Department of Computer Science and Engineering

B. Tech. CSE - 6th Semester

Jan – May 2025

UE22CS343BB3

DATABASE TECHNOLOGIES (DBT)

PROJECT REPORT

on

*Real Time Website Traffic Monitoring*

Submitted by: Team #: 5

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | Mahika Datta | PES2UG22CS292 | 6E | | Maitri Shekhda | PES2UG22CS294 | 6E | | Sri Sai Lakshmi | PES2UG22CS296 | 6E | |

*Class of* ***Dr. Saranya Rubini S***

|  |  |  |
| --- | --- | --- |
| *Table of Contents* | | |
| **Sl. No** | **Topic** | **Page No.** |
|  | Introduction | 3 |
|  | Installation of Software | 3 |
|  | Input Data   1. Source 2. Description | 4 |
|  | Streaming Mode Experiment   1. Description 2. Windows 3. Results | 5 |
|  | Batch Mode Experiment   1. Description 2. Data Size 3. Results | 12 |
|  | Comparison of Streaming & Batch Modes   1. Results and Discussion | 19 |
| 7. | Conclusion | 21 |

Introduction

In today's digital world, understanding user behavior on websites in real time is critical for improving engagement, optimizing content, and detecting anomalies such as traffic spikes or drop-offs. This project aims to design and implement a system for real-time website traffic monitoring using modern data streaming and storage technologies, while also enabling offline batch analytics for trend analysis and reporting.

The system captures user interactions such as page views, session durations, and time spent on pages from the dataset. These interactions are streamed in real time to a backend service using Kafka, and stored in a MySQL database. The real-time data is then analysed to identify immediate patterns such as page view distribution, engagement levels, and session categories within short time windows (e.g., 5-minute intervals).

In the batch processing phase, the same queries are executed on the accumulated dataset to uncover historical patterns, such as daily traffic trends, bounce and conversion rates, and engagement score statistics. This allows a direct comparison between real-time and batch analytics in terms of performance, accuracy, and use case relevance.

Overall, the project demonstrates how a combined streaming + batch architecture can be used for efficient, scalable website analytics.

Installation of Software

1. Kafka 3.9.0 (binary scala 2.12-3.9.0)
2. spark 3.5.5
3. Hadoop (winutils.exe 3.3.1)
4. Java 17 & Python

Input Data

1. Source

The dataset has been taken from Kaggle (Pre defined dataset):

<https://www.kaggle.com/datasets/anthonytherrien/website-traffic>

1. Description

The dataset simulates real-time website traffic logs and contains information about user interactions with a website. Each row in the dataset represents a user session, capturing key behavioural metrics like page views and time spent.

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| id | INT (PK) | Unique identifier for each session (auto-incremented) |
| page\_views | INT | Total number of pages viewed during the session |
| session\_duration | FLOAT | Duration of the session (in seconds) |
| time\_on\_page | FLOAT | Average time spent on each page (in seconds) |
| |  | | --- | | timestamp | | |  | | --- | | DATETIME | | |  | | --- | | Time when the session occurred (used for windowing) | |

This dataset serves as the source for both real-time and batch analytics in your project:

* In streaming mode, session data is sent in real-time via Kafka.
* In batch mode, the same session data is stored in MySQL and used for:
  + Trend analysis (quarterly)
  + Bounce and conversion rate analysis
  + Engagement score calculations
  + Session categorization (Short, Medium, Long)

Streaming Mode Experiment

1. Description

This project simulates a real-time website traffic monitoring system using Apache Kafka, Spark Structured Streaming, and MySQL.

