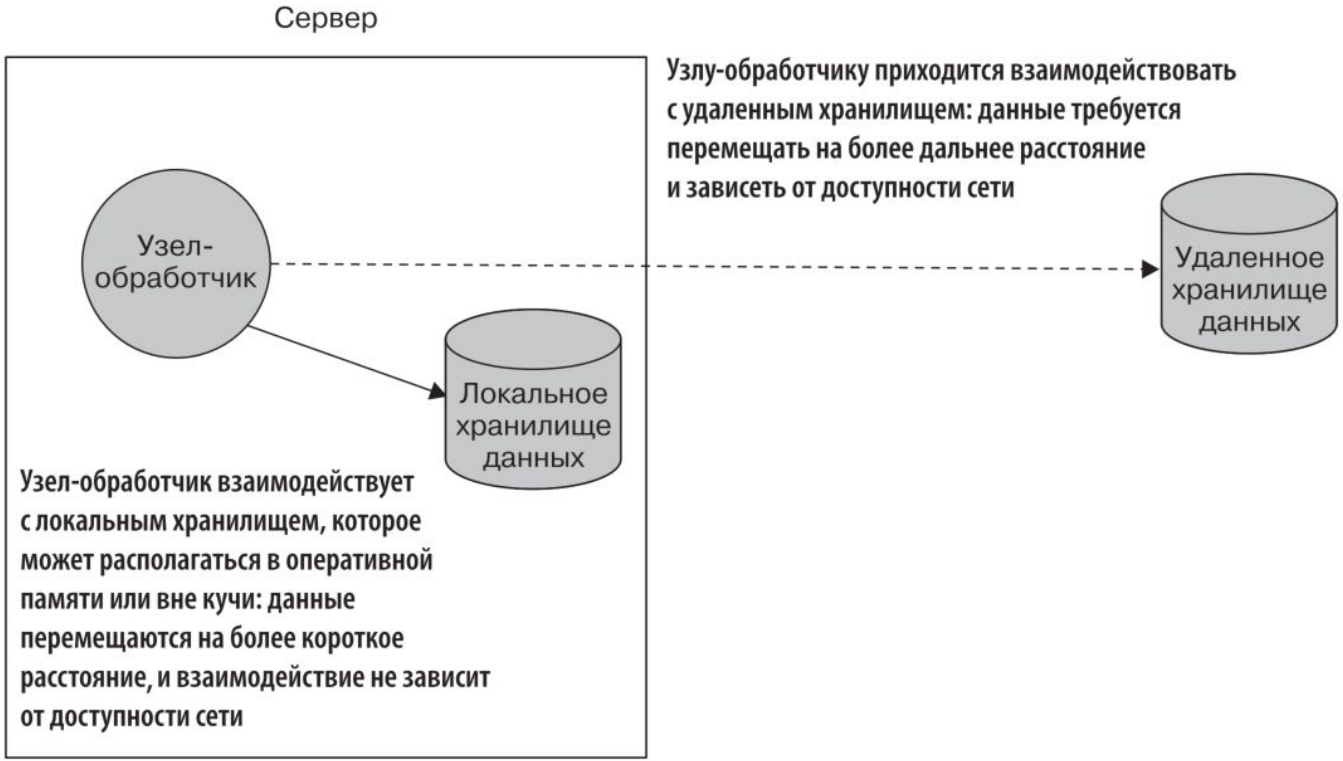
- `repartition`
- `StreamPartitioner`

Ex4Reward (**2)

