Spark Streaming Demo

- Ullas
- Sathvik

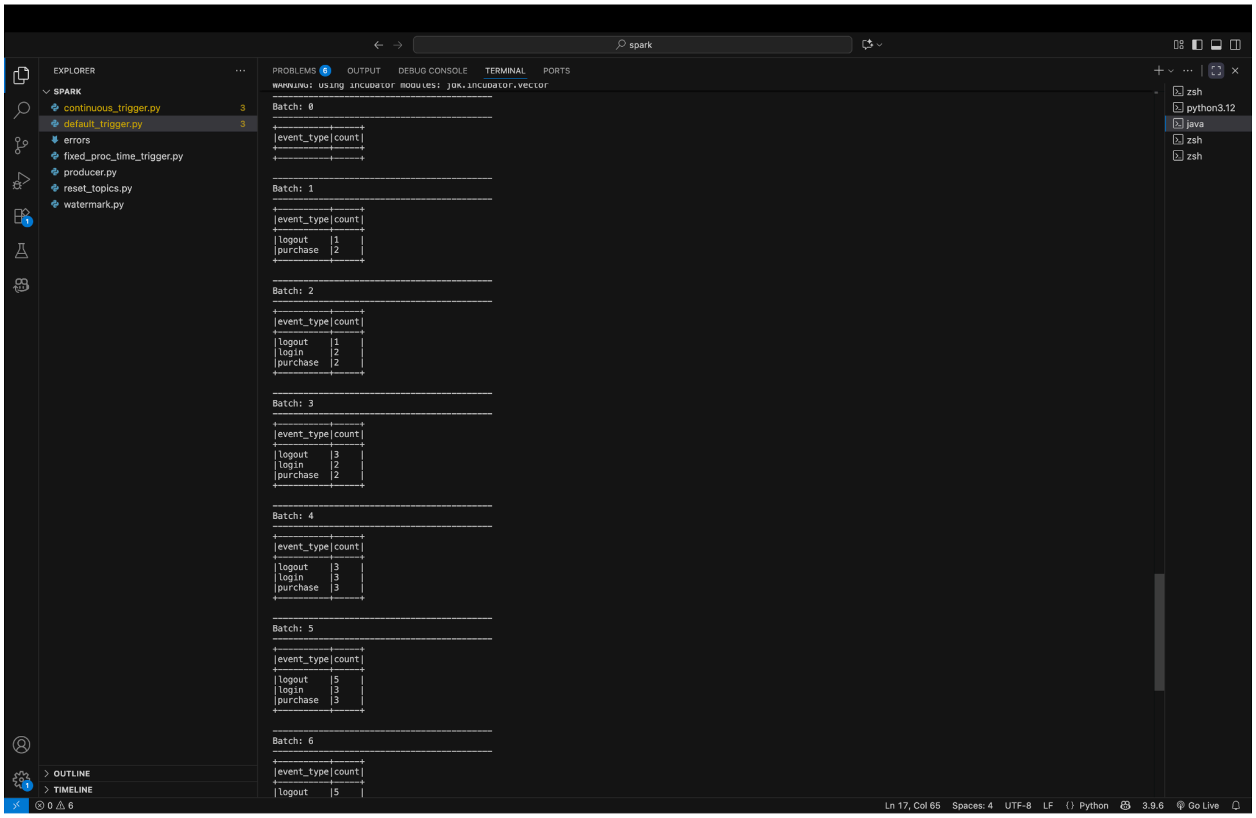
Introduction

- This document outlines a demonstration of Spark Structured Streaming. The objective is to illustrate how Spark handles live data streams from Apache Kafka and to demonstrate the different processing modes available.
- The demo consists of two main parts:
    - Trigger Modes**:** Processing a single Kafka data stream using Spark's default, processing-time, and continuous triggers.
    - Watermarking**:** Implementing a stream-stream join to show how Spark manages state and handles late data.

Part 1: Kafka Streaming Workflow: This workflow involves a data producer sending events to Kafka, which are then consumed by Spark using different processing strategies.