* **Kafka Producers** (kafka\_producer.py, kafka\_producer\_streaming.py):

Simulate live user sessions by sending data such as **page views**, **session duration**, and **time on page** to Kafka topics. One producer reads from a CSV file, and another generates random real-time values.

from kafka import KafkaProducer

import json

import pandas as pd

from datetime import datetime, timedelta

import time

# Load dataset

df = pd.read\_csv('updated\_db.csv') # Update this with the actual path

print(df)

# Reset index to ensure sequential order

#df = df.reset\_index(drop=True)

# Kafka producer setup

producer = KafkaProducer(

bootstrap\_servers='localhost:9092',

value\_serializer=lambda v: json.dumps(v).encode('utf-8')

)

count\_page\_views = 0

count\_session\_duration = 0

count\_time\_on\_page = 0

# Loop through rows one at a time

for i, row in df.iterrows():

try:

# Use i (0, 1, 2, ...) to increment timestamp by 1 second

current\_timestamp = row["timestamp"]

# Create messages with the same timestamp

page\_views\_msg = {

"page\_views": int(row["Page Views"]),

"timestamp": current\_timestamp,

"record\_id": i+1

}

session\_duration\_msg = {

'session\_duration': float(row['Session Duration']),

'timestamp': current\_timestamp,

'record\_id': i+1

}

time\_on\_page\_msg = {

'time\_on\_page': float(row['Time on Page']),

'timestamp': current\_timestamp,

'record\_id': i+1

}

print("\n Sending messages:")

print("Page Views Msg: ", page\_views\_msg)

print("Session Duration Msg: ", session\_duration\_msg)

print("Time on Page Msg: ", time\_on\_page\_msg)

# Send messages to each topic

producer.send('topic\_pageviews', value=page\_views\_msg)

count\_page\_views += 1

producer.send('topic\_sessionduration', value=session\_duration\_msg)

count\_session\_duration += 1

producer.send('topic\_timeonpage', value=time\_on\_page\_msg)

count\_time\_on\_page += 1

# Optional delay to simulate streaming

time.sleep(0.05) # 10 ms

if i % 100 == 0:

print(f" Processed record {i} with timestamp: {current\_timestamp}")

except Exception as e:

print(f" Error at record {i}: {e}")

# Flush messages

producer.flush()

# Final report

print(f"\n Sent {count\_page\_views} messages to 'pageviews' topic")

print(f" Sent {count\_session\_duration} messages to 'sessionduration' topic")

print(f" Sent {count\_time\_on\_page} messages to 'timeonpage' topic")

* **Kafka Consumer** (kafka\_consumer.py):

Listens to the Kafka topics. Assembles related session metrics using a common record\_id. Inserts complete session records into a MySQL table (raw\_traffic\_data).

from kafka import KafkaConsumer

import mysql.connector

import json

import time

from datetime import datetime

# Initialize buffer to hold partial data

data\_buffer = {}

# MySQL DB connection

db = mysql.connector.connect(

host="localhost",

user="root",

password="2004",

database="website\_traffic"

)

cursor = db.cursor()

# Kafka consumer setup

consumer = KafkaConsumer(

'topic\_pageviews',

'topic\_sessionduration',

'topic\_timeonpage',

bootstrap\_servers='localhost:9092',

auto\_offset\_reset='earliest',

value\_deserializer=lambda m: json.loads(m.decode('utf-8')),

group\_id='web-traffic-group-test' # <- new group name

)

try:

print("Starting Kafka consumer. Press Ctrl+C to exit.")

# Consume messages

for message in consumer:

topic = message.topic

value = message.value

record\_id = value['record\_id']

print(f"Received message on topic '{topic}': {value}")

# Add to buffer

if record\_id not in data\_buffer:

data\_buffer[record\_id] = {}

if topic == 'topic\_pageviews':

data\_buffer[record\_id]['page\_views'] = value['page\_views']

elif topic == 'topic\_sessionduration':

data\_buffer[record\_id]['session\_duration'] = value['session\_duration']

elif topic == 'topic\_timeonpage':

data\_buffer[record\_id]['time\_on\_page'] = value['time\_on\_page']

data\_buffer[record\_id]['timestamp'] = value['timestamp']