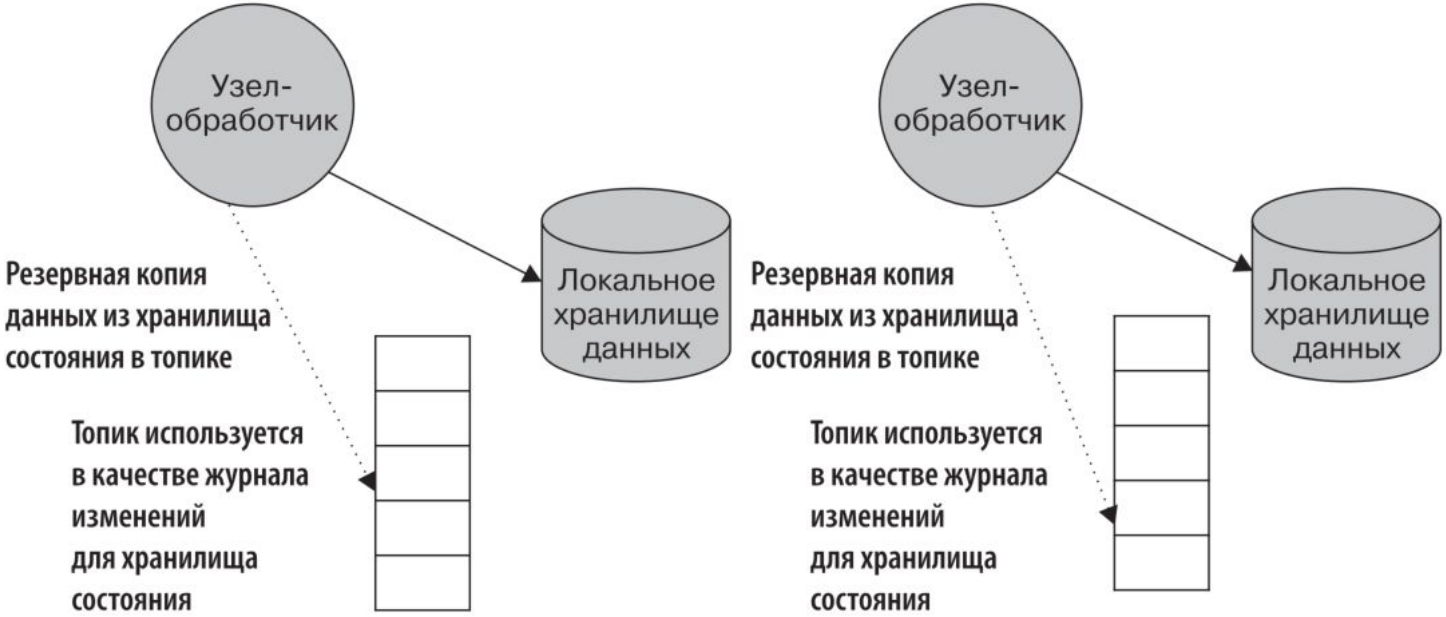
Требования к хранилищу

- Локальность данных
- Восстановление после сбоя

Локальность данных



Отказоустойчивость



В силу наличия у каждого из процессов своего собственного хранилища состояния и архитектуры, не предусматривающей разделения ресурсов, никакого влияния на второй процесс не оказывается. Кроме того, ключи/значения каждого из топиков реплицируются в топик, используемый для восстановления значений, утраченных при сбое или перезапуске процесса

Создание хранилища

```
String rewardsStateStoreName = "rewardsPointsStore";
KeyValueBytesStoreSupplier storeSupplier =
➥ Stores.inMemoryKeyValueStore(rewardsStateStoreName);

StoreBuilder<KeyValueStore<String, Integer>> storeBuilder =
➥ Stores.keyValueStoreBuilder(storeSupplier,
                               Serdes.String(),
                               Serdes.Integer());

builder.addStateStore(storeBuilder);
```

Создаем объект — поставщик StateStore

Создаем объект StoreBuilder и задаем типы ключа и значения

Добавляем хранилище состояния в топологию

Виды хранилищ

- Stores.persistentKeyValueStore - под капотом использует https://rocksdb.org/ (сохраняется на диск)
- Stores.inMemoryKeyValueStore
- Stores.lruMap

