Lab: Filter a stream of events

Problem Statement:

How do you filter messages in a Kafka topic to contain only those that you're interested in?

Example use case:

Consider a topic with events that represent book publications. In this lab, we'll write a program that creates a new topic which only contains the events for a particular author.

Hands-on code example:

Run it

1. Prerequisites
2. Initialize the project
3. Get Confluent Platform
4. Write the program interactively using the CLI
5. Write your statements to a file

Test it

1. Create the test data
2. Invoke the tests

Run it

Prerequisites

This lab installs Confluent Platform using Docker. Before proceeding:

- Connect with lab environment VM using SSH:

  `ssh USERNAME@YOUR_VM_DNS.courseware.io`

  - **Username:** Will be provided by Instructor.
  - **Password:** Will be provided by Instructor.

- Verify that Docker is set up properly by ensuring no errors are output when you run `docker info` and `docker compose version` on the command line.

Initialize the project

To get started, make a new directory anywhere you'd like for this project:

```
mkdir filter-events && cd filter-events
```

Then make the following directories to set up its structure:

```
mkdir src test
```

Get Confluent Platform

Next, create the following `docker-compose.yml` file to obtain Confluent Platform:

```
---
version: '2'

services:
  zookeeper:
    image: confluentinc/cp-zookeeper:7.3.0
    hostname: zookeeper
    container_name: zookeeper
    ports:
      - "2181:2181"
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000

  broker:
    image: confluentinc/cp-kafka:7.3.0
    hostname: broker
    container_name: broker
    depends_on:
      - zookeeper
    ports:
      - "29092:29092"
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: 'zookeeper:2181'
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
      KAFKA_ADVERTISED_LISTENERS:
PLAINTEXT://broker:9092,PLAINTEXT_HOST://localhost:29092
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 1
      KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 1
      KAFKA_GROUP_INITIAL_REBALANCE_DELAY_MS: 0

  schema-registry:
    image: confluentinc/cp-schema-registry:7.3.0
    hostname: schema-registry
    container_name: schema-registry
    depends_on:
      - broker
    ports:
      - "8081:8081"
    environment:
      SCHEMA_REGISTRY_HOST_NAME: schema-registry
      SCHEMA_REGISTRY_KAFKASTORE_BOOTSTRAP_SERVERS: 'broker:9092'

  ksqldb-server:
    image: confluentinc/ksqldb-server:0.28.2
    hostname: ksqldb-server
    container_name: ksqldb-server
```

```
    depends_on:
      - broker
      - schema-registry
    ports:
      - "8088:8088"
    environment:
      KSQL_CONFIG_DIR: "/etc/ksqldb"
      KSQL_LOG4J_OPTS: "-Dlog4j.configuration=file:/etc/ksqldb/log4j.properties"
      KSQL_BOOTSTRAP_SERVERS: "broker:9092"
      KSQL_HOST_NAME: ksqldb-server
      KSQL_LISTENERS: "http://0.0.0.0:8088"
      KSQL_CACHE_MAX_BYTES_BUFFERING: 0
      KSQL_KSQL_SCHEMA_REGISTRY_URL: "http://schema-registry:8081"

  ksqldb-cli:
    image: confluentinc/ksqldb-cli:0.28.2
    container_name: ksqldb-cli
    depends_on:
      - broker
      - ksqldb-server
    entrypoint: /bin/sh
    environment:
      KSQL_CONFIG_DIR: "/etc/ksqldb"
    tty: true
    volumes:
      - ./src:/opt/app/src
      - ./test:/opt/app/test
```

And launch it by running:

```
docker compose up -d
```

Write the program interactively using the CLI

To begin developing interactively, open up the ksqlDB CLI:

```
docker exec -it ksqldb-cli ksql http://ksqldb-server:8088
```