# If all 3 fields are present, insert into DB

if all(k in data\_buffer[record\_id] for k in ['page\_views', 'session\_duration', 'time\_on\_page']):

try:

row = data\_buffer.pop(record\_id)

insert\_query = """

INSERT INTO raw\_traffic\_data (page\_views, session\_duration, time\_on\_page, timestamp)

VALUES (%s, %s, %s, %s)

"""

cursor.execute(insert\_query, (

row['page\_views'],

row['session\_duration'],

row['time\_on\_page'],

row['timestamp']

))

db.commit()

print(f" Inserted record\_id {record\_id} into raw\_traffic\_data")

except Exception as e:

print(f" Failed to insert record\_id {record\_id}: {e}")

except KeyboardInterrupt:

print("Consumer stopped by user")

finally:

consumer.close()

* **Spark Streaming Processor** (spark\_streaming\_new.py):

Reads live data from Kafka in real time. Performs joins and transformations to combine session metrics. Stores these results in MySQL for later batch analysis.

Calculates key analytics in 5-minute tumbling windows:

* + **Page View Distribution**
  + **Session Categories** (Short/Medium/Long)
  + **Engagement Score Statistic**



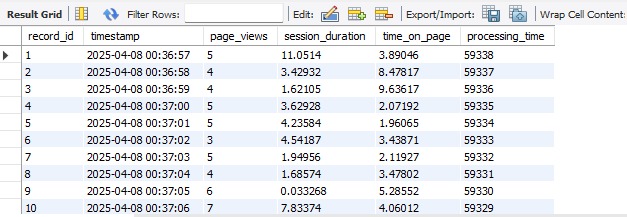
1. Windows

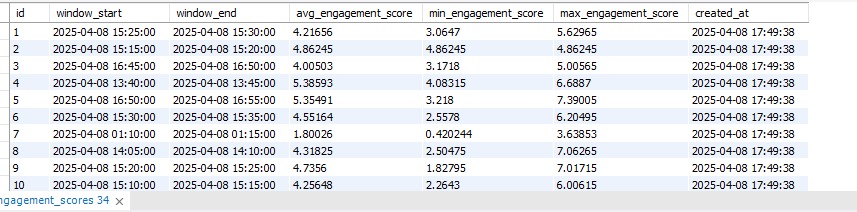
5-minute tumbling windows are used to group real-time streaming data into fixed time intervals for meaningful analysis. Tumbling windows divide the data stream into non-overlapping blocks of 5 minutes each. All user sessions that occur within the same 5-minute period are processed together. This windowing strategy is applied in the Spark Streaming pipeline to compute:

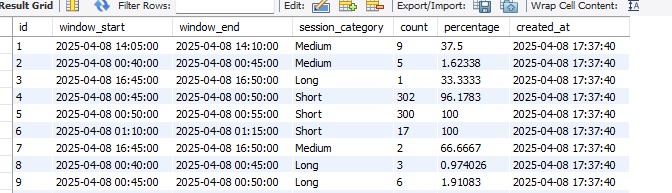
* Page Views Distribution
* Session Categories (Short, Medium, Long)
* Engagement Score Statistics (average, min, max)

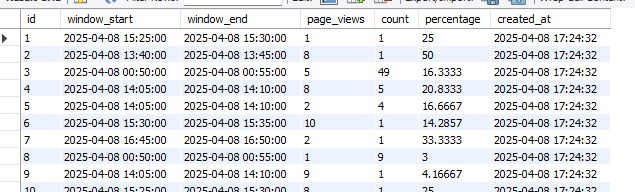
By using fixed-size windows, the system provides consistent and periodic summaries of user activity, making it easier to observe trends and detect anomalies in real time.

1. Results









Batch Mode Experiment

1. Description

Batch processing replicates the same analyses that were done in streaming (Spark) but now using SQL on historical data stored in MySQL.