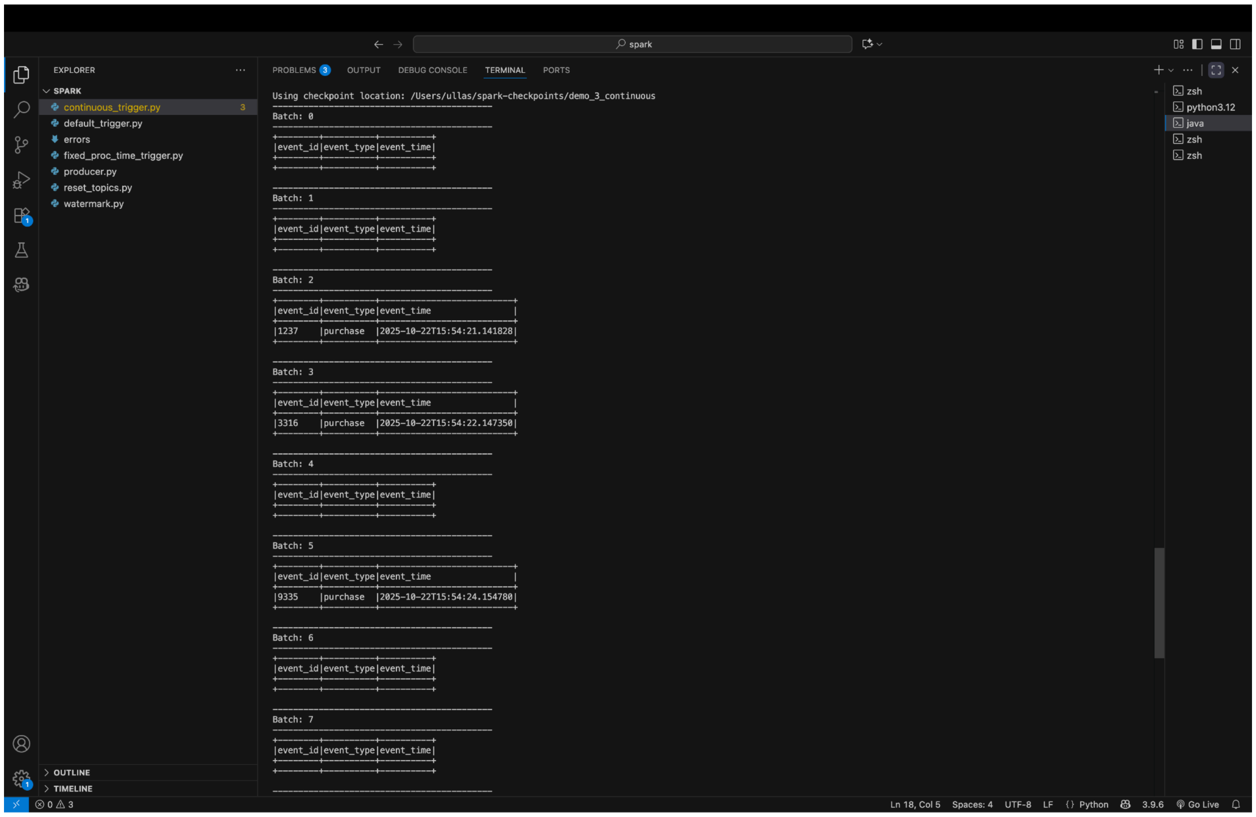
- **Workflow Components**
    - reset_topics.py: This utility script is run first to ensure a clean environment. It deletes the 'Demo' topic from Kafka and immediately recreates it, so each demo run starts from a fresh, empty state.
    - producer.py: This script acts as the live data source. It runs in a continuous loop, generating a new JSON event (e.g., 'login', 'purchase') and sending it to the 'Demo' Kafka topic once per second.
- **Spark Processing Methods (Triggers)**

