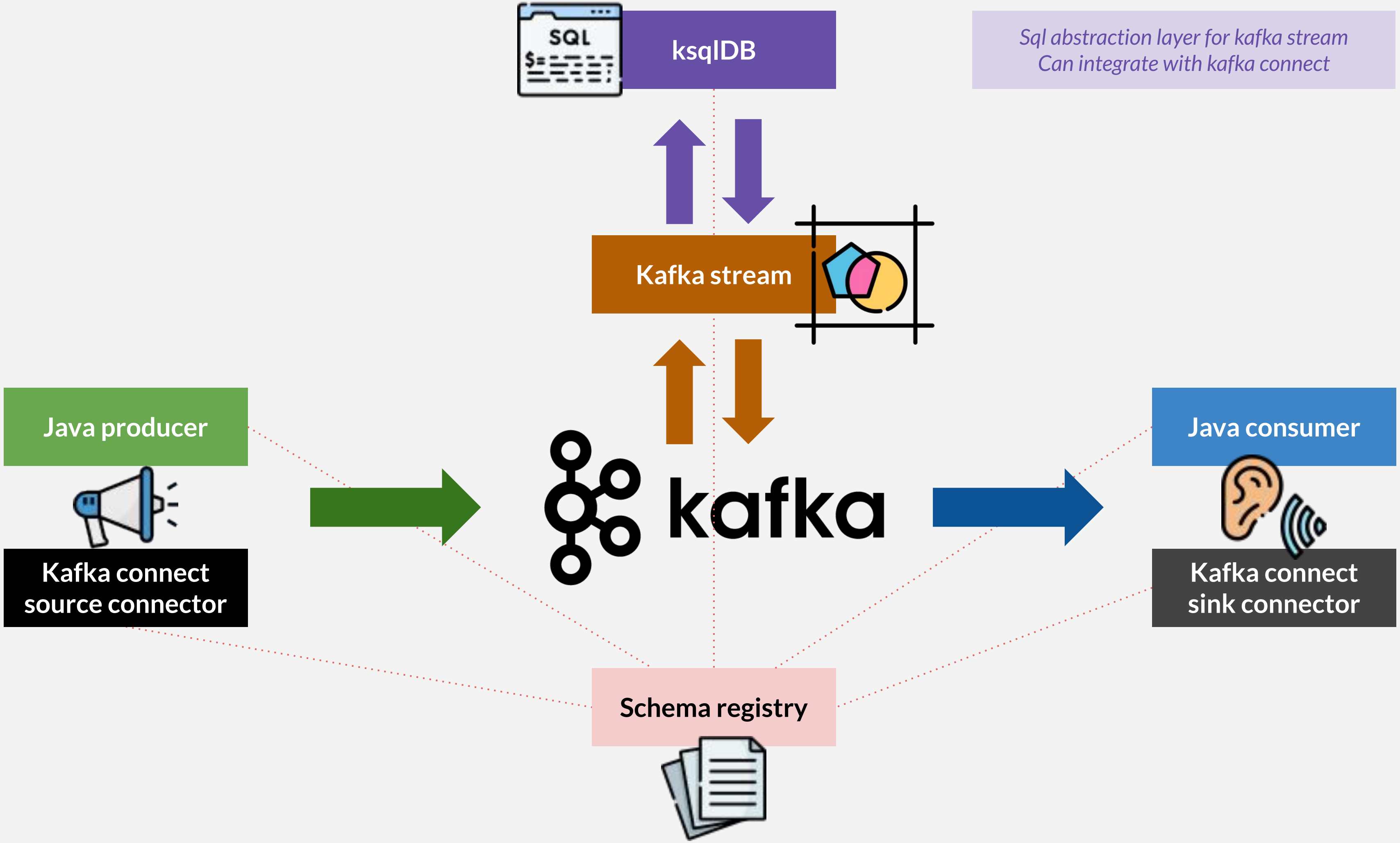
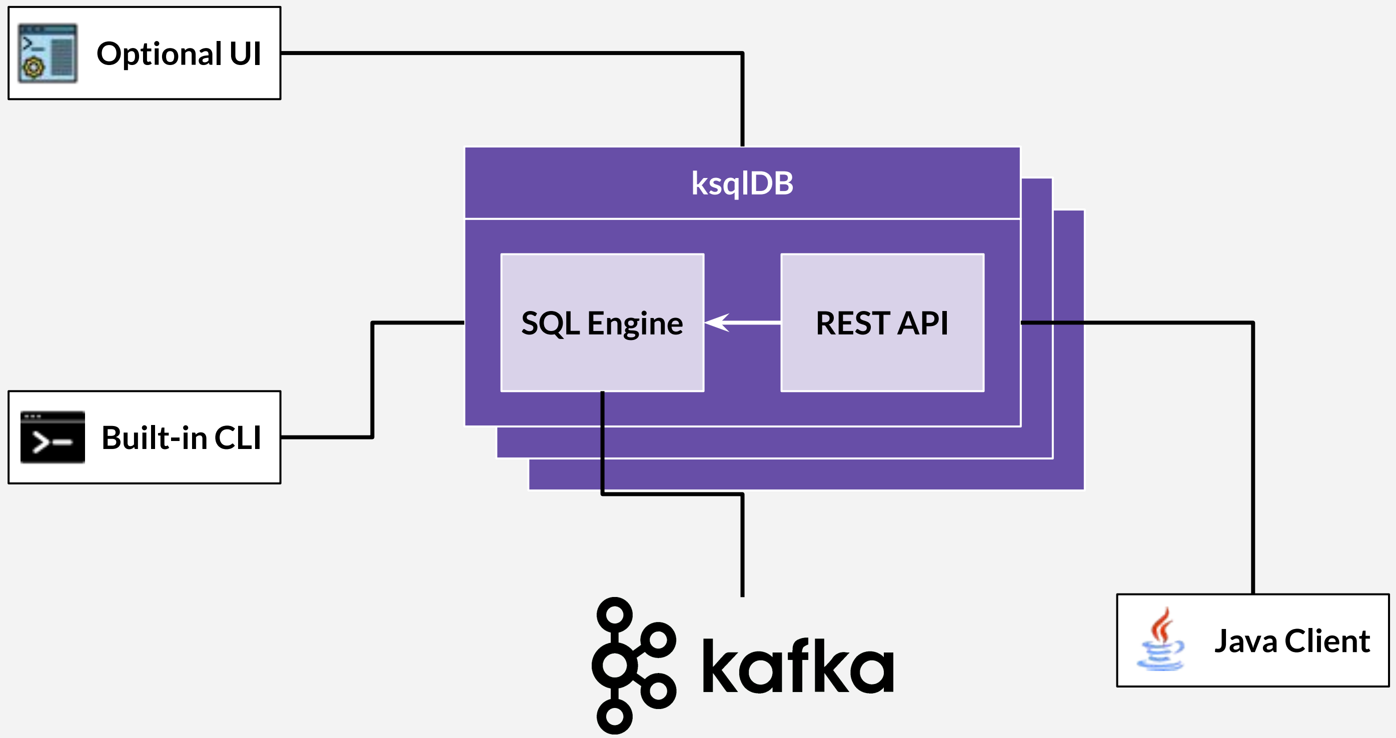
Introduction to ksqlDB



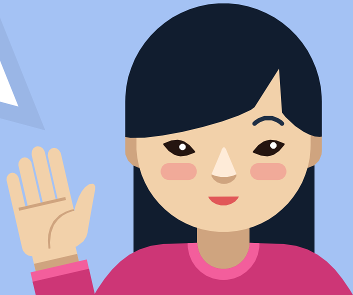
Sql abstraction layer for kafka stream
Can integrate with kafka connect

ksqlDB Architecture



Abstraction

- ksqlDB generates kafka stream code
- Abstraction in kafka stream
- Abstraction in ksqlDB
  - Write SQL-like syntax
  - ksqlDB generates kafka stream codes
  - Works with SQL interface, not kafka stream



Why SQL?

- × JVM only (Java, scala)
- × High learning curve
- × Most engineers familiar with SQL
- × SQL has lower learning curve
- × ***You must already familiar with SQL***
- × ***This course is not SQL course***

ksqlDB and Database

- Kafka as data store & ksqlDB as SQL interface
- ***Not*** replacement to database product (PostgreSQL, MySQL, etc)
- Kafka itself is not database replacement
  - Example: immutable message
- Complement for database
- ksqlDB is ***SQL-like***

ksqlDB vs Relational Database

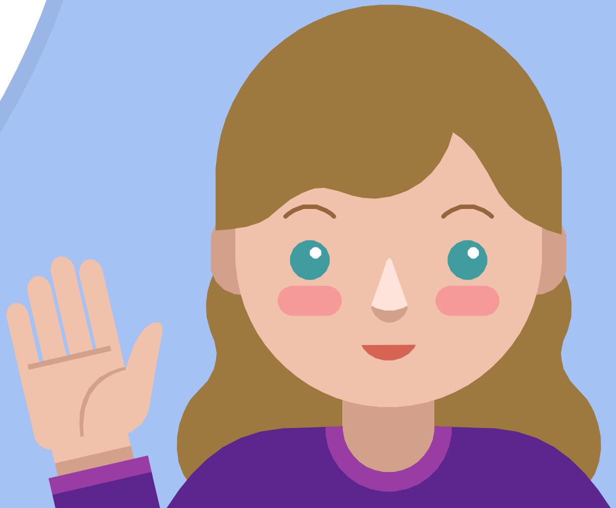
Similarities

- Uses SQL to interact with data
- DDL & DML statements
- Database client (CLI, Java) + ksqlDB REST API
- Schema
- Materialized view
- Built-in functions & operators