In the batch processing phase, the system analyses the website session data stored in the MySQL database to uncover historical patterns. Using SQL queries, the project replicates the same analyses performed during stream processing, but in an offline mode. The data is grouped into fixed 5-minute windows using SQL timestamp functions to simulate tumbling windows.

The batch analysis includes:

* Page Views Distribution: Calculates how often different page view counts occur in each 5-minute interval, along with their percentages.
* Session Categories: Classifies sessions as Short, Medium, or Long based on session duration and computes their proportions within each time window.
* Engagement Score Statistics: Computes engagement scores using a weighted formula and provides average, minimum, and maximum scores for each 5-minute window.

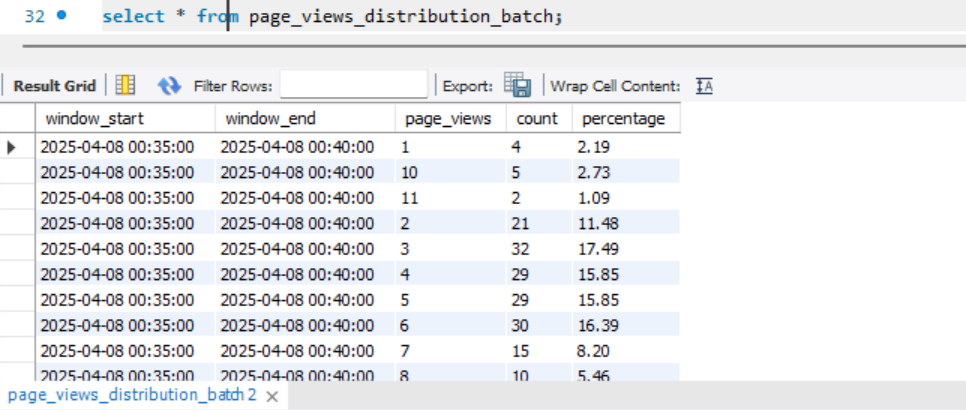
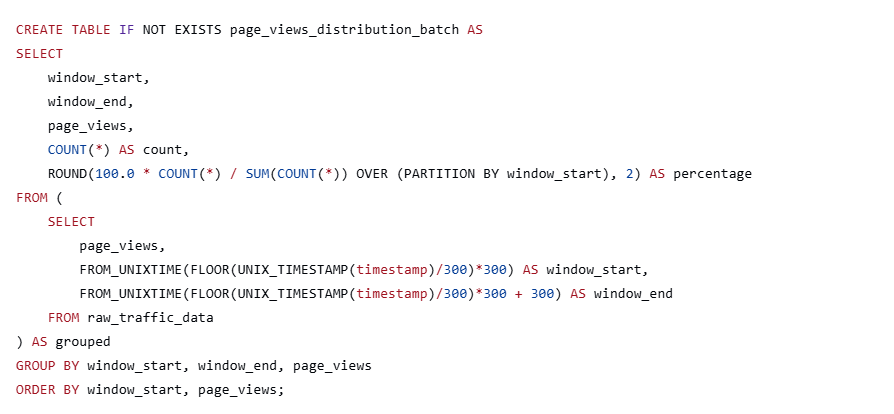
This phase enables in-depth retrospective analysis of user behavior and supports trend detection over time. It also analyses quarterly traffic trends, bounce and conversion rates.