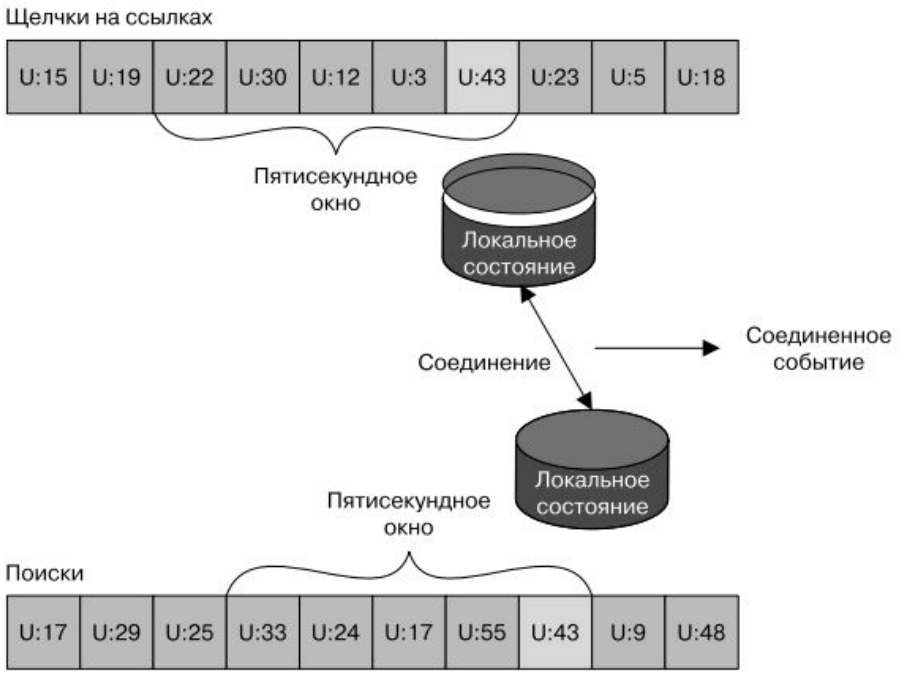
Настройки журналирования

- По умолчанию включено
- Можно выключить - `StoreBuilder.withLoggingDisabled()`
- Можно настроить параметры топика (который библиотека создает сама) - `StoreBuilder.withLoggingEnabled(Map<String, String> config)`
    - См `TopicConfig`

Объединение данных

Join

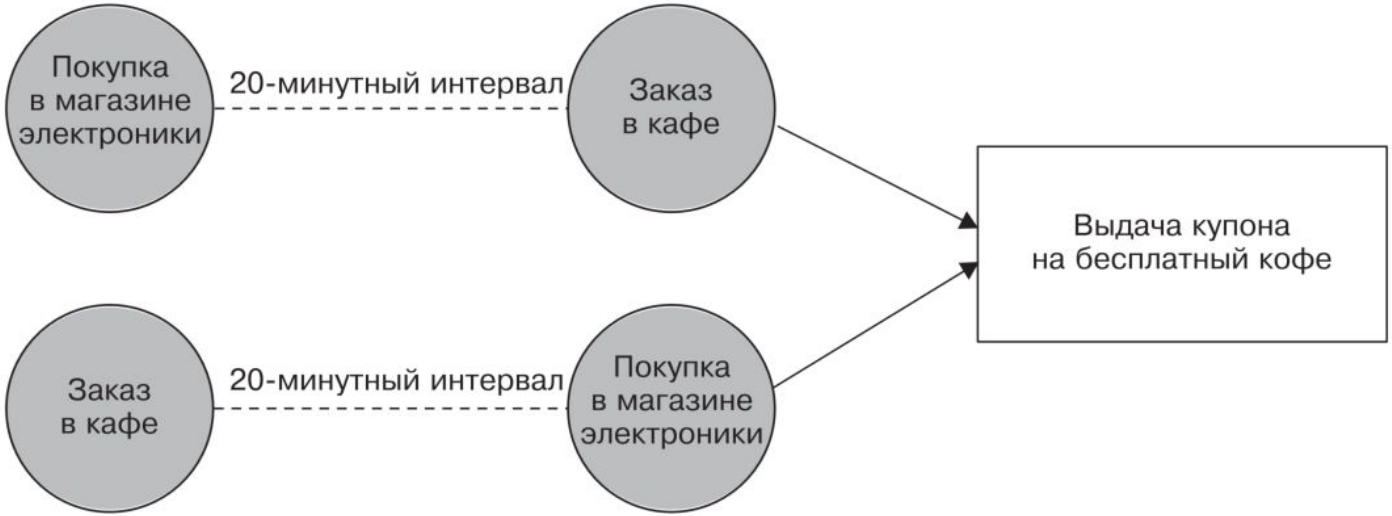
Похож на JOIN из SQL, но так как у нас не таблица, а бесконечный поток, JOIN выполняется в рамках временного окна (а не как в SQL в рамках всего содержимого таблицы)



Задача

Давайте выдавать купон покупателю, который купил и кофе и электронику в одно время (помните, у нас есть два топика для этих целей?)

Ex5Join



Join

- `join` - только совпадающие по ключам из обоих потоков
- `leftJoin` - все из первого + совпадающие по ключам из второго
- `outerJoin` - все из всех, а совпадающие по ключам сливаются

```
interface ValueJoiner<V1, V2, VR> {
    VR apply(final V1 value1, final V2 value2);
}

static JoinWindows ofTimeDifferenceAndGrace(Duration timeDifference,
Duration afterWindowEnd)
```

Ex6Join

Время в стриме

1. Из события
    a. из таймстампа
        i. можно устанавливать вручную при отправке
        ii. может устанавливать библиотека
        iii. может устанавливать брокер
    b. из содержимого
2. Из текущего локального времени в момент обработки

`TimestampExtractor` - можно указать при создании Consumed. По умолчанию берется из таймстампа.