Differences

- Different SQL dialect
- Continuous query (push query)
- Schema registry

Interact With ksqlDB

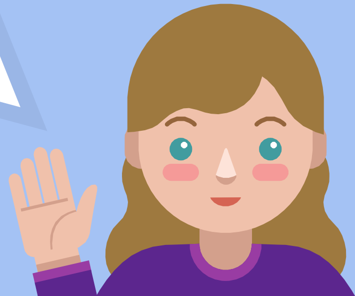


ksqlDB Server

- This course uses ksqlDB on docker
- On *docker-compose-full.yml*
- Runs on port 8088
- Also available as native installation
- Go to *ksqldb.io*

How To Start

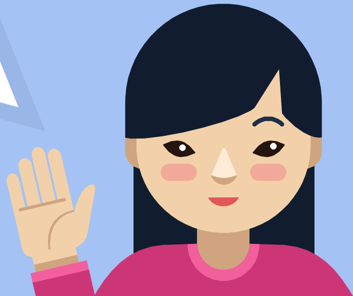
- Do these:
  - Stop docker (`docker-compose down`)
  - Delete subfolder `data`
  - Start docker compose using file **docker-compose-full.yml**
  - Run **kafka order java projects** from kafka stream lesson
  - Re-create kafka stream topics
- How-to & reference on last section



ksqlDB Syntax Reference

ksqlDB Syntax Reference

- ksqlDB has many syntax
- Some of them in this course
- Syntax reference
  - Last section of the course, lesson **Resources & References**
  - ksqlDB website (link available on lesson **Resources & References**)
- On the course : ksqlDB statement & brief explanation
- Commands are case-insensitive



Use Cases

Sample Use Cases

- Same use cases from kafka stream lesson
- ksqlDB will create kafka stream behind the screen
- See kafka stream lesson for refresher
- Create topics from kafka stream lesson
- Run kafka-stream-order project
- Use kafka stream postman collection for transaction

1

Hello

Promotion uppercase

2

Commodity

Key / value transformation, filter, branch

3

Feedback

KTable, grouping, counting

4

Customer

Merge, cogroup

5

Flash sale vote

State, stateful operations

6

Inventory

Grouping, aggregate, windowing

7

Online order & online payment

Join stream / stream

8

Web color & layout vote

Join table / table

9

Premium purchase & user

Join stream / table

10

Subscription purchase & user

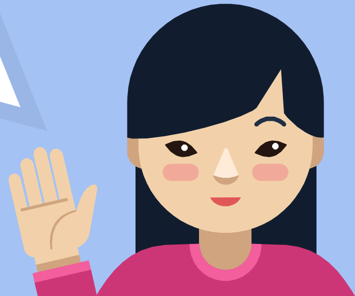
Join stream / global table

OTHER

Additional use cases

Java Source Codes

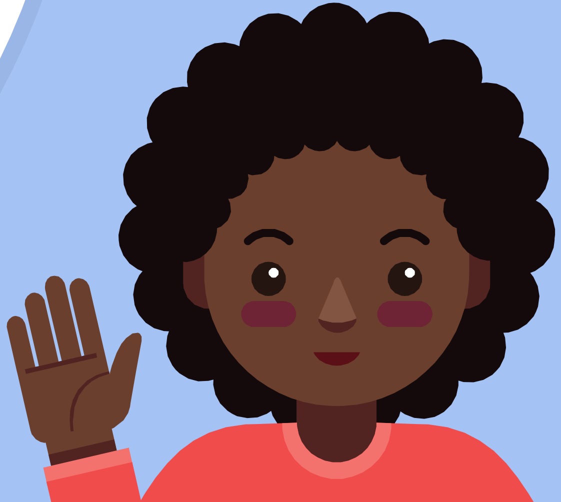
- Some java source code
- Will not go line-by-line 100%
- Download from lecture **Resource & Reference**
- Runs on localhost:9002
- Source code structure resembles kafka stream lesson
- Created from start.spring.io
  - Group : **`com.course.kafka`**
  - Artifact : **`kafka-ksqldb-sample`**
  - Package name : **`com.course.kafka`**
  - Dependency : **`Spring for Apache Kafka, Jackson`**



ksqlDB Scripts

- Explained, but not typing letter by letter
- Copy paste to ksqlDB console
- Download from lecture **Resource & Reference**
- Use AI assistant to explain the script

Hello ksqlDB



Examine message structure in topic

```
PRINT `input-topic`;
```

Create stream from input topic

```
CREATE STREAM `stream-name` (
  column_name data_type,
  ...
) WITH (
  kafka_topic = 'input-topic',
  property_name = 'property-value',
  ...
);
```

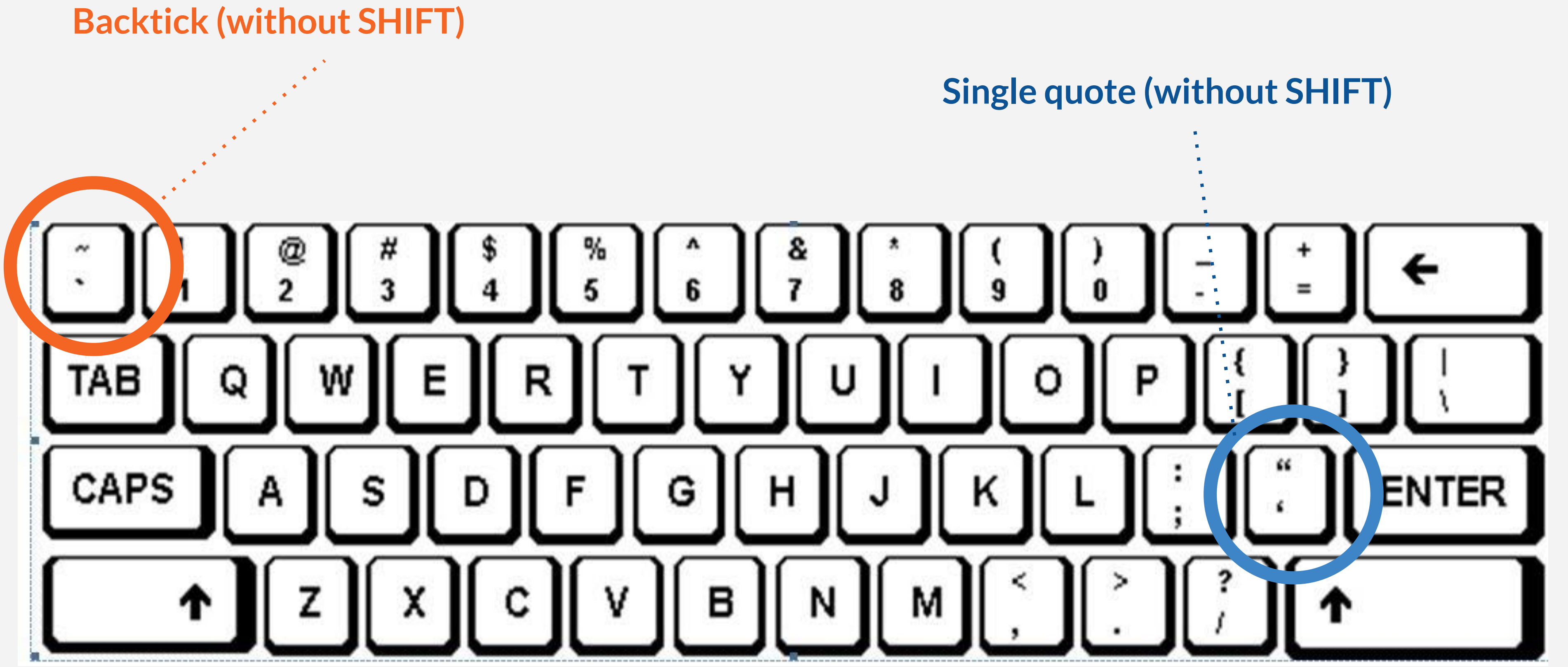
Create ksqldb statement as needed

```
SELECT ...
  FROM `stream-name`
  WHERE ...
EMIT CHANGES;
```

Create stream from select statement, to output topic

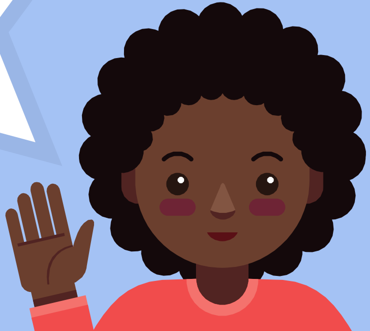
```
CREATE STREAM `output-topic` AS
```

Use backtick (`) if name contains special character
Use single quote (') if string contains special character
Terminate statement with semicolon (;)
New line is optional, not to terminate statement
Case insensitive