First, you'll need to create a Kafka topic and stream to represent the publications. The following creates both in one shot:

```
CREATE STREAM all_publications (bookid BIGINT KEY, author VARCHAR, title VARCHAR)
    WITH (kafka_topic = 'publication_events', partitions = 1, value_format = 'avro');
```

Then produce the following events to the stream:

```
INSERT INTO all_publications (bookid, author, title) VALUES (1, 'C.S. Lewis', 'The
Silver Chair');
INSERT INTO all_publications (bookid, author, title) VALUES (2, 'George R. R. Martin',
'A Song of Ice and Fire');
INSERT INTO all_publications (bookid, author, title) VALUES (3, 'C.S. Lewis',
'Perelandra');
INSERT INTO all_publications (bookid, author, title) VALUES (4, 'George R. R. Martin',
```

```
'Fire & Blood');
INSERT INTO all_publications (bookid, author, title) VALUES (5, 'J. R. R. Tolkien',
'The Hobbit');
INSERT INTO all_publications (bookid, author, title) VALUES (6, 'J. R. R. Tolkien',
'The Lord of the Rings');
INSERT INTO all_publications (bookid, author, title) VALUES (7, 'George R. R. Martin',
'A Dream of Spring');
INSERT INTO all_publications (bookid, author, title) VALUES (8, 'J. R. R. Tolkien',
'The Fellowship of the Ring');
INSERT INTO all_publications (bookid, author, title) VALUES (9, 'George R. R. Martin',
'The Ice Dragon');
```

Now that you have stream with some events in it, let's read them out. The first thing to do is set the following properties to ensure that you're reading from the beginning of the stream:

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

Let's find all of the books written by George R. R. Martin. Issue the following transient push query. This will block and continue to return results until its limit is reached or you tell it to stop.

```
SELECT * FROM all_publications WHERE author = 'George R. R. Martin' EMIT CHANGES LIMIT
4;
```

This should yield the following output:

```
+------------------------+------------------------+------------------------+
|BOOKID                  |AUTHOR                  |TITLE                   |
+------------------------+------------------------+------------------------+
|2                       |George R. R. Martin     |A Song of Ice and Fire  |
|4                       |George R. R. Martin     |Fire & Blood            |
|7                       |George R. R. Martin     |A Dream of Spring       |
|9                       |George R. R. Martin     |The Ice Dragon          |
Limit Reached
Query terminated
```

Since the output looks right, the next step is to make the query continuous. Issue the following to create a new stream that is continuously populated by its query:

```
CREATE STREAM george_martin WITH (kafka_topic = 'george_martin_books') AS
    SELECT *
      FROM all_publications
      WHERE author = 'George R. R. Martin';
```

To check that it's working, print out the contents of the output stream's underlying topic:

```
PRINT george_martin_books FROM BEGINNING LIMIT 4;
```

This should yield the following output:

```
Key format: KAFKA_BIGINT or KAFKA_DOUBLE or KAFKA_STRING
Value format: AVRO or KAFKA_STRING
rowtime: 2020/06/02 14:36:36.846 Z, key: 2, value: {"AUTHOR": "George R. R. Martin",
"TITLE": "A Song of Ice and Fire"}, partition: 0
```

```
rowtime: 2020/06/02 14:36:37.057 Z, key: 4, value: {"AUTHOR": "George R. R. Martin",
"TITLE": "Fire & Blood"}, partition: 0
rowtime: 2020/06/02 14:36:37.350 Z, key: 7, value: {"AUTHOR": "George R. R. Martin",
"TITLE": "A Dream of Spring"}, partition: 0
rowtime: 2020/06/02 14:36:37.541 Z, key: 9, value: {"AUTHOR": "George R. R. Martin",
"TITLE": "The Ice Dragon"}, partition: 0
Topic printing ceased
```