- Three separate Spark scripts are used to consume the data from the producer.py script, each demonstrating a different processing trigger.
- **Default Trigger** : This script uses Spark's default "micro-batch" processing mode. It processes all available new data in one batch. As soon as that batch is complete, Spark immediately kicks off a new one, attempting to process the data as fast as possible with low latency.
- **Processing Time Trigger** : This script also uses the micro-batch engine but adds a fixed schedule. The trigger is set to **trigger(processingTime="5 seconds")**. This forces Spark to run a batch, wait, and then run the next batch exactly five seconds later. This approach is useful for creating predictable, periodic updates.
- **Continuous Trigger** : This script uses a different, low-latency processing engine that operates row-by-row, not in batches. Its goal is to achieve near-instantaneous processing **(approx. 1ms)**. This demo highlights a key trade-off: this engine does not support stateful operations like aggregations (groupBy). This method is not suitable for complex operations and hence it is not used generally. It works well with simple filter conditions or other such cases where we can work on individual rows. In such cases it works very well and provides very low latency depicting actual streaming. In our example we have used a simple filter condition , where only **“purchase”** events are shown.



```
WARNING: Using incubator modules: jdk.incubator.vector
-------------------------------------------
Batch: 0
-------------------------------------------
+----------+-----+
|event_type|count|
+----------+-----+
+----------+-----+

-------------------------------------------
Batch: 1
-------------------------------------------
+----------+-----+
|event_type|count|
+----------+-----+
|logout    |1    |
|purchase  |2    |
+----------+-----+

-------------------------------------------
Batch: 2
-------------------------------------------
+----------+-----+
|event_type|count|
+----------+-----+
|logout    |1    |
|login     |2    |
|purchase  |2    |
+----------+-----+

-------------------------------------------
Batch: 3
-------------------------------------------
+----------+-----+
|event_type|count|
+----------+-----+
|logout    |3    |
|login     |2    |
|purchase  |2    |
+----------+-----+

-------------------------------------------
Batch: 4
-------------------------------------------
+----------+-----+
|event_type|count|
+----------+-----+
|logout    |3    |
|login     |3    |
|purchase  |3    |
+----------+-----+

-------------------------------------------
Batch: 5
-------------------------------------------
+----------+-----+
|event_type|count|
+----------+-----+
|logout    |5    |
|login     |3    |
|purchase  |3    |
+----------+-----+

-------------------------------------------
Batch: 6
-------------------------------------------
+----------+-----+
|event_type|count|
+----------+-----+
|logout    |5    |
```

Default Processing



```
Using checkpoint location: /Users/ullas/spark-checkpoints/demo_3_continuous
-------------------------------------------
Batch: 0
-------------------------------------------
+--------+----------+----------+
|event_id|event_type|event_time|
+--------+----------+----------+
+--------+----------+----------+

-------------------------------------------
Batch: 1
-------------------------------------------
+--------+----------+----------+
|event_id|event_type|event_time|
+--------+----------+----------+
+--------+----------+----------+

-------------------------------------------
Batch: 2
-------------------------------------------
+--------+----------+--------------------------+
|event_id|event_type|event_time                |
+--------+----------+--------------------------+
|1237    |purchase  |2025-10-22T15:54:21.141828|
+--------+----------+--------------------------+

-------------------------------------------
Batch: 3
-------------------------------------------
+--------+----------+--------------------------+
|event_id|event_type|event_time                |
+--------+----------+--------------------------+
|3316    |purchase  |2025-10-22T15:54:22.147350|
+--------+----------+--------------------------+

-------------------------------------------
Batch: 4
-------------------------------------------
+--------+----------+----------+
|event_id|event_type|event_time|
+--------+----------+----------+
+--------+----------+----------+

-------------------------------------------
Batch: 5
-------------------------------------------
+--------+----------+--------------------------+
|event_id|event_type|event_time                |
+--------+----------+--------------------------+
|9335    |purchase  |2025-10-22T15:54:24.154780|
+--------+----------+--------------------------+

-------------------------------------------
Batch: 6
-------------------------------------------
+--------+----------+----------+
|event_id|event_type|event_time|
+--------+----------+----------+
+--------+----------+----------+

-------------------------------------------
Batch: 7
-------------------------------------------
+--------+----------+----------+
|event_id|event_type|event_time|
+--------+----------+----------+
+--------+----------+----------+

-------------------------------------------
```

Continuous Processing

Part 2: Watermarking Demo(watermark.py)

- This final demonstration explains watermarking by joining two independent data streams.
- This code is independent and is not related to Part-1.