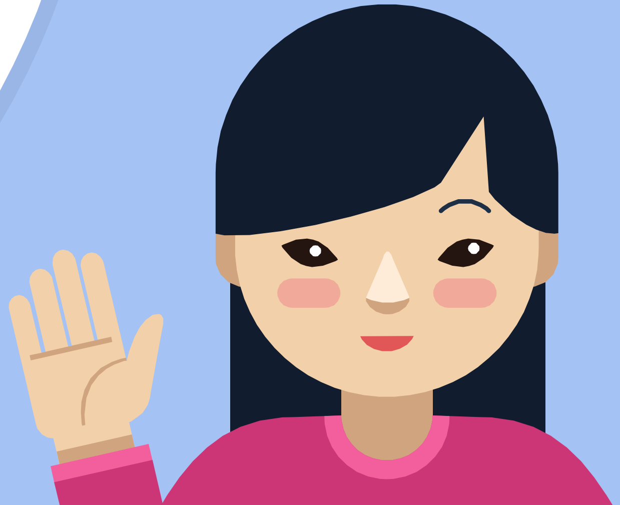

Backtick (without SHIFT)
Single quote (without SHIFT)
TAB
CAPS
ENTER

Push Query

× Kafka consumer constantly monitor & takes data from topic
× This behaviour in ksqldb called as **push query**
× Push query monitor & takes data until
    × Terminate manually (CTRL-C)
    × Hit defined **`LIMIT`** on **SELECT** statement
× **`EMIT CHANGES`** at the end of **SELECT**



Basic ksqlDB Stream



```
CREATE STREAM `s-commodity-promotion-uppercase`
AS
  SELECT UCASE(promotionCode) AS uppercasePromotionCode
    FROM `s-commodity-promotion`
  EMIT CHANGES;
```

Customize field name

```
CREATE STREAM `s-commodity-promotion-uppercase`
AS
  SELECT UCASE(promotionCode) AS `UPPERCASE_promotion-Code`
    FROM `s-commodity-promotion`
  EMIT CHANGES;
```

```
AS
  SELECT UCASE(promotionCode) AS uppercasePromotionCode
    FROM `s-commodity-promotion`
  EMIT CHANGES;
```

ksqlDB Tips

Show data from topic / stream

```
PRINT `my-topic`;


SELECT ... FROM `my-stream`
EMIT CHANGES;
```

Create stream if it not exists, do nothing otherwise

```
CREATE STREAM IF NOT EXISTS `my-stream`;
```

Create stream if it not exists, or replace existing

```
CREATE OR REPLACE STREAM `my-stream`;
```

Delete stream

```
DROP STREAM `my-stream`;


DROP STREAM IF EXISTS `my-stream`;
```

Full reference on ksqldb documentation (website : ksqldb.io)

Data Types

Data Types

Data type
`CREATE STREAM `my-stream` ( field1 data_type, field2 data_type ) ...`

```
CREATE STREAM `my-stream` (
  field1 data_type,
  field2 data_type
)
...
```

ksqlDB keyword	Java counterpart
BOOLEAN	boolean
VARCHAR STRING	String
BYTES	byte[]
INT	int
BIGINT	long
DOUBLE	double
DECIMAL(precision, scale)	BigDecimal

ksqlDB keyword	Java counterpart
DATE	Epoch day
TIME	Millis of day
TIMESTAMP	Epoch millis
ARRAY	Java array
MAP	java.util.Map
STRUCT	Nested class

Full reference on ksqldb documentation (website : *ksqldb.io*)

Data Types

- Use Java project for Kafka ksqlDB sample
- Download from ***Resources & References***
- Runs on port 9002

Data Types for Primitive

Data Types for Date & Time

Data Types for Date & Time

ksqldb keyword	Java counterpart	Description
`DATE`	Epoch day	Number of days since 1 January 1970. Value 0 means 1 January 1970, value 5 means 6 January 1970, etc. Negative numbers represent earlier days.
`TIME`	Millis of day	Millisecond of the day. 1 day is (24 *60 * 60) = 86,400 second, so the valid value for millis of day is 0 to 86,400,000
`TIMESTAMP`	Epoch millis	Number of milliseconds since 1 January 1970 00:00:00. Negative numbers represent earlier milliseconds.

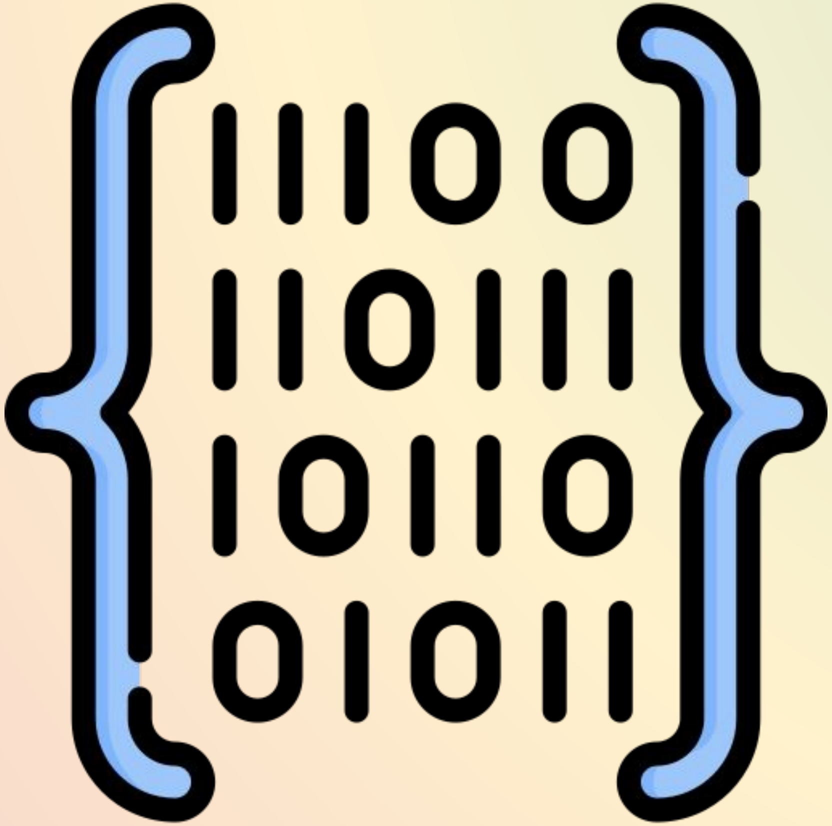
ⓘ Online epoch converter example : *epochconverter.com*

LocalDate, LocalTime, LocalDateTime

Java 8 Date Time Data Types

- Date / time at REST API is usually string
- ISO 8601 format
- Java **`LocalDate`**, **`LocalTime`**, **`LocalDateTime`**, **`OffsetDateTime`**
- Annotated with **`@JsonFormat`**
- ksqlDB?