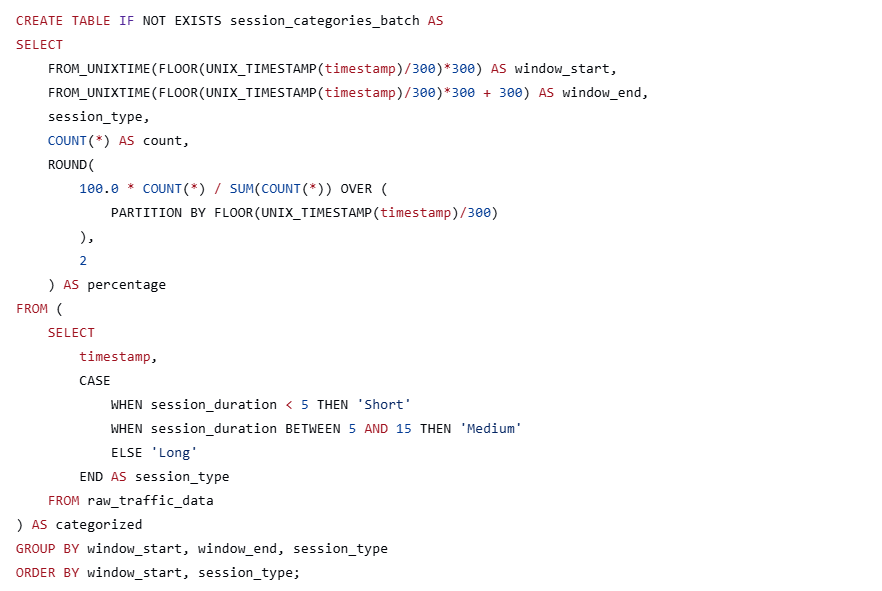
1. Data Size

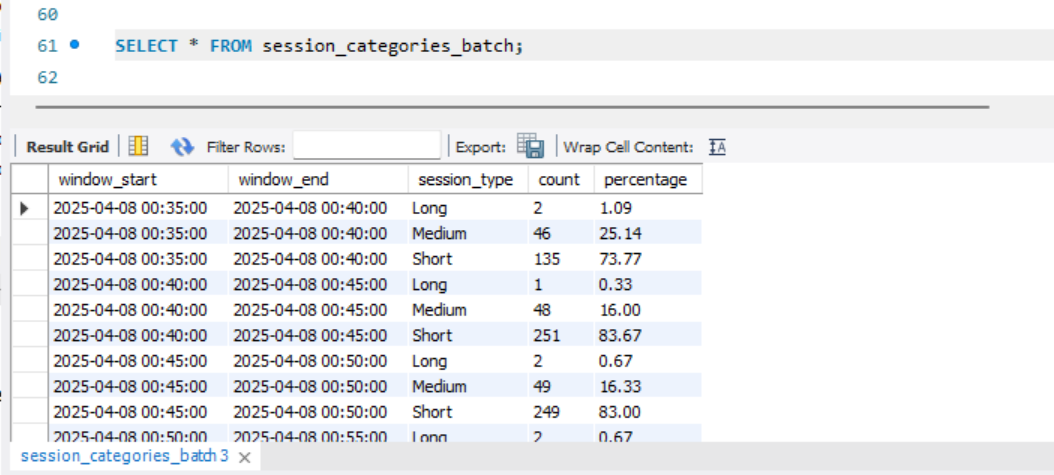
The dataset contains:

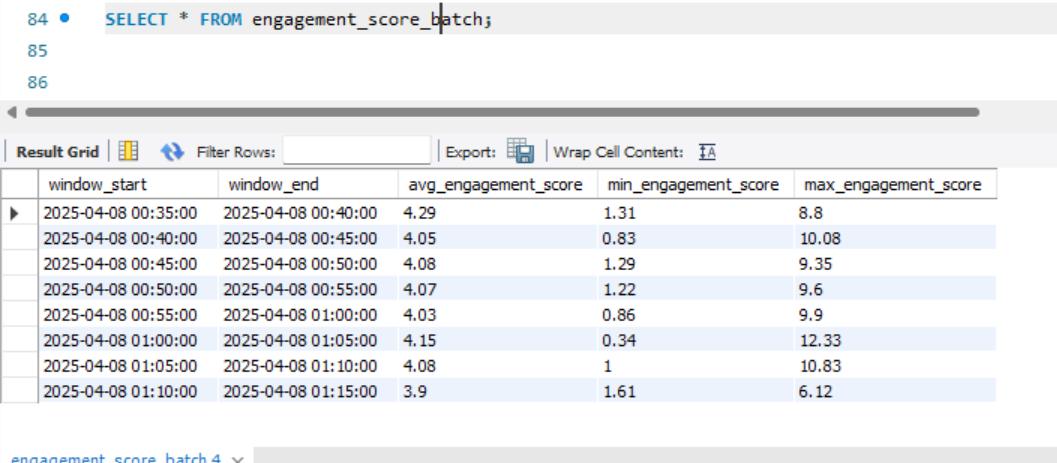
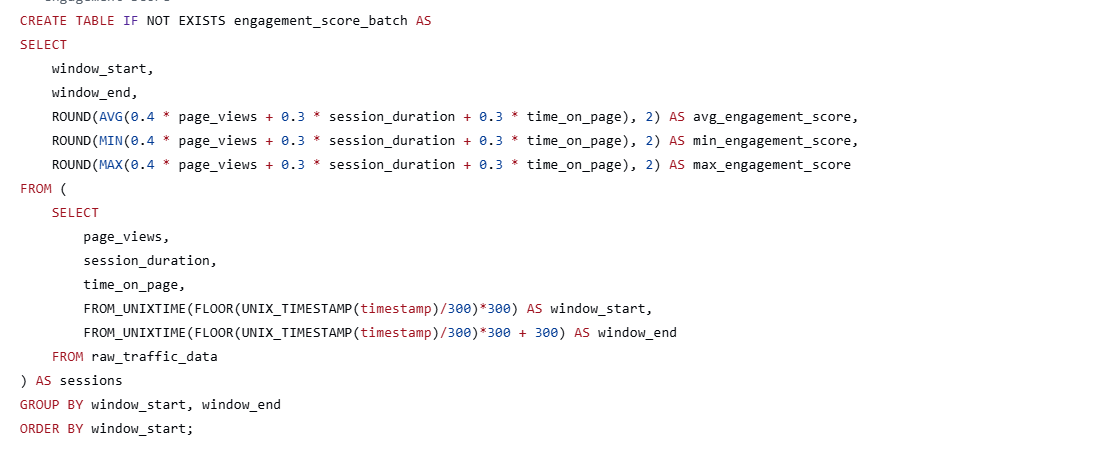
* 2,000 rows and 4 columns
* Memory size: approximately 0.19 MB

