Table of Contents

# Video Analytics Databricks - Quick Reference Guide

## 🎯 Overview

**Problem:** Raw Events (play, pause, resume, end) → Aggregated User-Video metrics

**Solution:** PySpark script that transforms atomic events into a clean table with **one row per User+Video**

## 📊 Output Schema

The final aggregated\_user\_video\_engagement table contains:

userId STRING -- User Identifier  
videoId STRING -- Video Identifier   
videoTitle STRING -- Video Title (from metadata)  
videoDuration DOUBLE -- Video length in seconds  
  
-- Watch Time Metrics  
totalWatchTime DOUBLE -- Total watched time in seconds  
totalUniqueSecondsWatched DOUBLE -- Unique seconds (without counting replays twice)  
maxPositionReached DOUBLE -- Furthest position reached in video  
  
-- Percentages  
watchPercentage DOUBLE -- (totalWatchTime / videoDuration) \* 100  
completionPercentage DOUBLE -- (maxPositionReached / videoDuration) \* 100  
uniqueWatchPercentage DOUBLE -- (uniqueSecondsWatched / videoDuration) \* 100  
  
-- Session Metrics  
sessionCount LONG -- Number of sessions  
avgWatchTimePerSession DOUBLE -- Average watch time per session  
avgSessionDuration DOUBLE -- Average session duration (incl. pauses)  
  
-- Interaction Metrics   
avgPausesPerSession DOUBLE -- Average number of pauses  
totalForwardSkips LONG -- Number of forward skips  
totalBackwardSkips LONG -- Number of backward skips  
  
-- Completion  
completionCount LONG -- How many times video was completed  
isCompletedAtLeastOnce BOOLEAN -- Completed at least once  
  
-- Engagement  
engagementScore DOUBLE -- Weighted engagement metric  
engagementTier STRING -- High / Medium / Low / Minimal  
  
-- Temporal  
firstWatchDate TIMESTAMP -- First watch session  
lastWatchDate TIMESTAMP -- Last watch session  
  
-- Flags  
isReplay BOOLEAN -- Multiple sessions (replay behavior)  
dataQualityFlag STRING -- ok / excessive\_watch\_time / very\_short\_watch / etc.  
  
-- Meta  
processedAt TIMESTAMP -- When aggregated

## 🚀 Quick Start

### 1. Upload Script to Databricks

# Upload databricks\_video\_aggregation.py to Databricks Workspace  
# Path: /Workspace/Users/<your-email>/video\_analytics/

### 2. Run in Notebook

# In Databricks Notebook  
%run /Workspace/Users/your-email/video\_analytics/databricks\_video\_aggregation  
  
from databricks\_video\_aggregation import VideoEngagementAggregator  
  
# Initialize  
aggregator = VideoEngagementAggregator(  
 spark=spark,  
 input\_table="your\_raw\_events\_table",  
 output\_table="aggregated\_user\_video\_engagement",  
 video\_metadata\_table="video\_metadata" # Optional  
)  
  
# Run  
result = aggregator.run\_aggregation()  
  
# Save  
aggregator.save\_results(result)

### 3. Query Results

# Read aggregated table  
df = spark.table("aggregated\_user\_video\_engagement")  
  
# Peter's engagement for Video 1  
df.filter((col("userId") == "peter") & (col("videoId") == "video\_001")).show()

## 📝 Example: Peter’s Video Journey

**Raw Events:**

timestamp | eventName | currentTime  
-----------------------|--------------|------------  
2025-01-15 10:00:00 | video\_play | 0  
2025-01-15 10:00:30 | video\_pause | 30 ← Watched 30s  
2025-01-15 10:00:35 | video\_resume | 30  
2025-01-15 10:02:05 | video\_pause | 120 ← Watched 90s  
2025-01-15 10:02:10 | video\_resume | 110 ← Skip back 10s  
2025-01-15 10:02:20 | video\_pause | 120 ← Watched 10s

**Aggregated Result (1 Row):**

userId: peter  
videoId: video\_001  
videoDuration: 300s (5 minutes)  
  
totalWatchTime: 130s (30 + 90 + 10)  
uniqueSecondsWatched: 120s (0-120, without counting 110-120 twice)  
maxPositionReached: 120s  
  
watchPercentage: 43.3% (130/300)  
completionPercentage: 40% (120/300)  
uniqueWatchPercentage: 40% (120/300)  
  
sessionCount: 1  
completionCount: 0  
isCompletedAtLeastOnce: False  
  
forwardSkips: 0  
backwardSkips: 1  
  
engagementScore: ~17.2  
engagementTier: Low

## 🔧 Configuration

### Input Table Requirements

# Your raw events table must have the following columns:  
# - timestamp (TimestampType)  
# - userId (StringType)  
# - sessionId (StringType)  
# - videoId (StringType)  
# - eventName (StringType): "video\_play", "video\_pause", "video\_resume", "video\_ended"  
# - currentTime (DoubleType): Position in video in seconds

### Optional: Video Metadata Table

# If available, create table with:  
# - videoId (StringType)  
# - duration (DoubleType)  
# - title (StringType)  
  
# Or: Script estimates duration from maxPosition

### Performance Tuning

# For large datasets (>10M events):  
aggregator.run\_aggregation(  
 calculate\_unique\_seconds=True, # True = accurate, False = faster  
 use\_efficient\_method=True # True for production (interval merging)  
)  
  
# Incremental processing:  
from datetime import datetime, timedelta  
start\_date = datetime.now() - timedelta(days=1)  
result = aggregator.run\_aggregation(start\_date=start\_date)