- `ExtractRecordMetadataTimestamp` (`FailOnInvalidTimestamp`, `LogAndSkipOnInvalidTimestamp`, `UsePreviousTimeOnInvalidTimestamp`)
- `WallclockTimestampExtractor`

Автоматическое репартиционирование

При каждом вызове в Kafka Streams метода, в результате выполнения которого может быть сгенерирован новый ключ (selectKey, map или transform), значение специального внутреннего булева флага устанавливается в true, указывая, что этот новый экземпляр KStream требует повторного секционирования.
А при таком значении указанного флага Kafka Streams автоматически осуществляет вместо вас повторное секционирование при выполнении вами любой операции соединения, свертки или агрегирования.

Ex5Join (**2)

KTable

Связь таблицы и потока

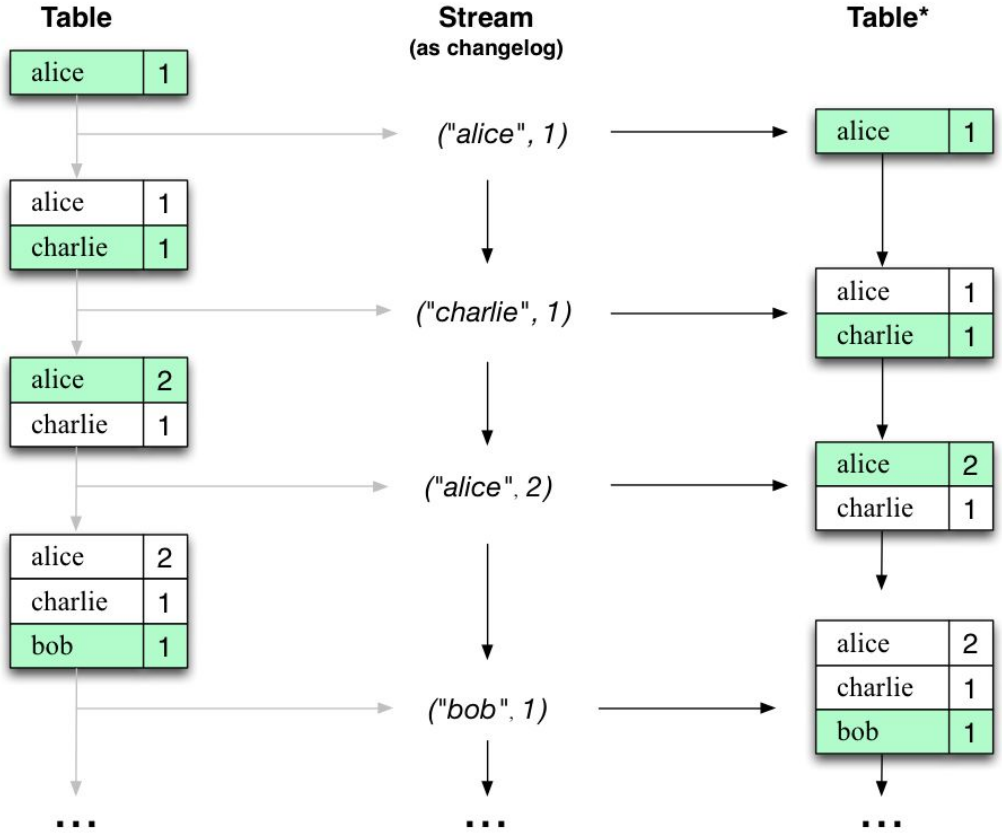
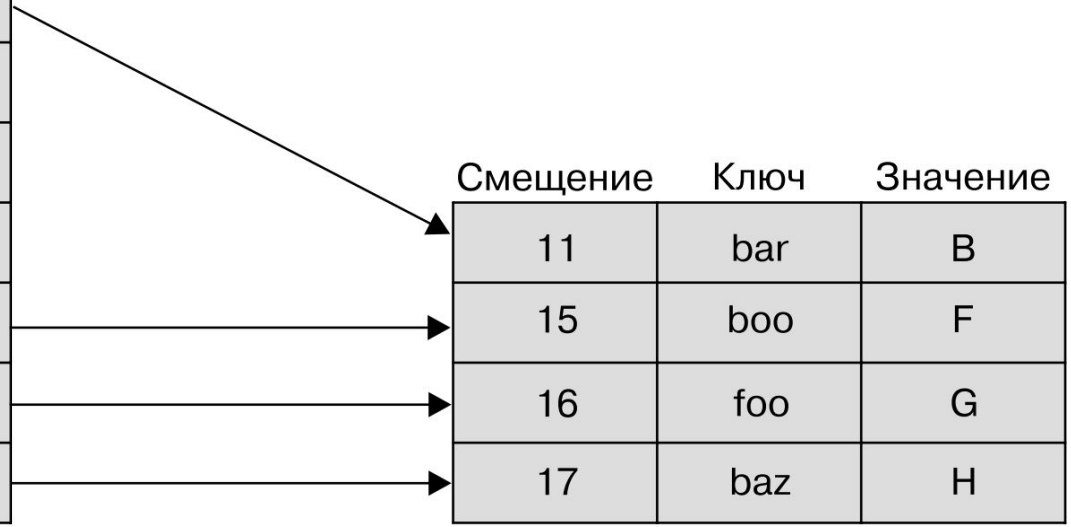