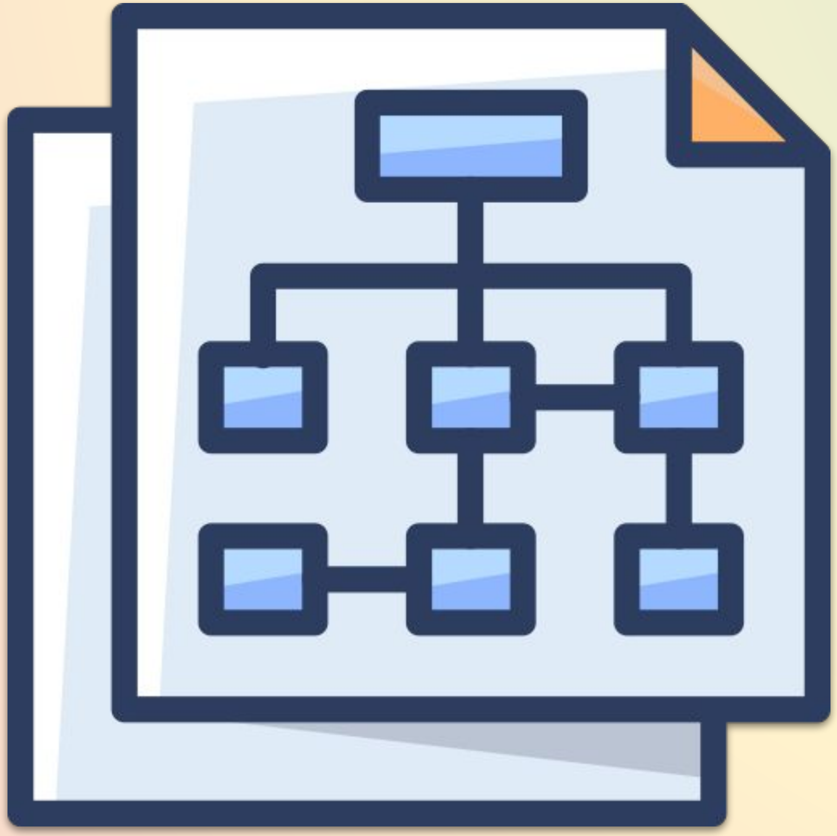
Array, List, Set



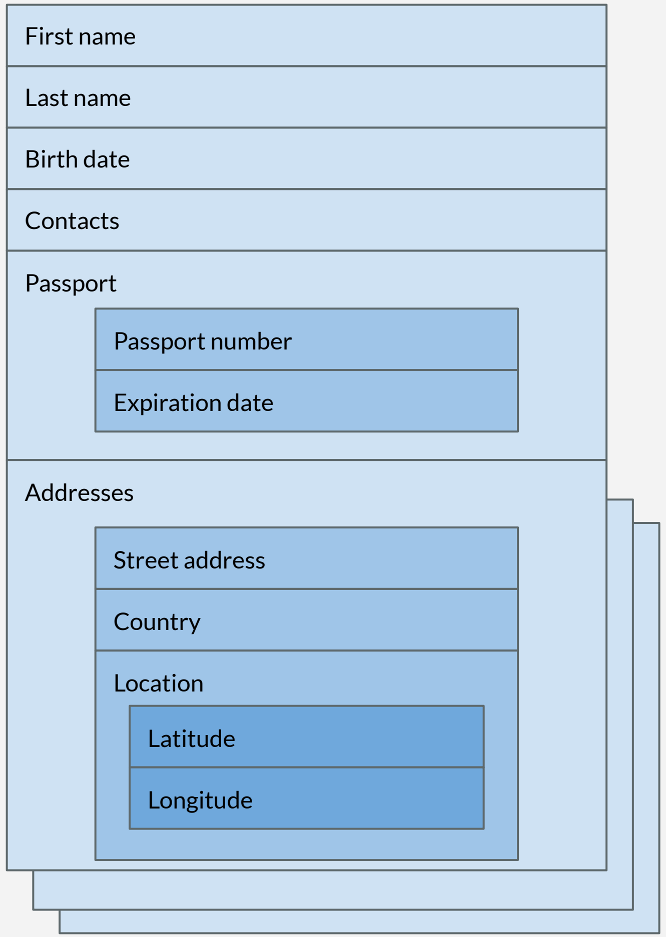
Map

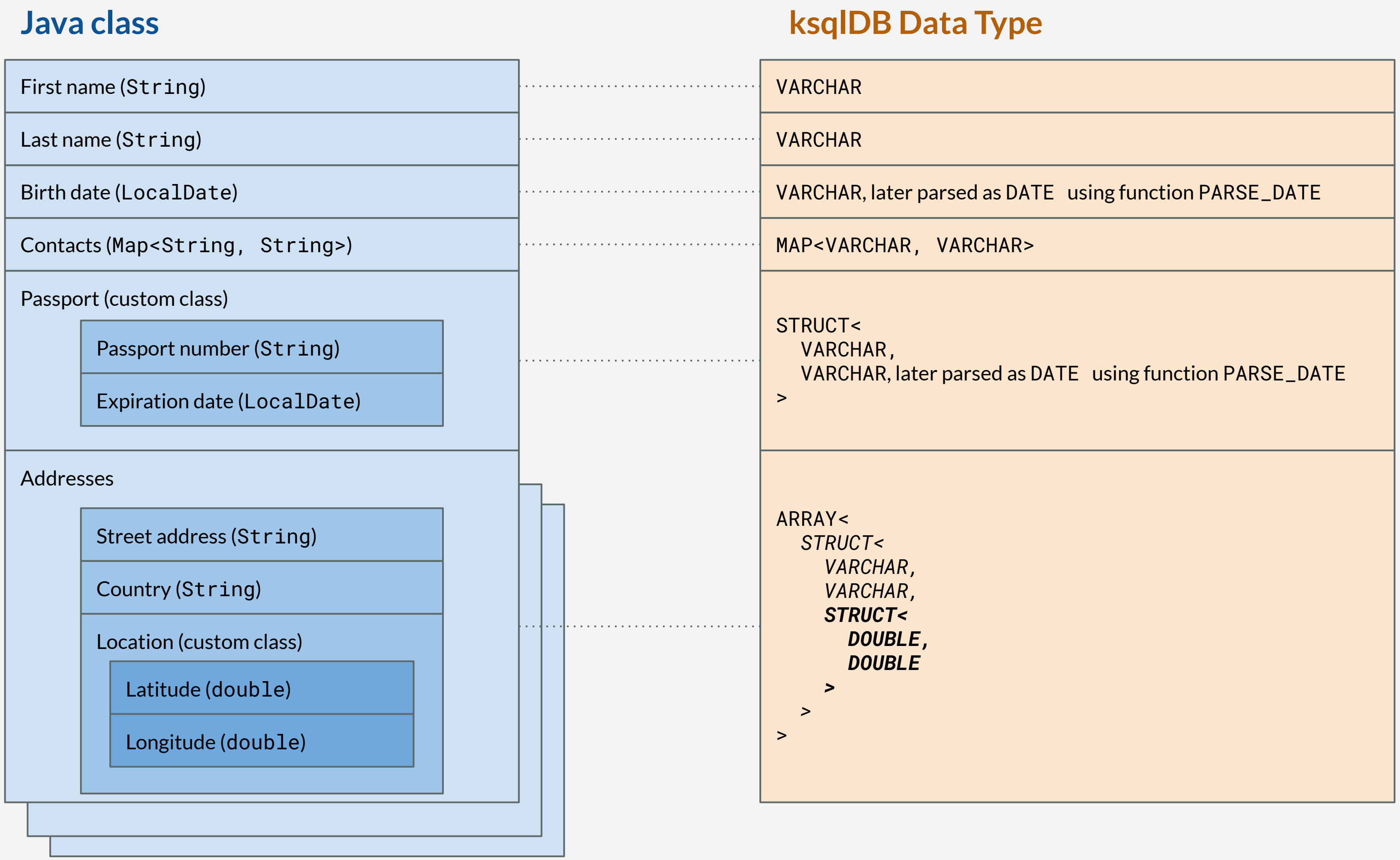
01	01	01
01	01	01
01	01	01

Complex Data Types



```json
{
    "firstName": "Adele",
    "lastName": "Hudson",
    "birthDate": "1999-11-25",
    "contacts": {
        "email": "adele.hudson@example.com",
        "phoneWork": "254-378-3404",
        "phoneHome": "569-811-9309"
    },
    "passport": {
        "number": "10668055",
        "expirationDate": "2026-12-15"
    },
    "addresses": [
        {
            "streetAddress": "4186 Hattie Throughway",
            "country": "Macao",
            "location": {
                "latitude": -36.514,
                "longitude": -166.6333
            }
        },
        {
            "streetAddress": "120 McLaughlin Heights",
            "country": "Myanmar",
            "location": {
                "latitude": -32.4193,
                "longitude": 85.1359
            }
        },
        {
            "streetAddress": "174 Marjolaine Viaduct",
            "country": "Christmas Island",
            "location": {
                "latitude": -14.2745,
                "longitude": -164.8439
            }
        }
    ]
}
```





```
VARCHAR
```

```
VARCHAR
```

```
VARCHAR, later parsed as DATE  using function PARSE_DATE
```

```
MAP<VARCHAR, VARCHAR>
```

```
STRUCT<
   VARCHAR,
   VARCHAR, later parsed as DATE  using function PARSE_DATE
>
```

```
ARRAY<
   STRUCT<
      VARCHAR,
      VARCHAR,
      STRUCT<
         DOUBLE,
         DOUBLE
      >
   >
>
```

Data Types

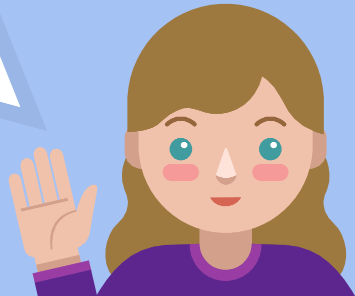
With AI Assistant

AI Assistant

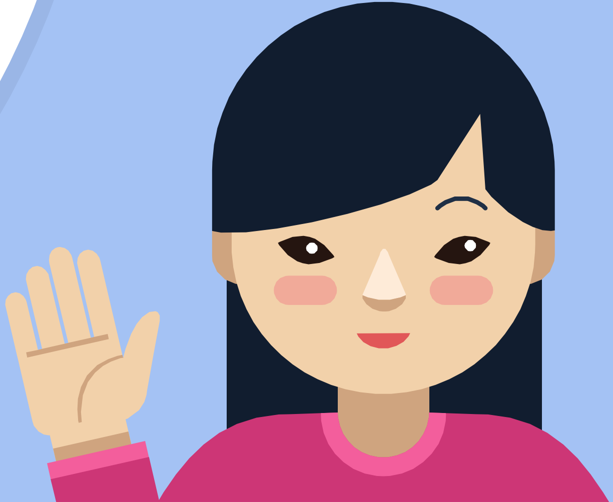
- A pattern exists
- Good enough to create stream definition
- Not reliable on create ksqlDB query
- See more examples

AI Assistant

- Good enough to create stream definition
- Not always good with query (especially functions)
- Feel free to use AI assistant
- The course will give manual examples of stream definitions & queries



Stream & Table Key



Use Case

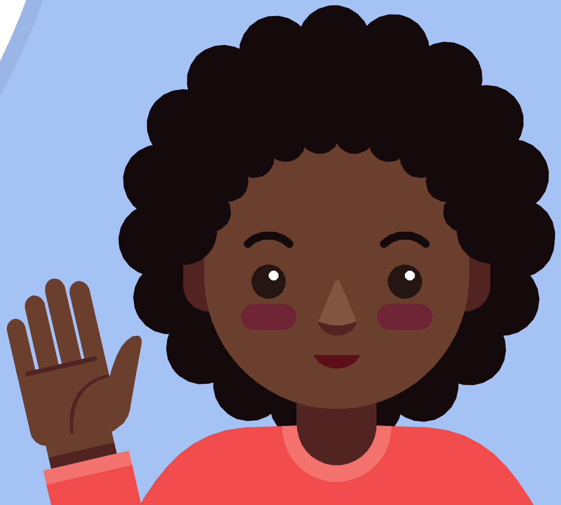
- Project *kafka-ksqldb-sample* : Country
- Original : publish without record key
- Add / change key (rekey)
- Stream / table at ksqlDB is equal to stream / table at Kafka Stream (earlier lesson)
- Data combination might not valid
- Field **`population`** will be used for aggregation

Stream vs Table Key

Case	Stream	Table
NULL key	No effect	Message is ignored
Same key with existing	No effect	Create new record with same key and updated value
Same key, NULL value	Message is ignored	Treaded as *tombstone*. Create new record with same key and NULL value (as if record is deleted)