The Problem: Infinite State

When joining two live streams ("ad impressions" and "ad clicks"), Spark must keep the first event (the impression) in its memory or state while it waits for the second event (the click).

If Spark waits indefinitely, its state will grow forever as it stores all unmatched impressions, eventually causing the job to fail from lack of memory. Watermarking is the mechanism to prevent this.

How Watermarking Actually Works (State Management)

It is important to distinguish between the **join condition** and the **watermark**.

1. **The Join Condition:** This is the logic defined in the expr(...) block. In this demo, it's the 30-second rule (**click_time <= impression_time + interval 30 seconds**). This defines what a correct match is.
2. **The Watermark:** This is the *engine logic* defined with .withWatermark(...). It is a promise to Spark about how late data can be (e.g., "10 seconds" for impressions). Its only purpose is to tell Spark when it is safe to **clean up old state** (delete old impressions from memory).

Spark tracks the "latest time" it has seen in each stream and subtracts the watermark delay. It then calculates a **Global Watermark**, which is the **minimum (earliest) time** of all stream watermarks.

Spark uses this Global Watermark to clean its state. It will purge any old event from its memory whose "join window" is entirely older than the watermark.

Demo Explained (with State Cleanup)

This demo joins an 'impression' stream with a 'click' stream. A **valid join** is defined by the 30-second event time rule.

A Note on "Event Time" (Dummy Timestamps) In this demo, the timestamps (like 18:30:00) are manually typed. These are "dummy" timestamps used to simulate **"Event Time"**, the time the event actually happened in the real world.

This is different from **"Processing Time,"** which is the time Spark's engine receives the data. Watermarking must use "Event Time" to correctly handle data that might be delayed in the network and arrive out of order. By using these dummy timestamps, we can simulate complex scenarios, such as a late-arriving event.

Example 1: The Valid Join (Testing the 30-second rule)

1. An impression for ad_123 is sent at time 18:30:00.
2. A click for ad_123 is sent at time 18:30:05.
3. Result: This join is successful and appears in the output. The 5-second delay is inside the 30-second valid window.

Example 2: The Invalid Join (Testing the 30-second rule)

1. An impression for ad_456 is sent at time 18:30:10.
2. A click for ad_456 is sent at time 18:30:50.
3. Result: Nothing appears in the output. The 40-second delay is outside the 30-second rule. Spark correctly identifies this as an invalid match.

Example 3: The "Late but Valid" Join (Proving the Watermark)