Таблица как сжатый топик

Смещение	Ключ	Значение
10	foo	A
11	bar	B
12	baz	C
13	foo	D
14	baz	E
15	boo	F
16	foo	G
17	baz	H

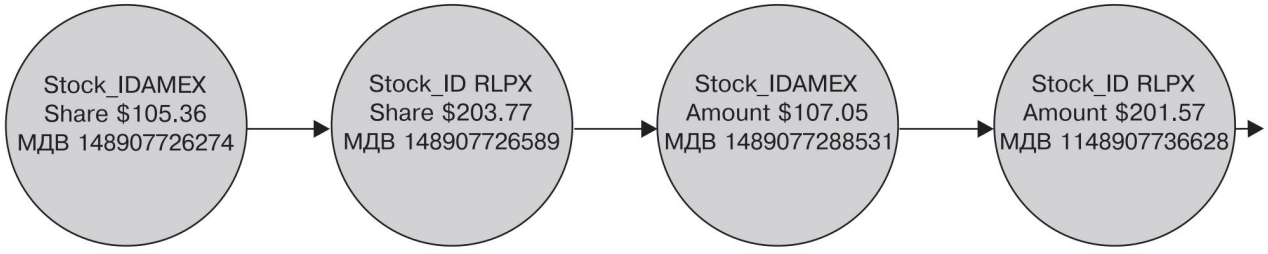


Смещение	Ключ	Значение
11	bar	B
15	boo	F
16	foo	G
17	baz	H

Задача

Дальше мы будем рассматривать пример котировок (цен на акции)

Ex7TableVsStream



Stock_ID	Метка даты/времени	Share_Price
AMEX	148907726274	105.36
RLPX	148907726589	203.77
AMEX	1489077288531	107.05
RLPX	148907736628	201.57

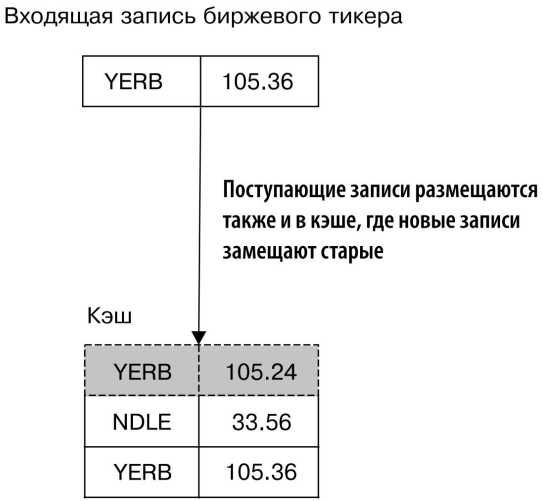
Предыдущие записи для этих акций были заменены обновлениями

Последние записи из потока событий

KTable

- Хранит состояние в локальном хранилище
- Отправляет состояние дальше по потоку в некоторых ситуациях
  - Чем больше входной поток - тем чаще
  - Чем больше уникальных ключей - тем чаще
  - cache.max.bytes.buffering - размер кэша
  - commit.interval.ms - частота фиксации