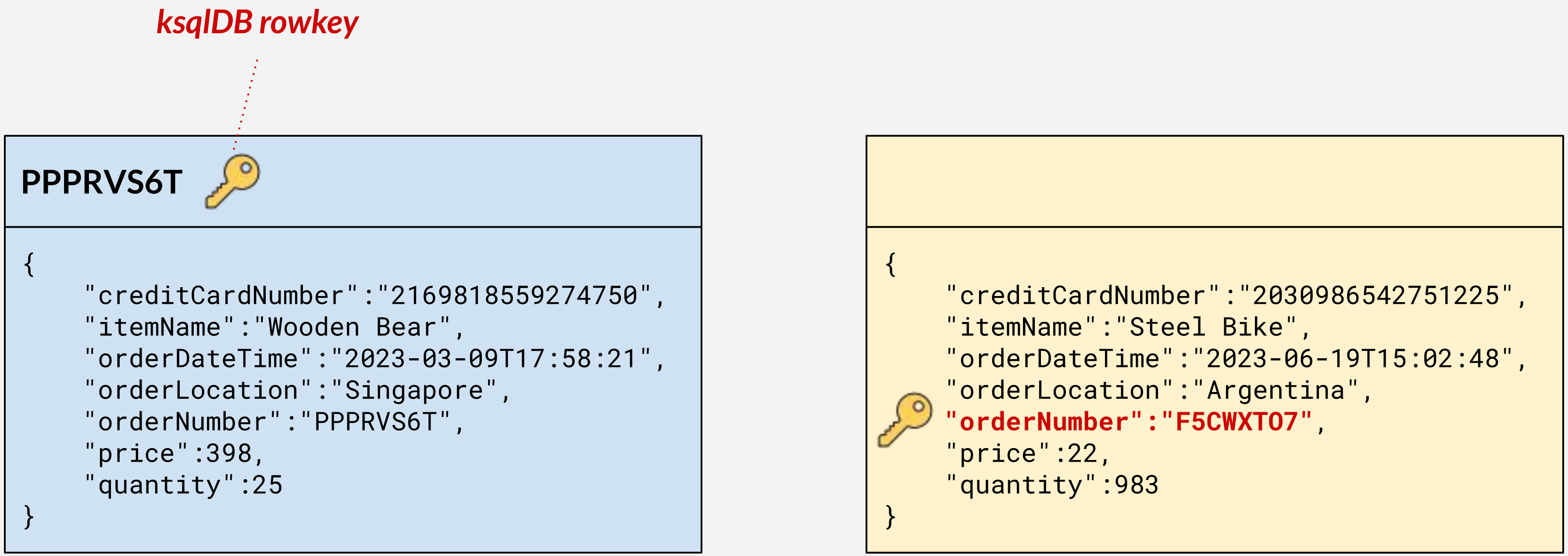
Commodity Stream

First Step



Row Key

Alternative

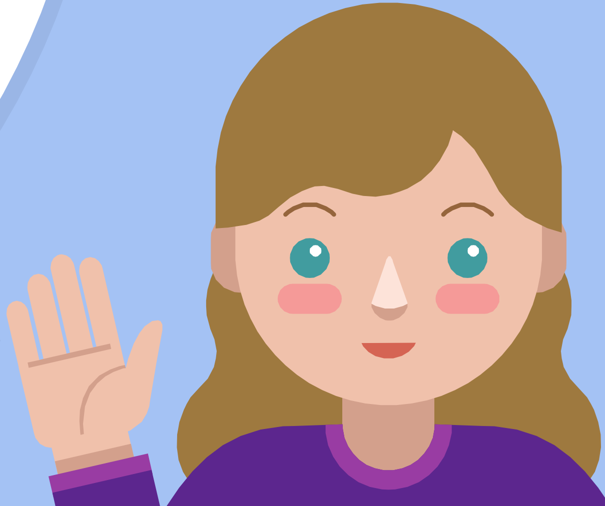


```
{
    "creditCardNumber":"2169818559274750",
    "itemName":"Wooden Bear",
    "orderDateTime":"2023-03-09T17:58:21",
    "orderLocation":"Singapore",
    "orderNumber":"PPPRVS6T",
    "price":398,
    "quantity":25
}
```

```
{
    "creditCardNumber":"2030986542751225",
    "itemName":"Steel Bike",
    "orderDateTime":"2023-06-19T15:02:48",
    "orderLocation":"Argentina",
    "orderNumber":"F5CWXTO7",
    "price":22,
    "quantity":983
}
```

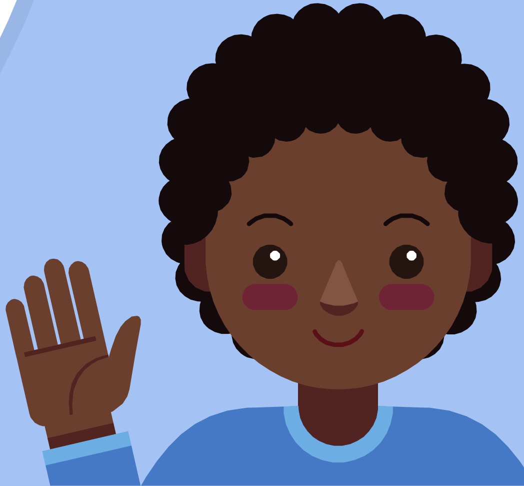
Commodity Stream

Additional Requirements

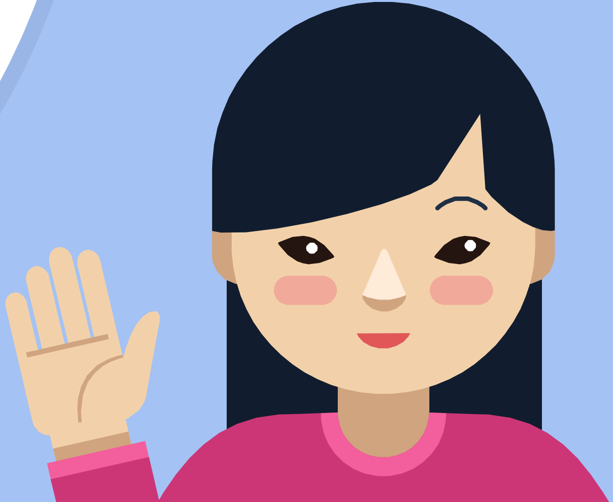


Commodity Stream

Reward Each Location



Run Script File



1

```
CREATE STREAM IF NOT EXISTS `s-first-stream` (
  `price` INT,
  `name` VARCHAR
) WITH (
  KAFKA_TOPIC = 't-source-topic',
  VALUE_FORMAT = 'JSON'
)
```

2

```
DROP STREAM IF EXISTS `s-second-stream`;
```

3

```
CREATE STREAM `s-my-second-stream`
AS
SELECT *
  FROM `s-first-stream`
 WHERE `price` > 500
EMIT CHANGES;
```

4

```
SET 'auto.offset.reset'='earliest'
```

```
CREATE STREAM IF NOT EXISTS `s-complex-stream`
WITH (
  KEY_FORMAT = 'JSON',
  VALUE_FORMAT = 'JSON',
  PARTITIONS = 3,
  ...
)
AS
SELECT ...,
       ...
  FROM `s-base-stream`
WHERE <condition 1>
      AND <condition 2>
      AND <condition 3>
      ...,
      ...
PARTITION BY ...
EMIT CHANGES;
```

Commodity Stream

Calling API or Other Process

Calling API / other

- Previously : call API / other process from kafka stream
- Vanilla ksqldb **cannot** call API / other process
- Using user-defined-function
- Kafka stream is java code (greater flexibility)
- ksqldb is easier, but lost some power
- ksqldb provides a lot built-in functions
- Good (faster) alternative when no API call / other process involved

Calling Other Process

- User defined function (UDF) can be any Java code
  - Thus, can call API / other process
- I don't recommend calling other process on ksqlDB UDF
- KsqlDB should be independent
- Generally speaking : ksqlDB cannot call API / other

Commodity Stream

Further Fraud Processing

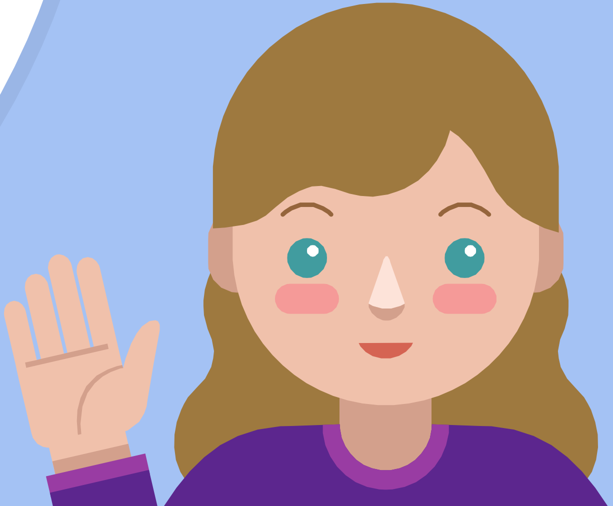
ksqlDB REST API

ksqlDB REST API

- Built-in REST API
- Statements & query
- Next use case will execute ksqldb statements from REST API
- Postman collection available at Resource & References
- ksqldb statements are same between console & REST API
- Complete reference link available at Resource & References

Feedback Stream

Are We Good Enough?



Good Feedback

- Steps
  1. Remove all non-alphabet / non-space
  2. Lowercase the string
  3. Split string by whitespace (become words)
  4. Distinct the words
  5. Filter only good words from the words stream
- **`SELECT DISTINCT`** is **not available** on ksql
- Use **`ARRAY_DISTINCT`**
- Subquery is ***not straightforward, but possible***

Feedback Stream

Who Owns This Feedback?