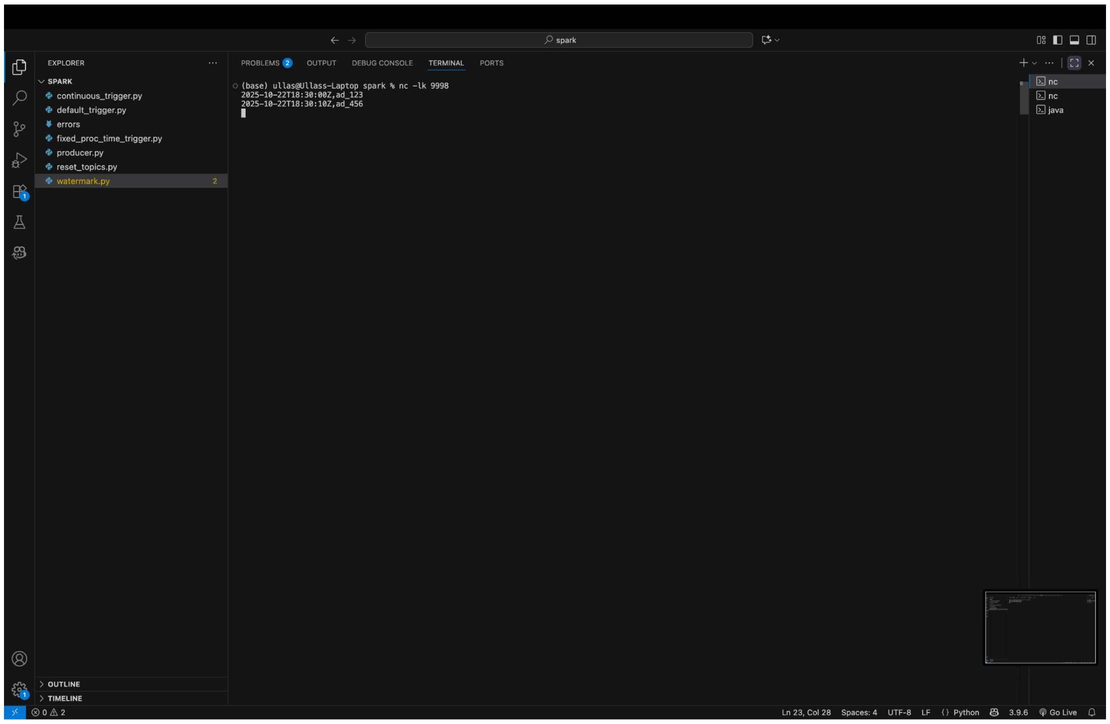
This test proves that the watermark is actively cleaning up memory.

1. Step 1**:** An impression for ad_123 is sent at time 18:30:00. (Spark now stores ad_123 in its state, waiting for a click).
2. Step 2**:** A much later impression for ad_789 is sent at 18:32:00.
3. What Spark Does: This new 18:32:00 event pushes Spark's "latest time" forward. The Global Watermark now moves far past 18:30:30 (the end of ad_123's 30-second join window). Spark

identifies that ad_123 is now too old to ever be matched and drops it from its state to save memory.
4. Step 4**:** A click for ad_123 is sent with time 18:30:20.
5. Result: No join appears in the output.

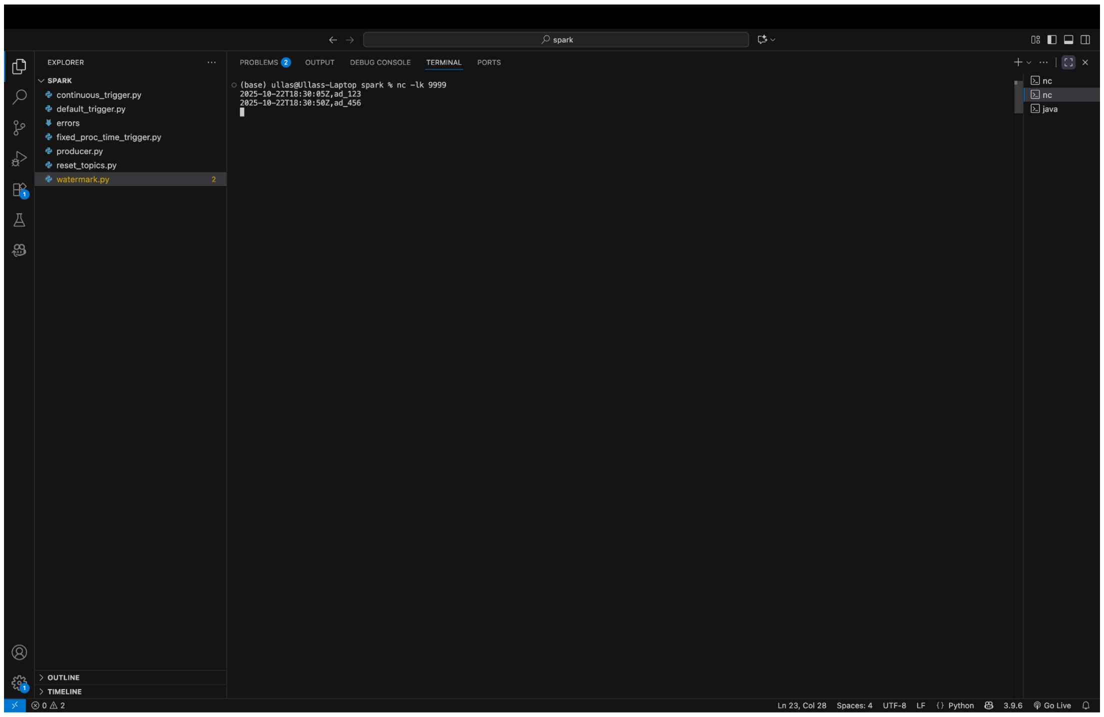
Reason: Even though the click was "valid" by the 30-second rule (18:30:20 is within 30s of 18:30:00), it arrived *too late*. The watermark had already advanced, and Spark had already **deleted the matching impression** from its state. The click arrived, but the impression it was looking for was already gone. This demonstrates the watermark's true purpose: state cleanup.