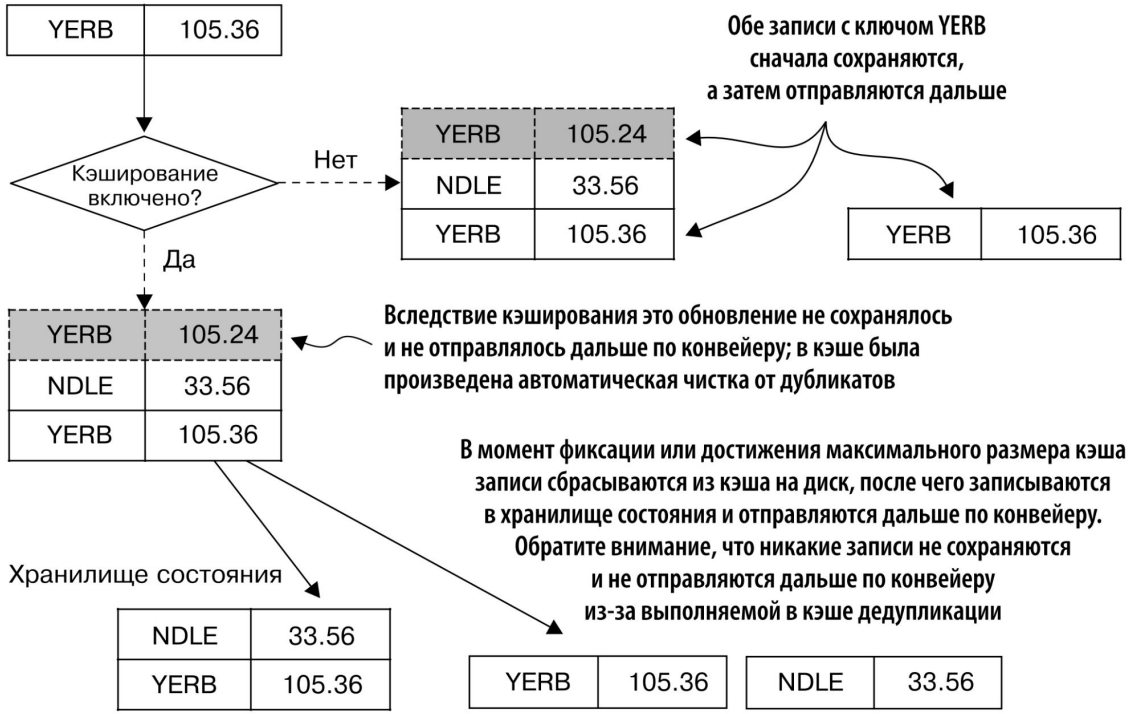
Ex7TableVsStream (**1)



KTable - кэширование

- Хранит состояние в локальном хранилище
- Отправляет состояние дальше по потоку в некоторых ситуациях
  - Чем больше входной поток - тем чаще
  - Чем больше уникальных ключей - тем чаще
  - cache.max.bytes.buffering - размер кэша
  - commit.interval.ms - частота фиксации

Ex7TableVsStream (**1, **2)

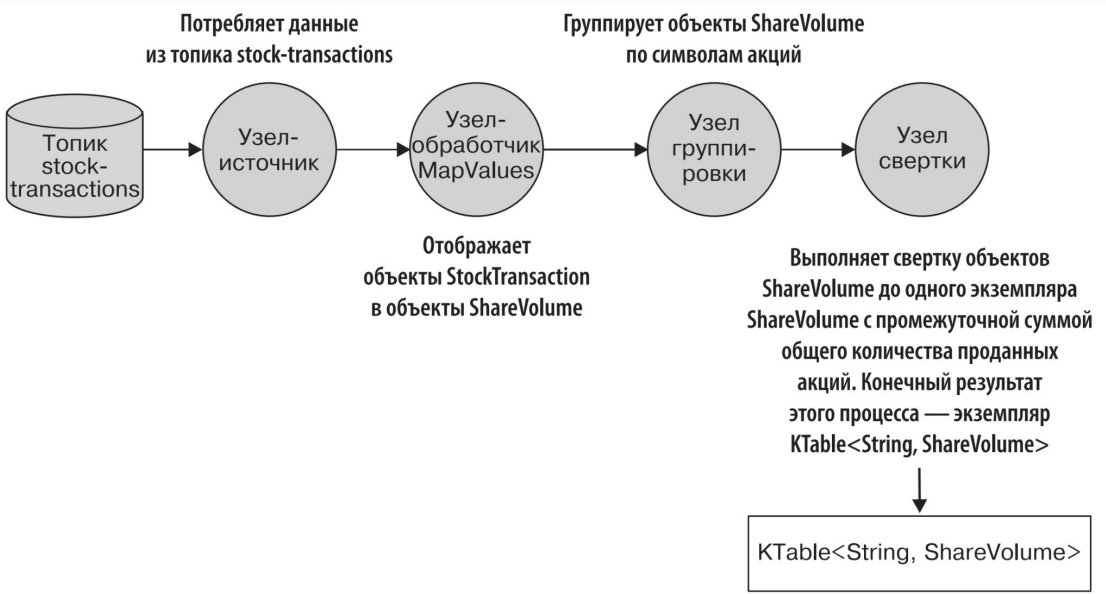


Задача

Есть данные по сделкам с акциями (StockTransaction). Надо

- посчитать кол-во проданных акций по каждой компании
- сгруппировать по сферам деятельности
- по каждой сфере выдать 3 компании с наибольшим оборотом