Feedback Stream

Good Feedback or Bad Feedback?

Feedback Stream

Group Using Table

Feedback Stream

Send and Continue

Send and Continue

```
streams.to("first-output-topic");
streams.groupByKey().....to("second-output-topic");
```

```
streams.through("first-output-topic").groupByKey().....toStream()
     .to("second-output-topic");
```

```
streams.repartition(
          Repartitioned.as("first-output-topic")
     ).groupByKey().....toStream().to("second-output-topic");
```

KSQL

```
CREATE STREAM `first-output` AS SELECT ...

CREATE TABLE `second-output` AS SELECT ...
```

Feedback Stream

Overall Good (or Bad)

Insert Data Using ksqlDB

Support stream or table

Single record

```
INSERT INTO stream-or-table-name (
  column-1, column-2, column-3, ...
) VALUES (
  value-1,value-2, value-3, ...
);
```

From stream

```
INSERT INTO target-stream-name
  SELECT ...
    FROM source-stream-name
  EMIT CHANGES;
```

Support stream only

Update & Delete

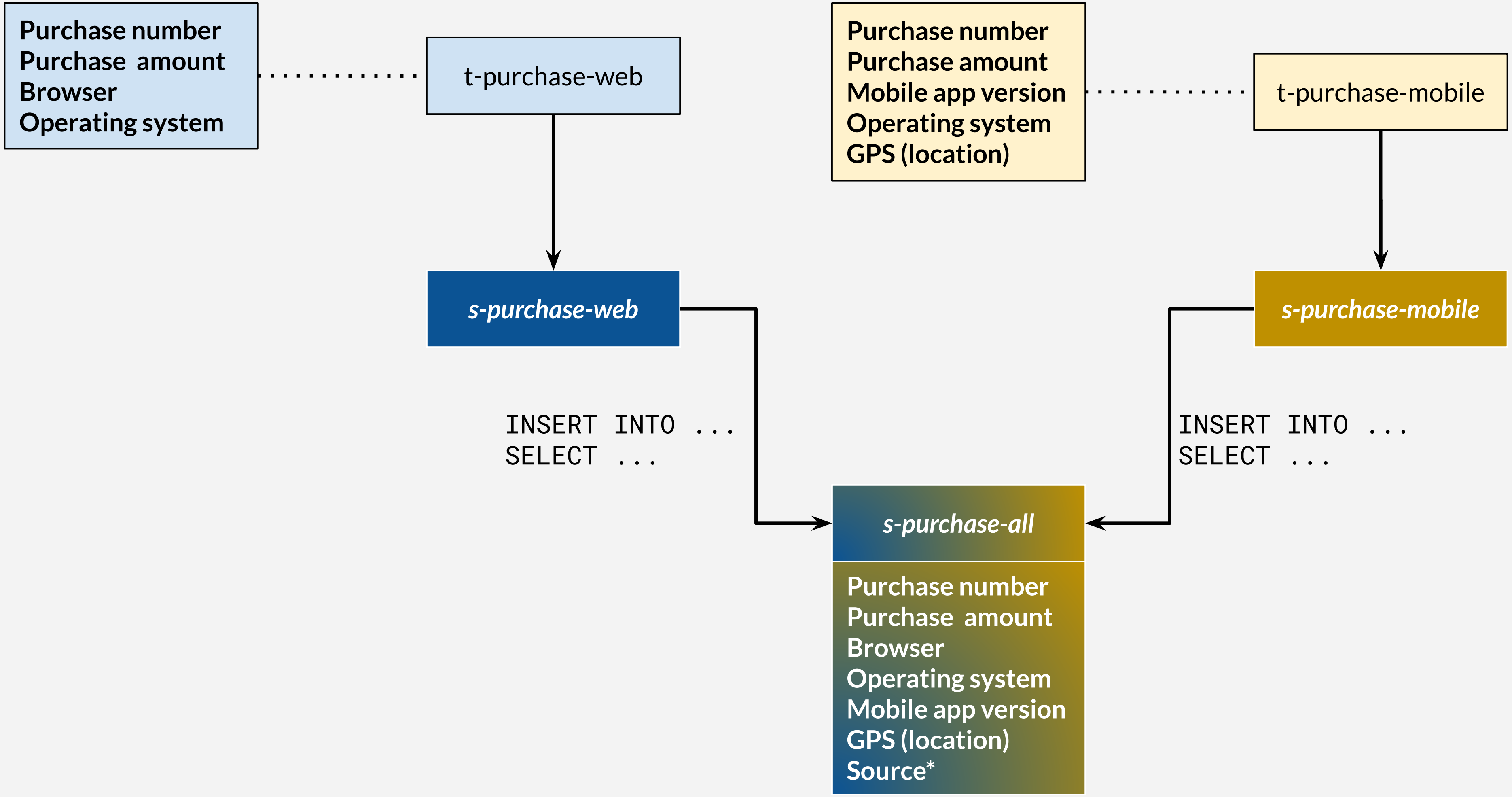
- Kafka data is immutable
- Update not available
- Delete single record not available
- Delete topic is possible
  - `DROP STREAM `my-stream` DELETE TOPIC;`
  - `DROP TABLE `my-table` DELETE TOPIC;`

Customer Stream

Web & Mobile

From Many, Into One

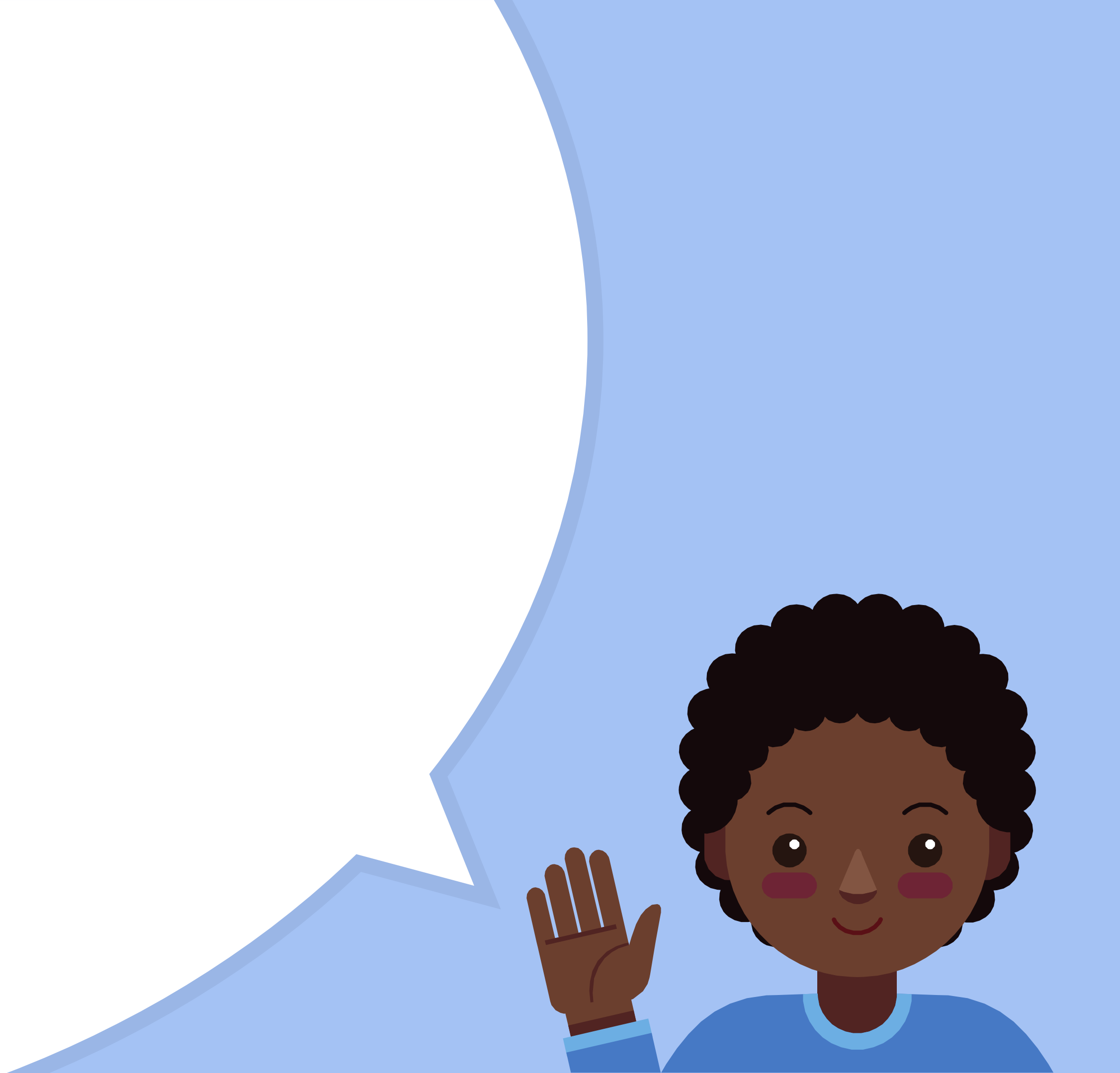
- Insert stream to other stream
- Merge multiple streams into one stream
- Example case


Purchase number
Purchase amount
Browser
Operating system
t-purchase-web
Purchase number
Purchase amount
Mobile app version
Operating system
GPS (location)
t-purchase-mobile
s-purchase-web
s-purchase-mobile
INSERT INTO ...
SELECT ...
INSERT INTO ...
SELECT ...
s-purchase-all
Purchase number
Purchase amount
Browser
Operating system
Mobile app version
GPS (location)
Source*