.



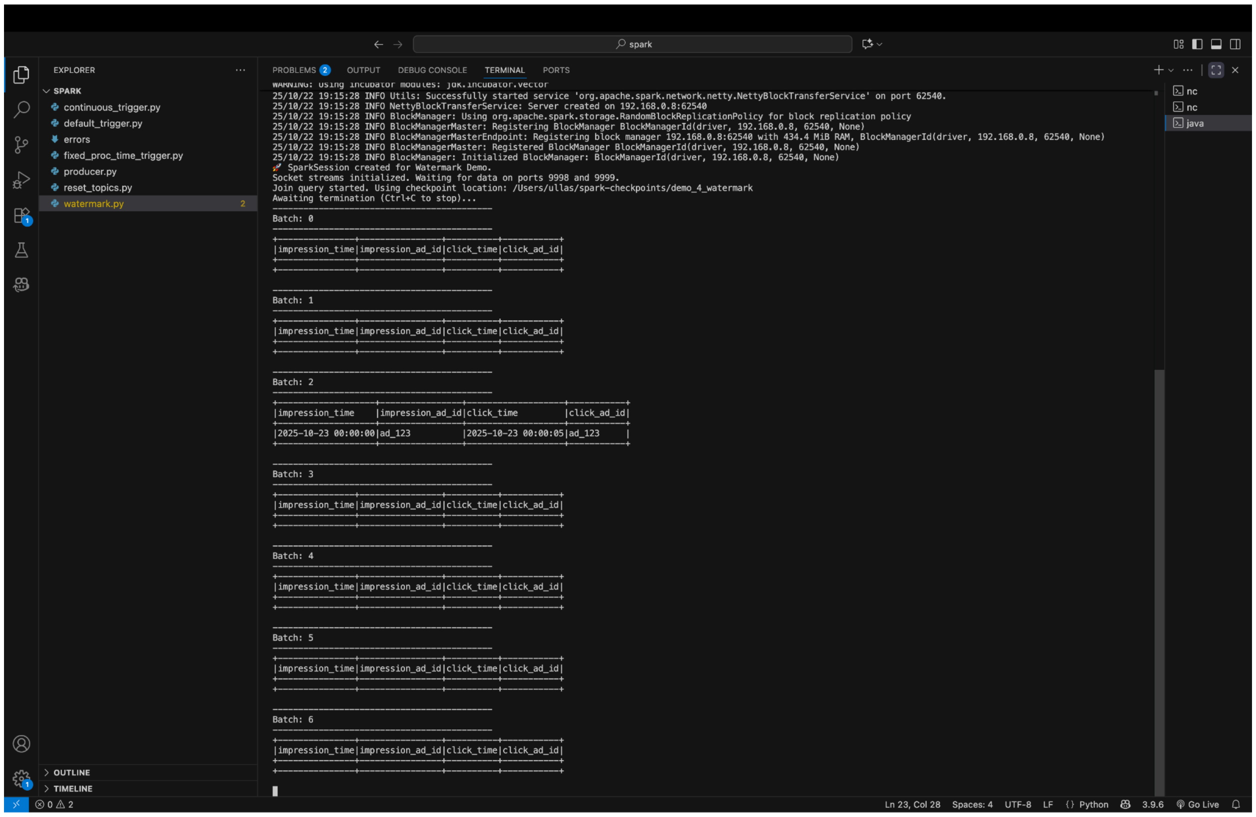
```
(base) ullas@Ullass-Laptop spark % nc -lk 9998
2025-10-22T18:30:00Z,ad_123
2025-10-22T18:30:10Z,ad_456
```