Ex8Aggregation

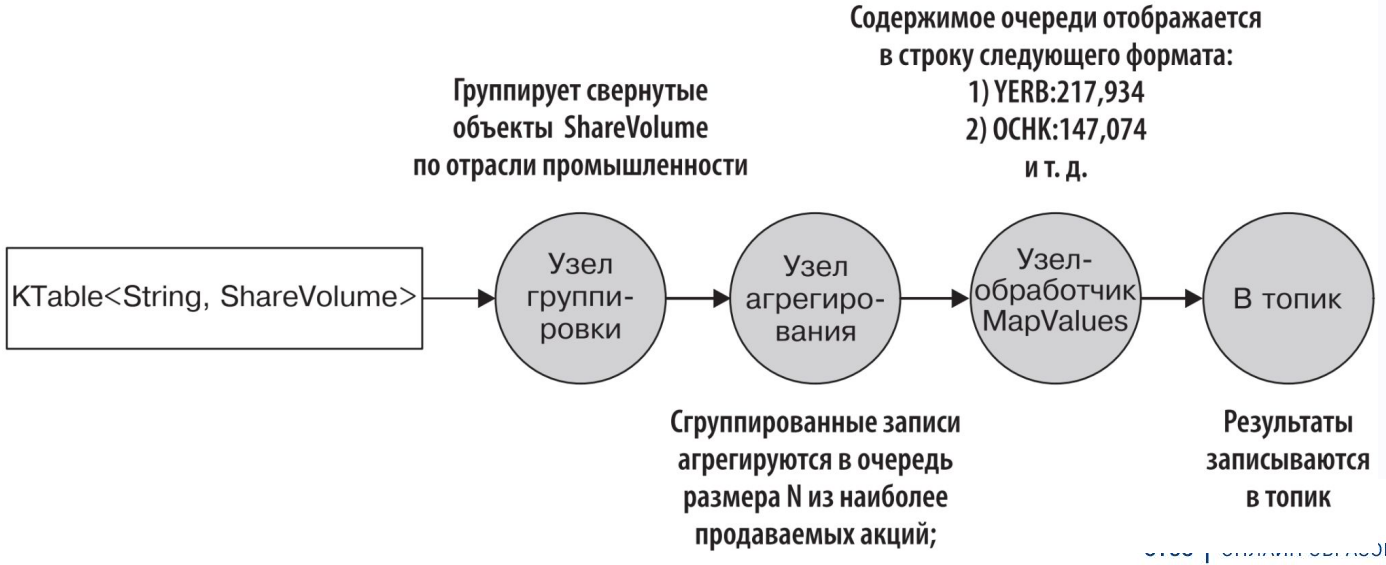


Задача

Есть данные по сделкам с акциями (StockTransaction). Надо

- посчитать кол-во проданных акций по каждой компании
- сгруппировать по сферам деятельности
- по каждой сфере выдать 3 компании с наибольшим оборотом

Ex8Aggregation



KGroupedStream

KStream.groupBy()

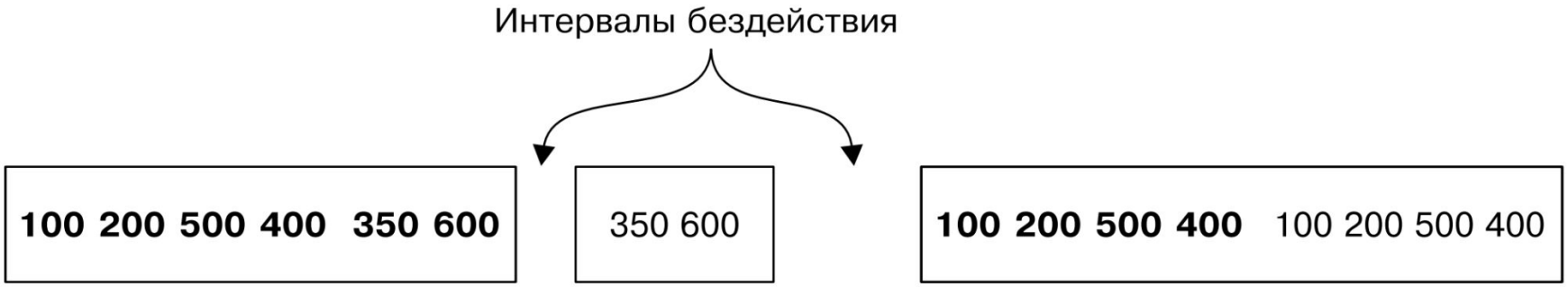
- count
- reduce
- aggregate
- windowedBy