Customer Stream

Cart & Wishlist

Pull Query



Push Query

```
SELECT ...
  FROM ...
 WHERE ...
EMIT CHANGES
LIMIT n;
```

- New data will be sent directly as query result
- EMIT CHANGES
- Live until we terminate, or reaches LIMIT

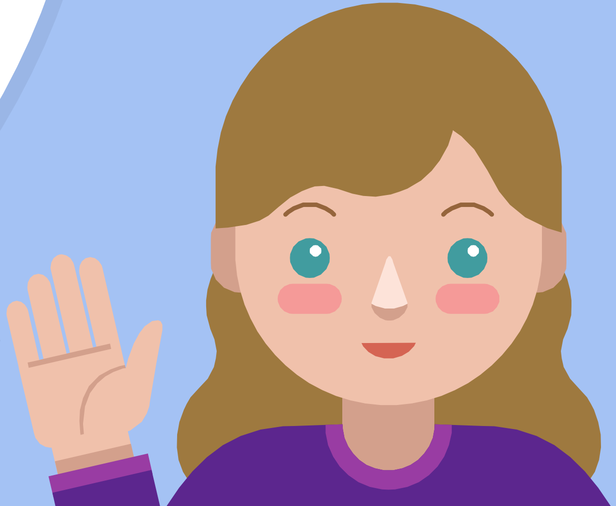
Pull Query

```
SELECT ...
  FROM ...
 WHERE ...
```

- Take snapshot of data
- Like traditional query
- No EMIT CHANGES
- New data will not be sent to output
- Since 0.23.1 can pull from stream / table

Flash Sale Stream

Most Recent Data Feed

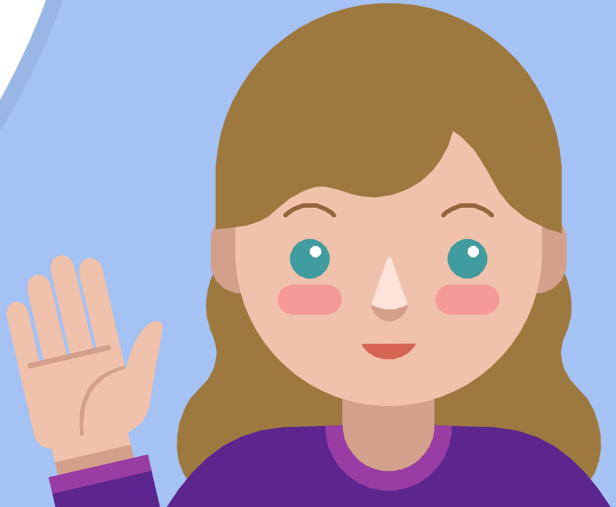


Flash Sale Stream

Timestamp

Feedback Rating

Average Rating



Feedback Rating

Detailed Rating

Inventory

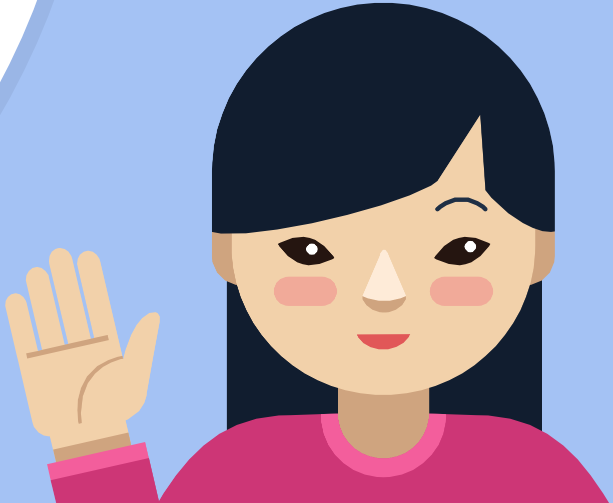
Summing Records
Subtracting Value

Inventory

Timestamp Extractor

Inventory

Tumbling Time Window



Inventory

Hopping Time Window

Inventory

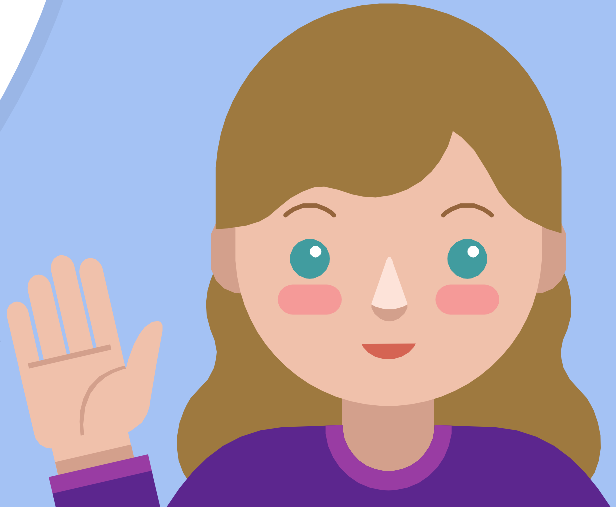
Session Window

Join

Inner Join Stream / Stream

Join

Left Join Stream / Stream



Join

Outer Join Stream / Stream

Synthetic Key

- Auto generated column by ksqlDB
- Full outer join case
- Synthetic key : first non null key

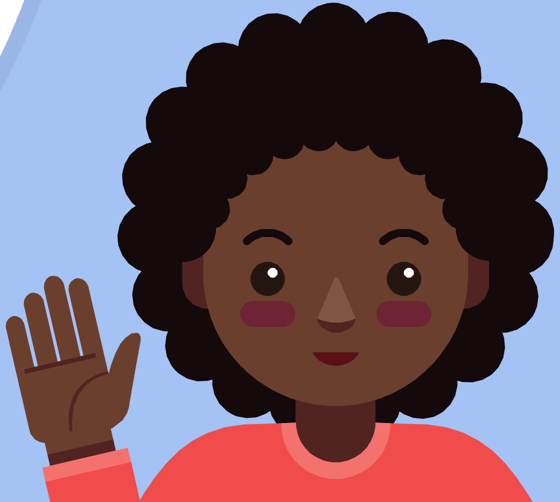
Left key	Right key	Synthetic key
99	null	99
null	88	88
77	77	77
null	null	No join

Join

Inner Join Table / Table

Join

Left Join Table / Table

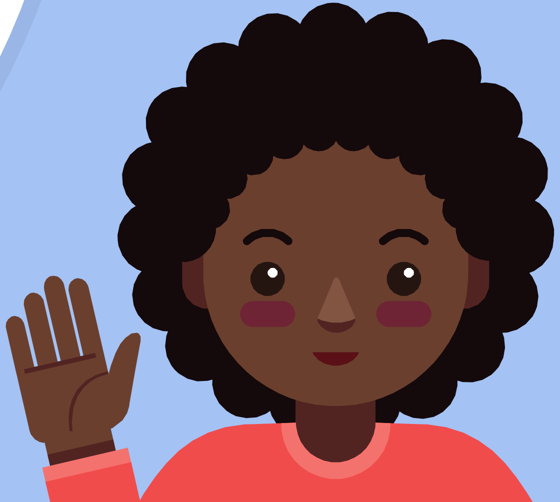


Join

Outer Join Table / Table

Join

Inner Join Stream / Table

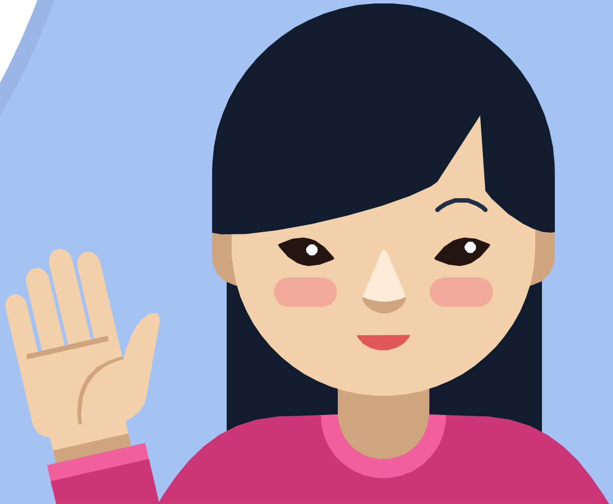


Join

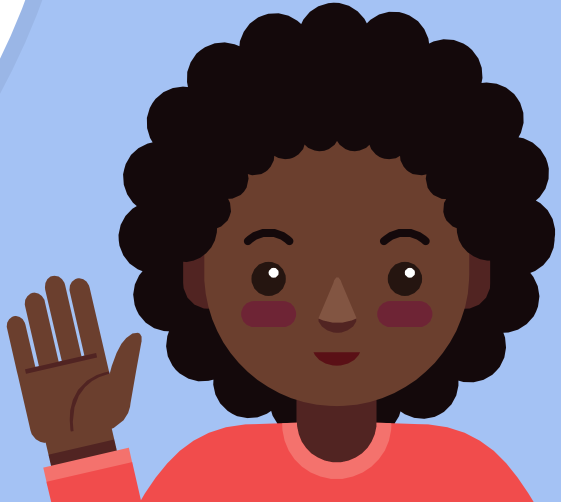
Left Join Stream / Table

Join

Stream / Table Co-Partition



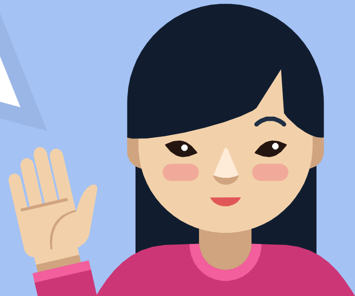
Enabling Exactly Once



User Defined Function (UDF)

