



Ministry of Higher Education and Research
Higher School of Computer Science 08 May 1945 - Sidi Bel Abbas
Second Year Second Cycle - Artificial Intelligence and Data Science

LAB 07 : STREAM PROCESSING

Presented By : FELLAH Abdelnour.

Date: May 10, 2024

1 INTRODUCTION

Stream processing is a powerful paradigm for handling real-time data streams, enabling organizations to derive valuable insights and make informed decisions. In this lab, we explore the use of ksqlDB, a streaming SQL engine for Apache Kafka, to perform stream processing tasks.

2 SETUP

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

3 PART 01 : KSQLDB QUERIES

QUESTION 01 :

```
ksql> CREATE TYPE season_length AS STRUCT<season_id INT, episode_count INT>;
ksql> SHOW TYPES;
```

```
ksql> SHOW TYPES;
Type Name      | Schema
-----
SEASON_LENGTH | STRUCT<SEASON_ID INTEGER, EPISODE_COUNT INTEGER>
```

QUESTION 02 :

The appropriate collection type for titles and production_changes :

- titles : tables, they are mutable.
- production_changes : stream because they are immutable events.

QUESTION 03 :

```
ksql> CREATE STREAM productionChanges (
    rowkey VARCHAR KEY, title_id INT, change_type VARCHAR,
    before SEASON_LENGTH, after SEASON_LENGTH, created_at VARCHAR)
WITH (
    kafka_topic='production_changes',
    value_format='json',
    partitions=4,
    TIMESTAMP='created_at',
    TIMESTAMP_FORMAT='yyyy-MM-dd HH:mm:ss')
```

```
);
```

```
ksql> CREATE TABLE title (id INT PRIMARY KEY,title VARCHAR)
WITH (KAFKA.TOPIC='titles',PARTITIONS='4',VALUE_FORMAT='JSON');
```

```
ksql> LIST STREAMS;
Stream Name      | Kafka Topic      | Key Format | Value Format | Windowed
-----
KSQL_PROCESSING_LOG | default_ksql_processing_log | KAFKA      | JSON        | false
PRODUCTIONCHANGES | production_changes | KAFKA      | JSON        | false
SEASON_LENGTH_CHANGES | season_length_changes | KAFKA      | JSON        | false
SEASON_LENGTH_CHANGES_ENRICHED | season_length_changes_enriched | KAFKA      | JSON        | false

ksql> LIST TABLES;
Table Name      | Kafka Topic      | Key Format | Value Format | Windowed
-----
SEASON_LENGTH_CHANGE_COUNTS | season_length_change_counts | KAFKA      | JSON        | true
TITLE            | titles           | KAFKA      | JSON        | false

ksql>
```

QUESTION 04 :

```
ksql> INSERT INTO title (id, title) VALUES (1,'title1');
ksql> INSERT INTO productionChanges (rowkey, title_id,change.type,before,after,created.at)
VALUES ('rowkey1',1,'season.length',STRUCT(season.id := 1,episode.count := 12),
STRUCT(season.id := 1,episode.count := 8),'2021-02-08 11:30:00');
```

```
ksql> SELECT * FROM QUERYABLE_TITLE;
+-----+-----+
| ID | TITLE |
+-----+-----+
| 1 | title1 |
+-----+-----+
Query terminated
ksql> SELECT * FROM productionChanges;
+-----+-----+-----+-----+-----+-----+
| ROWKEY | TITLE_ID | CHANGE_TYPE | BEFORE | AFTER | CREATED_AT |
+-----+-----+-----+-----+-----+-----+
| rowkey1 | 1 | season_length | {SEASON_ID=1, EPISODE_COUNT=12} | {SEASON_ID=1, EPISODE_COUNT=8} | 2021-02-08 11:30:00 |
| rowkey1 | 1 | season_length | {SEASON_ID=1, EPISODE_COUNT=12} | {SEASON_ID=1, EPISODE_COUNT=8} | 2021-02-08 11:30:00 |
+-----+-----+-----+-----+-----+-----+
Query Completed
Query terminated
ksql>
```

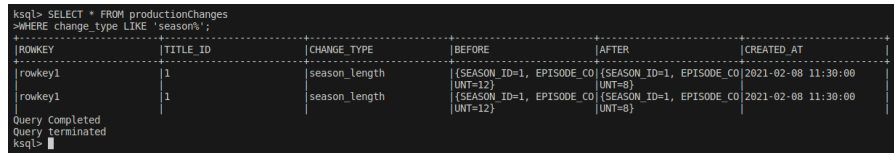
QUESTION 05 :

```
ksql> SET 'auto.offset.reset' = 'earliest';
ksql> SELECT * FROM productionChanges
WHERE created_at < '2023-04-14 à 12:00:00'
EMIT CHANGES;
```

```
ksql> SELECT * FROM productionChanges
WHERE created_at < '2023-04-14 à 12:00:00';
+-----+-----+-----+-----+-----+-----+
| ROWKEY | TITLE_ID | CHANGE_TYPE | BEFORE | AFTER | CREATED_AT |
+-----+-----+-----+-----+-----+-----+
| rowkey1 | 1 | season_length | {SEASON_ID=1, EPISODE_COUNT=12} | {SEASON_ID=1, EPISODE_COUNT=8} | 2021-02-08 11:30:00 |
| rowkey1 | 1 | season_length | {SEASON_ID=1, EPISODE_COUNT=12} | {SEASON_ID=1, EPISODE_COUNT=8} | 2021-02-08 11:30:00 |
+-----+-----+-----+-----+-----+-----+
Query Completed
Query terminated
ksql>
```