Write your statements to a file

Now that you have a series of statements that's doing the right thing, the last step is to put them into a file so that they can be used outside the CLI session. Create a file at `src/statements.sql` with the following content:

```
CREATE STREAM all_publications (bookid BIGINT KEY, author VARCHAR, title VARCHAR)
    WITH (kafka_topic = 'publication_events', partitions = 1, value_format = 'avro');

CREATE STREAM george_martin WITH (kafka_topic = 'george_martin_books') AS
    SELECT *
      FROM all_publications
      WHERE author = 'George R. R. Martin';
```

Test it

Create the test data

Create a file at `test/input.json` with the inputs for testing:

```
{
  "inputs": [
    {
      "topic": "publication_events",
      "key": 1,
      "value": {
        "author": "C.S. Lewis",
        "title": "The Silver Chair"
      }
    },
    {
      "topic": "publication_events",
      "key": 2,
      "value": {
        "author": "George R. R. Martin",
        "title": "A Song of Ice and Fire"
      }
    },
    {
      "topic": "publication_events",
      "key": 3,
      "value": {
        "author": "C.S. Lewis",
        "title": "Perelandra"
```

```
      }
    },
    {
      "topic": "publication_events",
      "key": 4,
      "value": {
        "author": "George R. R. Martin",
        "title": "Fire & Blood"
      }
    },
    {
      "topic": "publication_events",
      "key": 5,
      "value": {
        "author": "J. R. R. Tolkien",
        "title": "The Hobbit"
      }
    },
    {
      "topic": "publication_events",
      "key": 6,
      "value": {
        "author": "J. R. R. Tolkien",
        "title": "The Lord of the Rings"
      }
    },
    {
      "topic": "publication_events",
      "key": 7,
      "value": {
        "author": "George R. R. Martin",
        "title": "A Dream of Spring"
      }
    },
    {
      "topic": "publication_events",
      "key": 8,
      "value": {
        "author": "J. R. R. Tolkien",
        "title": "The Fellowship of the Ring"
      }
    },
    {
      "topic": "publication_events",
      "key": 9,
      "value": {
        "author": "George R. R. Martin",
        "title": "The Ice Dragon"
      }
    }
  ]
}
```

Similarly, create a file at `test/output.json` with the expected outputs:

```
{
  "outputs": [
    {
      "topic": "george_martin_books",
      "key": 2,
      "value": {
        "AUTHOR": "George R. R. Martin",
        "TITLE": "A Song of Ice and Fire"
      }
    },
    {
      "topic": "george_martin_books",
      "key": 4,
      "value": {
        "AUTHOR": "George R. R. Martin",
        "TITLE": "Fire & Blood"
      }
    },
    {
      "topic": "george_martin_books",
      "key": 7,
      "value": {
        "AUTHOR": "George R. R. Martin",
        "TITLE": "A Dream of Spring"
      }
    },
    {
      "topic": "george_martin_books",
      "key": 9,
      "value": {
        "AUTHOR": "George R. R. Martin",
        "TITLE": "The Ice Dragon"
      }
    }
  ]
}
```

Invoke the tests

Lastly, invoke the tests using the test runner and the statements file that you created earlier:

```
docker exec ksqldb-cli ksql-test-runner -i /opt/app/test/input.json -s
/opt/app/src/statements.sql -o /opt/app/test/output.json
```

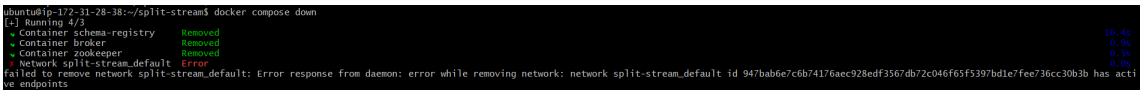
Which should pass:

```
  >>> Test passed!
```

Cleanup Resources

Delete all the resources by running following command in the `docker-compose.yml` file directory from the terminal:

```
docker compose down
```



Note: If you get above error while running above command. Manually stop the containers and run `docker compose down` again. **Do not delete kafkanew container**.