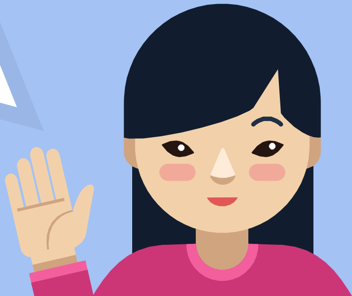
User Defined Function

- Make our own ksqlDB user-defined-function (UDF)
- Regular java project
- This course using gradle project
- Reference : **kafka-ksqldb-udf**
- UDF for calculate loan monthly installment
  - Principal loan amount
  - Annual interest rate
  - Loan period (months)

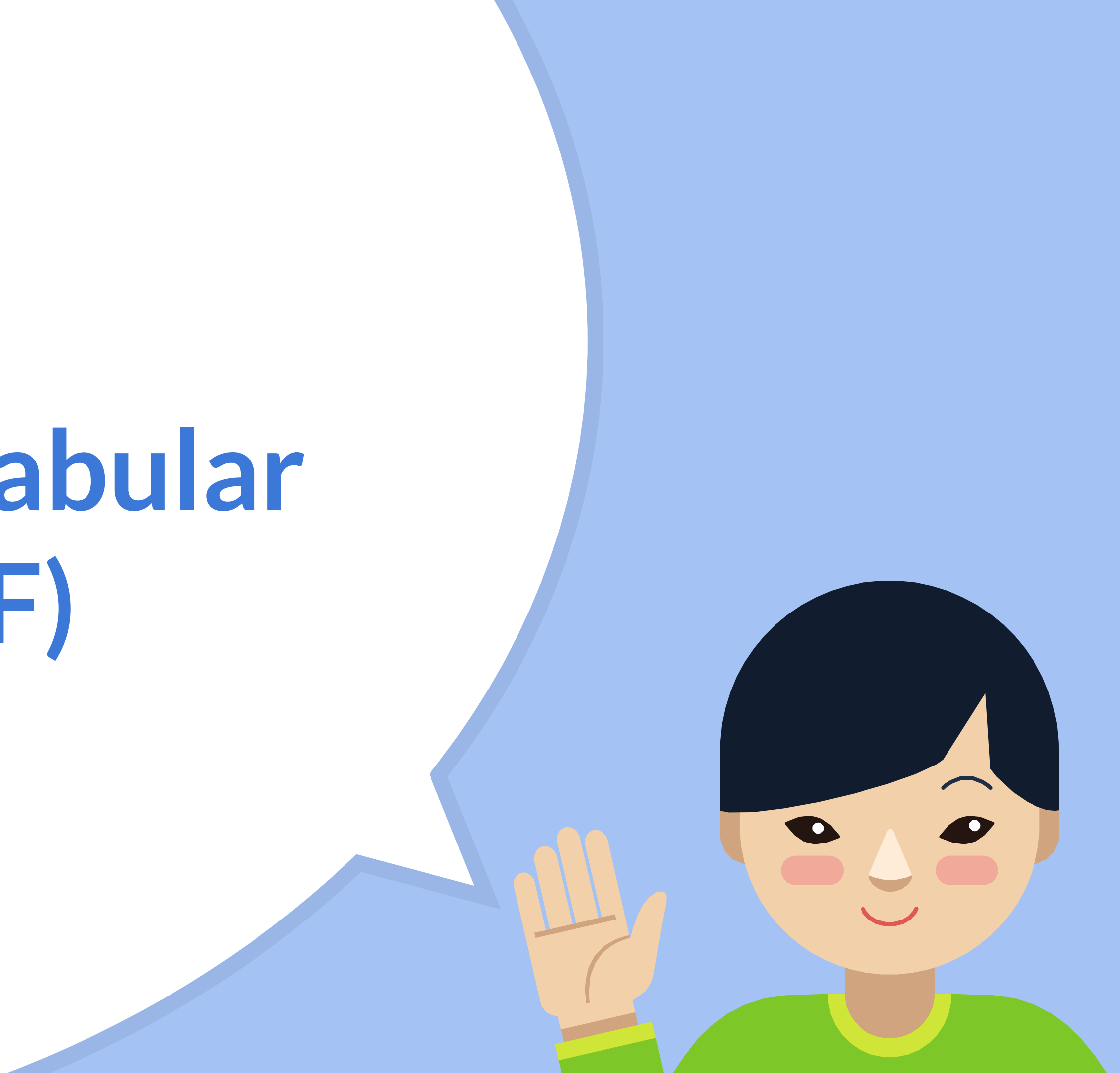


Creating UDF

- Regular Java class with annotation
- `@UdfDescription(name="...")`
- `@Udf`
- `@UdfParameter`



User Defined Tabular Function (UDTF)



UDTF

- One input, one or more output
- Flatmap
- Can use struct
- This lesson : UDTF for loan installment schedule
- Input : **struct loan submission** (principal loan amount, annual interest rate, loan period month, loan approved date)
- Output : **list of monthly installment** (installment amount, installment due date)

Struct LoanSubmission

```
principalLoanAmount
annualInterestRate
loanPeriodMonth
loanApprovedDate
```

Struct Loan Monthly Installment

```
installmentAmount
installmentDueDate
```

Schema String / Descriptor
(on annotation)

```
STRUCT<`principalLoanAmount` DOUBLE,
       `annualInterestRate` DOUBLE,
       `loanPeriodMonth` INT,
       `loanApprovedDate` VARCHAR
      >
```

```
STRUCT<`installmentAmount` DOUBLE,
       `installmentDueDate` VARCHAR
      >
```

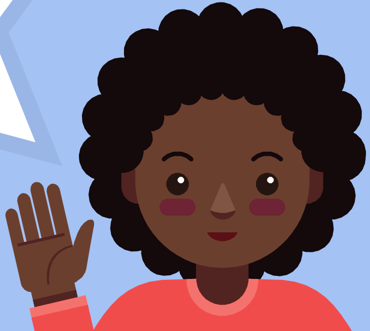
Schema instance (to be used as java class)

```
SchemaBuilder.struct()
  .field("installmentAmount", Schema.FLOAT64_SCHEMA)
  .field("installmentDueDate", Schema.STRING_SCHEMA);
```

User Defined Aggregation Function (UDAF)

UDAF

- **`@UdafFactory`** and class implementation
- This lesson : UDAF to calculate loan rating
- No data validation / error handler



Payment latency (days) = actual payment date - due date

👎 Positive : bad payment

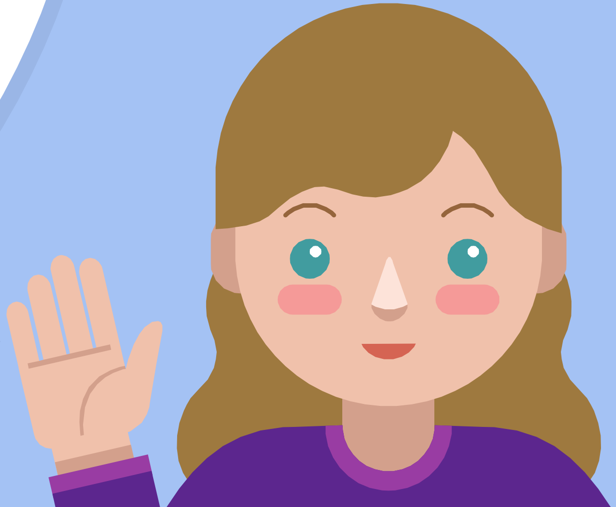
👍 Zero or negative : good payment

% Good payment	**Loan rating**
p <= 25	VERY BAD
25 < p <= 50	BAD
50 < p <=75	MODERATE
P > 75	GOOD

Due date	***Payment date***	***Latency (days)***	***Payment status***
15-Jan-2025	8-Jan-2025	-7	Good
15-Feb-2025	19-Feb-2025	4	Bad
15-Mar-2025	12-Mar-2025	-3	Good
15-Apr-2025	15-Apr-2025	0	Good
15-May-2025	17-May-2025	2	Bad

Good payment = 60%
Rating : ***MODERATE***

ksqlDB & Schema Registry

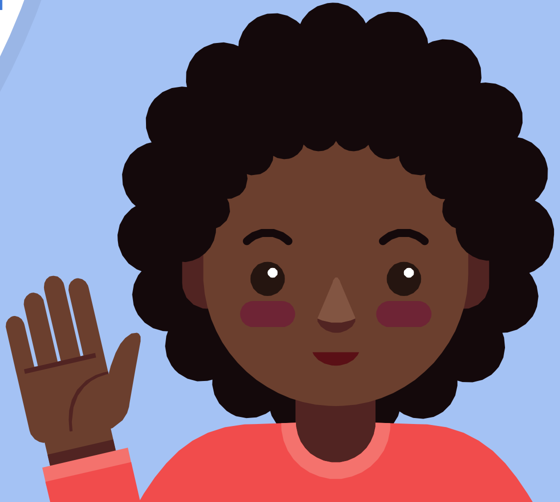


KsqlDB & Schema Registry

- Direct integration
- Useful for avro
- No need to manually type schema
- Automatic schema generation
- Convert format
- Use avro schema from previous lesson

Avro on ksqlDB

Writing Avro Schema



Avro-JSON Conversion

ksqlDB & Kafka Connect

ksqlDB Java Client