QUESTION 06 :

```
ksql> SELECT * FROM productionChanges
      WHERE change_type LIKE 'season%'
      EMIT CHANGES;
```



The screenshot shows the output of a KSQL query. It displays a table with columns: ROWKEY, TITLE_ID, CHANGE_TYPE, BEFORE, AFTER, and CREATED_AT. There are two rows of data, both for rowkey1 and title_id 1, showing a change in season_length from 12 to 8. The BEFORE column shows the previous state with SEASON_ID=1, EPISODE_COUNT=12, and UNT=12. The AFTER column shows the new state with SEASON_ID=1, EPISODE_COUNT=8, and UNT=8. The CREATED_AT column shows the timestamp 2021-02-08 11:30:00. Below the table, it says 'Query Completed' and 'Query Terminated'.

ROWKEY	TITLE_ID	CHANGE_TYPE	BEFORE	AFTER	CREATED_AT
rowkey1	1	season_length	{SEASON_ID=1, EPISODE_COUNT=12, UNT=12}	{SEASON_ID=1, EPISODE_COUNT=8, UNT=8}	2021-02-08 11:30:00
rowkey1	1	season_length	{SEASON_ID=1, EPISODE_COUNT=12, UNT=12}	{SEASON_ID=1, EPISODE_COUNT=8, UNT=8}	2021-02-08 11:30:00

Query Completed
Query Terminated
ksql>

QUESTION 07 :

```
ksql> CREATE STREAM season_length_changes
      WITH (KAFKA.TOPIC='season_length_changes',PARTITIONS='4',VALUE_FORMAT='JSON',REPLICAS = '1')
      AS
      SELECT rowkey,title_id,createdat,
      IFNULL(after->season.id, before->season.id) as season_id,
      before->episode.count as old_episode_count,
      after->episode.count as new_episode_count
      FROM productionChanges
      WHERE change_type = 'season_length';
```

QUESTION 08 :

```
ksql> SELECT title
      FROM season_length_changes s
      INNER JOIN title t
      ON CAST(s.title_id AS INT) = t.id
      EMIT CHANGES ;
```

QUESTION 09 :

```
ksql> CREATE STREAM season_length_changes_enriched
WITH (
    KAFKA.TOPIC='season_length_changes_enriched',
    PARTITIONS='4',
    VALUE.FORMAT='JSON',
    REPLICAS = '1',
    TIMESTAMP='created_at',
    TIMESTAMP.FORMAT='yyyy-MM-dd HH:mm:ss'
) AS
SELECT rowkey,title_id,created_at,old_episode_count,new_episode_count,title_id
FROM season_length_changes s
INNER JOIN title t
ON CAST(s.title_id AS INT) = t.id;
```

QUESTION 10 :

```
ksql> CREATE TABLE season_length_change_counts
WITH (
    KAFKA.TOPIC='season_length_change_counts',
    PARTITIONS='4',
    VALUE.FORMAT='JSON',
    REPLICAS = '1'
) AS
SELECT title_id,COUNT(*) AS change_count,LATEST_BY_OFFSET(new_episode_count) AS latest_episode_count
FROM season_length_changes_enriched
WINDOW TUMBLING (SIZE 1 HOUR)
GROUP BY title_id;
```

4 PART 02 : APACHE KAFKA

PROGRAM :

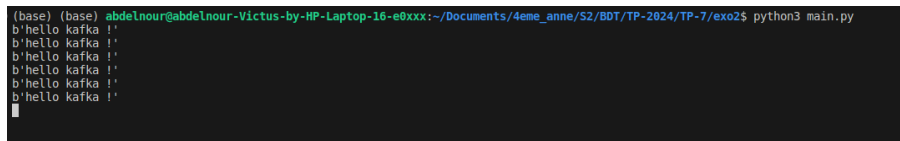
```
from kafka import KafkaProducer, KafkaConsumer

producer = KafkaProducer(bootstrap_servers=['localhost:29092'], api_version=(2, 0, 2))
producer.send('topic-example', bytes("hello kafka !", 'UTF-8'))

consumer = KafkaConsumer('topic-example', bootstrap_servers=['localhost:29092'],
auto_offset_reset='earliest')

for message in consumer:
    print(message.value)
```

OUTPUT :

A terminal window screenshot showing the output of a Kafka consumer. The prompt is '(base) (base) abdelnour@abdelnour-Victus-by-HP-Laptop-16-e0xxx:~/Documents/4eme_anne/S2/BDT/TP-2024/TP-7/exo2\$'. The command 'python3 main.py' has been executed. The output consists of six lines, each showing a byte string: 'b\'hello kafka !\''.

```
(base) (base) abdelnour@abdelnour-Victus-by-HP-Laptop-16-e0xxx:~/Documents/4eme_anne/S2/BDT/TP-2024/TP-7/exo2$ python3 main.py
b'hello kafka !'
b'hello kafka !'
b'hello kafka !'
b'hello kafka !'
b'hello kafka !'
b'hello kafka !'
```

5 CONCLUSION

Conclusion:

In this lab, we have explored stream processing using ksqlDB and Apache Kafka. We have learned how to set up a streaming data pipeline, write streaming SQL queries with ksqlDB, Through this process, we have gained valuable insights into the capabilities of stream processing and its potential applications.