При группировке таблицы возвращается KGroupedTable, у которой похожий интерфейс (кроме windowedBy)

Оконные операции

Мы только что занимались непрерывным агрегированием.
Но иногда хочется агрегировать по какому-то временному окну - скажем сколько было операций с акциями за 10 минут. Или сколько пользователей нажали на баннер за 20 минут. При этом такие операции надо проводить непрерывно

Виды окон - сеансовые



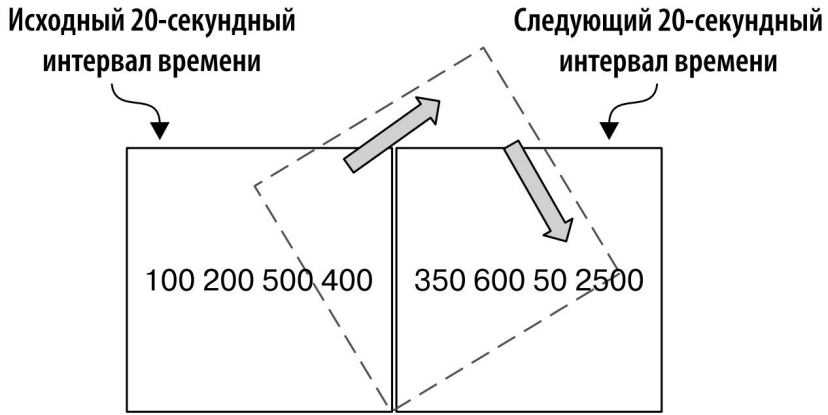
Интервал бездействия здесь невелик, так что эти сеансы, вероятно, окажутся слиты в один больший сеанс

Интервал бездействия здесь довольно большой, так что новые события попадут в отдельный сеанс

Сеансовые окна отличаются тем, что не ограничиваются по времени, а отражают интервалы активности. Границы сеансов отмечаются указанными периодами бездействия

Виды окон - кувыркающиеся

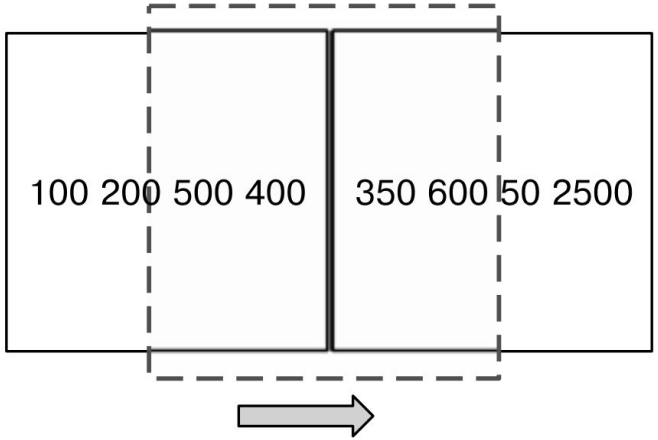
Текущий интервал времени «кувырком» полностью переходит (показано в виде штрихового квадрата) в следующий интервал времени, без всякого их перекрытия



Квадрат слева соответствует первому 20-секундному интервалу времени. Через 20 секунд он «кувыркается» для захвата событий в новом 20-секундном промежутке времени

События не перекрываются. Первое окно содержит события [100, 200, 500, 400], а второе — [350, 600, 50, 2500]

Виды окон - скользящие



Квадрат слева — первое 20-секундное окно, которое каждые 5 секунд скользит вправо (обновляется), создавая новое окно. На рисунке видно, что события пересекаются. Окно 1 содержит [100, 200, 500, 400], окно 2 содержит [500, 400, 350, 600], а окно 3 — [350, 600, 50, 2500]

Задача

Допустим мы хотим посчитать кол-во операций для трейдера+тикета относительно окна

Ex9Window

Что осталось за кадром

Что осталось за кадром

- GlobalKTable - вытаскивает данные из всех партиций. Это может быть полезно, если данных мало, чтобы не заниматься репартиционированием
- join таблиц
- Запросы к локальным хранилищам состояния
- Processor API (то, чем мы делали сегодня - это фактически dsl для него)

Вопросы?

Ставим “+”,
если вопросы есть

Ставим “–”,
если вопросов нет

Рефлексия

Ключевые тезисы

1. KStream - поток событий из топика
2. KTable - интерпретация топика как потока обновлений таблицы

Заполните, пожалуйста, опрос о занятии по ссылке в чате

Спасибо за внимание!

Приходите на следующие вебинары

Непомнящий Евгений

Разработчик Java/ Kotlin IT-Sense

@evgeniyN