Port 9998(Ad Impressions Port)



```
(base) ullas@Ullass-Laptop spark % nc -lk 9999
2025-10-22T18:30:05Z,ad_123
2025-10-22T18:30:50Z,ad_456
```

Port 9999(Click events Port)



```
WARNING: Using incubator modules: jdk.incubator.vector
25/10/22 19:15:28 INFO Utils: Successfully started service 'org.apache.spark.network.netty.NettyBlockTransferService' on port 62540.
25/10/22 19:15:28 INFO NettyBlockTransferService: Server created on 192.168.0.8:62540
25/10/22 19:15:28 INFO BlockManager: Using org.apache.spark.storage.RandomBlockReplicationPolicy for block replication policy
25/10/22 19:15:28 INFO BlockManagerMaster: Registering BlockManager BlockManagerId(driver, 192.168.0.8, 62540, None)
25/10/22 19:15:28 INFO BlockManagerMasterEndpoint: Registering block manager 192.168.0.8:62540 with 434.4 MiB RAM, BlockManagerId(driver, 192.168.0.8, 62540, None)
25/10/22 19:15:28 INFO BlockManagerMaster: Registered BlockManager BlockManagerId(driver, 192.168.0.8, 62540, None)
25/10/22 19:15:28 INFO BlockManager: Initialized BlockManager: BlockManagerId(driver, 192.168.0.8, 62540, None)
🚀 SparkSession created for Watermark Demo.
Socket streams initialized. Waiting for data on ports 9998 and 9999.
Join query started. Using checkpoint location: /Users/ullas/spark-checkpoints/demo_4_watermark
Awaiting termination (Ctrl+C to stop)...
-------------------------------------------
Batch: 0
-------------------------------------------
+---------------+----------------+----------+-----------+
|impression_time|impression_ad_id|click_time|click_ad_id|
+---------------+----------------+----------+-----------+
+---------------+----------------+----------+-----------+

-------------------------------------------
Batch: 1
-------------------------------------------
+---------------+----------------+----------+-----------+
|impression_time|impression_ad_id|click_time|click_ad_id|
+---------------+----------------+----------+-----------+
+---------------+----------------+----------+-----------+

-------------------------------------------
Batch: 2
-------------------------------------------
+-------------------+----------------+-------------------+-----------+
|impression_time    |impression_ad_id|click_time         |click_ad_id|
+-------------------+----------------+-------------------+-----------+
|2025-10-23 00:00:00|ad_123          |2025-10-23 00:00:05|ad_123     |
+-------------------+----------------+-------------------+-----------+

-------------------------------------------
Batch: 3
-------------------------------------------
+---------------+----------------+----------+-----------+
|impression_time|impression_ad_id|click_time|click_ad_id|
+---------------+----------------+----------+-----------+
+---------------+----------------+----------+-----------+

-------------------------------------------
Batch: 4
-------------------------------------------
+---------------+----------------+----------+-----------+
|impression_time|impression_ad_id|click_time|click_ad_id|
+---------------+----------------+----------+-----------+
+---------------+----------------+----------+-----------+

-------------------------------------------
Batch: 5
-------------------------------------------
+---------------+----------------+----------+-----------+
|impression_time|impression_ad_id|click_time|click_ad_id|
+---------------+----------------+----------+-----------+
+---------------+----------------+----------+-----------+

-------------------------------------------
Batch: 6
-------------------------------------------
+---------------+----------------+----------+-----------+
|impression_time|impression_ad_id|click_time|click_ad_id|
+---------------+----------------+----------+-----------+
+---------------+----------------+----------+-----------+

```

Watermark code output