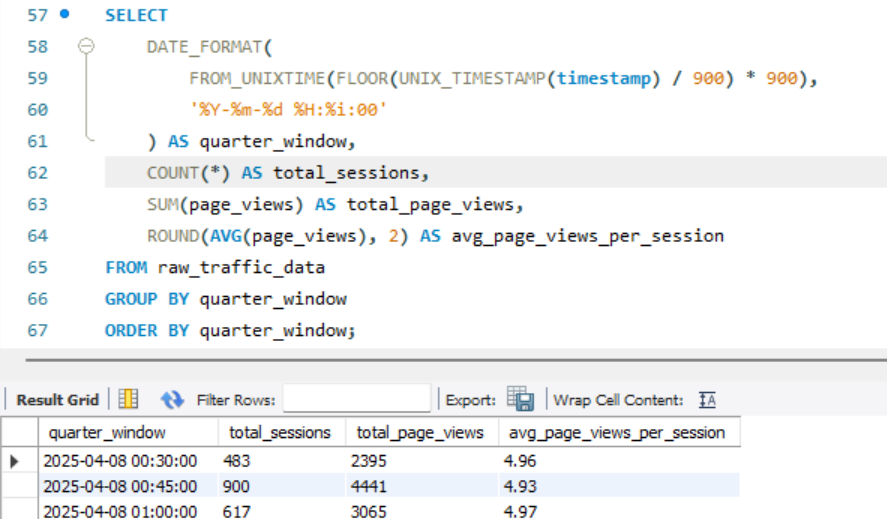
1. Results

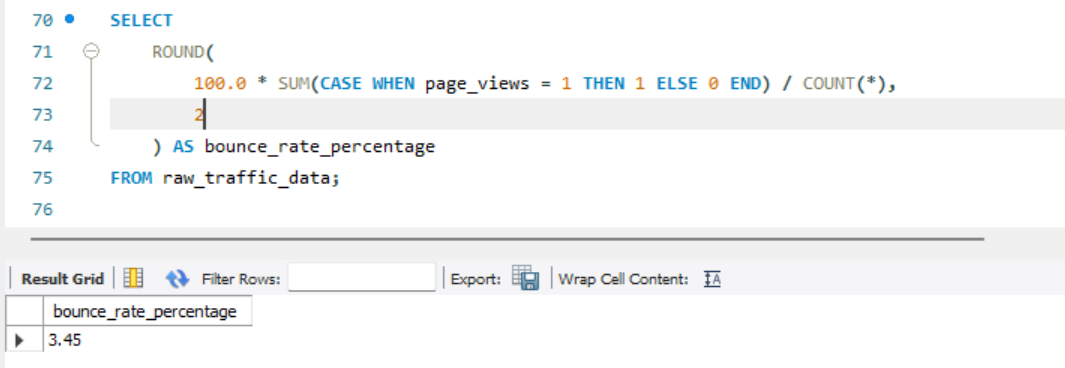


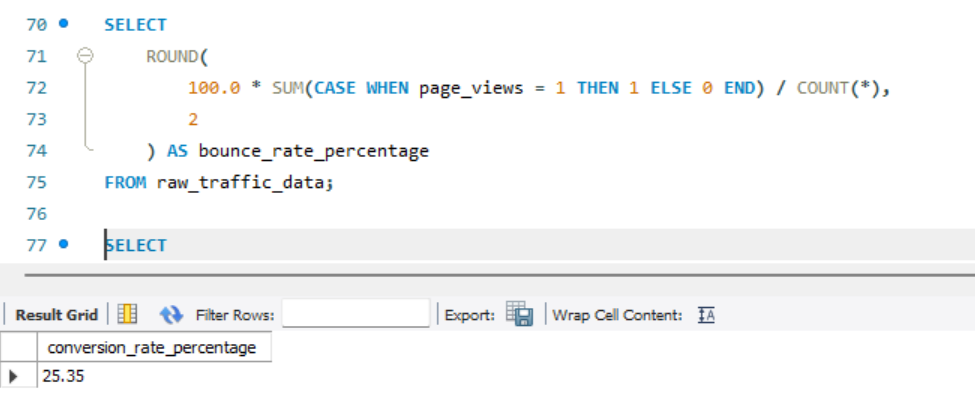










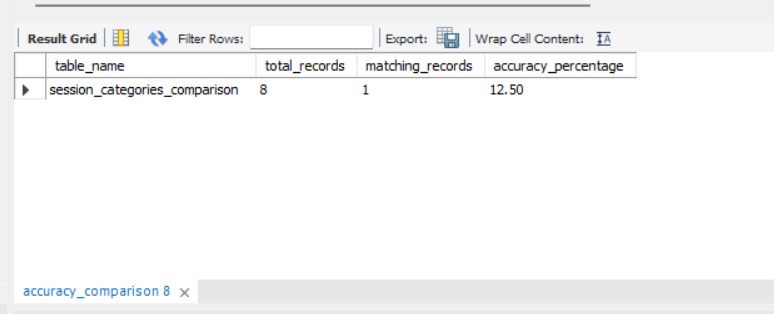
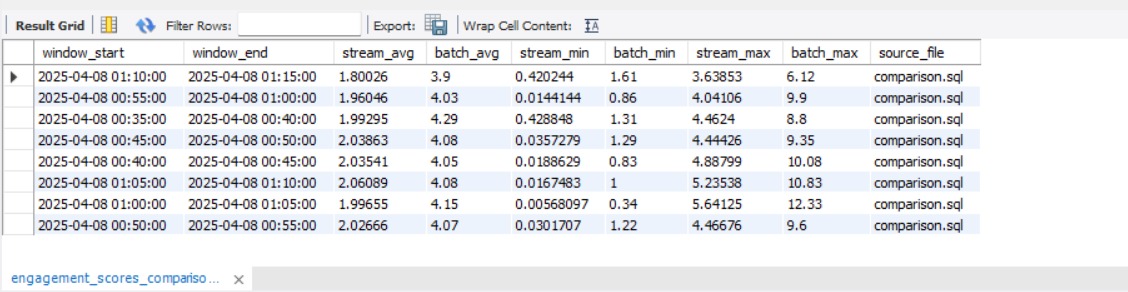
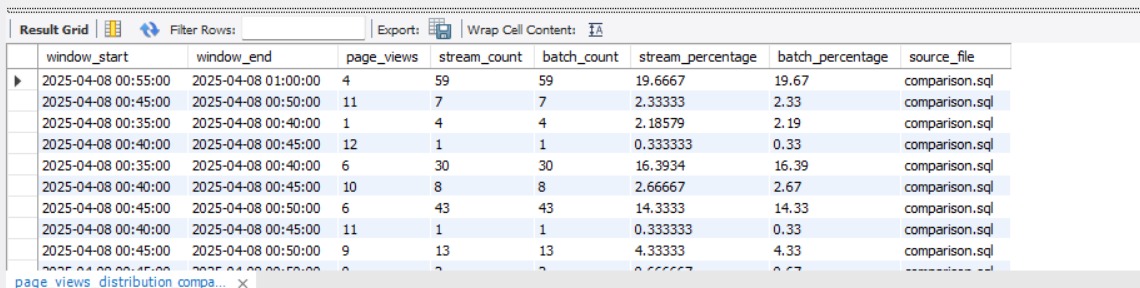
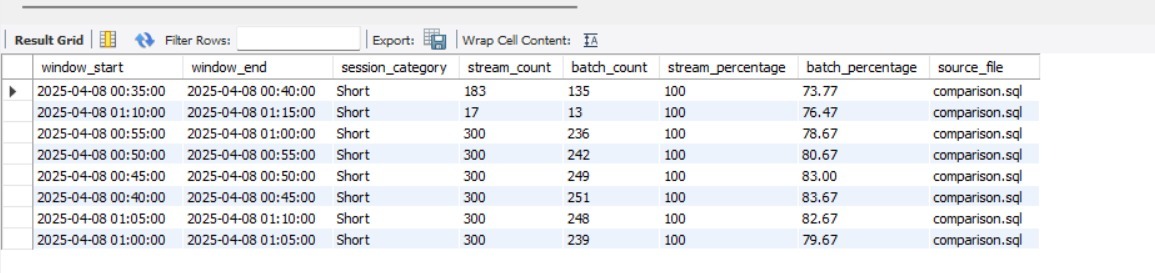


Comparison of Streaming & Batch Modes

* 1. Results







* 1. Discussion
* In this project, streaming mode processes data in real-time using Kafka and Spark, enabling instant insights like live page view trends, session categories, and engagement scores. In contrast, batch mode analyses stored session data in MySQL using SQL queries to extract historical trends, such as bounce rates and conversion patterns.
* Streaming offers low-latency, immediate feedback, while batch processing provides deeper, retrospective analysis. Together, they enable both real-time monitoring and long-term decision-making.

Conclusion

This project demonstrates an end-to-end system for real-time and batch-based website traffic analysis. By integrating Kafka, Spark Streaming, and MySQL, it effectively captures, processes, and analyses user interactions in both live and offline modes. The system provides valuable insights such as session trends, engagement levels, and user behavior patterns, proving the effectiveness of combining streaming and batch processing for data-driven decision-making.