## 📅 Scheduling (Production Setup)

### Option A: Databricks Job

{  
 "name": "Video Analytics Daily Aggregation",  
 "schedule": {  
 "quartz\_cron\_expression": "0 0 2 \* \* ?",  
 "timezone\_id": "Europe/Zurich"  
 },  
 "tasks": [{  
 "task\_key": "aggregate\_video\_engagement",  
 "notebook\_task": {  
 "notebook\_path": "/video\_analytics/aggregation\_notebook",  
 "base\_parameters": {  
 "start\_date": "{{job.start\_time.iso\_datetime}}"  
 }  
 }  
 }]  
}

### Option B: Workflow Notebook

# Notebook: daily\_video\_aggregation.py  
  
from datetime import datetime, timedelta  
  
# Process yesterday's data  
yesterday = datetime.now() - timedelta(days=1)  
start\_date = yesterday.replace(hour=0, minute=0, second=0)  
end\_date = start\_date + timedelta(days=1)  
  
# Run aggregation  
result = aggregator.run\_aggregation(  
 start\_date=start\_date,  
 end\_date=end\_date  
)  
  
# Save with append mode for incremental updates  
aggregator.save\_results(result, mode="append")

## 🎨 BI Integration

### Power BI / Tableau Connection

-- Create optimized view  
CREATE OR REPLACE VIEW vw\_video\_engagement\_bi AS  
SELECT   
 userId,  
 videoId,  
 videoTitle,  
 totalWatchTime / 60.0 as watchMinutes,  
 watchPercentage,  
 completionPercentage,  
 engagementTier,  
 DATE(firstWatchDate) as firstWatchDay,  
 sessionCount,  
 isCompletedAtLeastOnce  
FROM aggregated\_user\_video\_engagement  
WHERE dataQualityFlag = 'ok';

### Example Dashboard Queries

**Top Videos by Engagement:**

SELECT   
 videoId,  
 videoTitle,  
 COUNT(DISTINCT userId) as uniqueViewers,  
 SUM(totalWatchTime) / 3600 as totalWatchHours,  
 AVG(watchPercentage) as avgWatchPercentage,  
 SUM(completionCount) as totalCompletions  
FROM aggregated\_user\_video\_engagement  
GROUP BY videoId, videoTitle  
ORDER BY totalWatchHours DESC  
LIMIT 10;

**User Engagement Distribution:**

SELECT   
 engagementTier,  
 COUNT(\*) as userVideoCount,  
 COUNT(\*) \* 100.0 / SUM(COUNT(\*)) OVER () as percentage  
FROM aggregated\_user\_video\_engagement  
GROUP BY engagementTier  
ORDER BY   
 CASE engagementTier  
 WHEN 'High' THEN 1  
 WHEN 'Medium' THEN 2  
 WHEN 'Low' THEN 3  
 ELSE 4  
 END;

**Drop-off Analysis:**

-- Where do users drop off?  
SELECT   
 videoId,  
 FLOOR(maxPositionReached / 30) \* 30 as positionBucket,  
 COUNT(\*) as userCount  
FROM aggregated\_user\_video\_engagement  
WHERE completionCount = 0  
GROUP BY videoId, positionBucket  
ORDER BY videoId, positionBucket;

## ✅ Data Quality Checks

### Automated Validation Queries

**1. Watch Time Cannot Exceed Duration:**

quality\_issues = spark.sql("""  
 SELECT   
 userId, videoId,  
 totalWatchTime, videoDuration,  
 (totalWatchTime - videoDuration) as excess  
 FROM aggregated\_user\_video\_engagement  
 WHERE totalWatchTime > videoDuration \* 1.1  
 ORDER BY excess DESC  
""")  
  
if quality\_issues.count() > 0:  
 print("⚠️ Found excessive watch times!")  
 quality\_issues.show()

**2. Completion Without Sufficient Watch:**

incomplete\_completions = spark.sql("""  
 SELECT userId, videoId, completionCount, watchPercentage  
 FROM aggregated\_user\_video\_engagement  
 WHERE completionCount > 0 AND watchPercentage < 75  
""")

**3. Data Freshness:**

freshness = spark.sql("""  
 SELECT   
 MAX(processedAt) as lastProcessed,  
 TIMESTAMPDIFF(HOUR, MAX(processedAt), CURRENT\_TIMESTAMP()) as hoursAgo  
 FROM aggregated\_user\_video\_engagement  
""").collect()[0]  
  
if freshness['hoursAgo'] > 24:  
 print("⚠️ Data is stale! Last processed over 24h ago")

## 🐛 Troubleshooting

### Problem: “Column not found”

# Check input table schema  
spark.table("raw\_video\_events").printSchema()  
  
# Ensure all required columns exist:  
# timestamp, userId, sessionId, videoId, eventName, currentTime

### Problem: “Out of Memory”

# For very large datasets:  
# 1. Disable unique seconds calculation  
result = aggregator.run\_aggregation(calculate\_unique\_seconds=False)  
  
# 2. Process in batches  
for month in range(1, 13):  
 start = f"2024-{month:02d}-01"  
 end = f"2024-{month:02d}-28"  
 result = aggregator.run\_aggregation(start\_date=start, end\_date=end)  
 aggregator.save\_results(result, mode="append")

### Problem: Negative Watch Time

# Debug: Find problematic sessions  
spark.sql("""  
 SELECT userId, videoId, sessionId,   
 COLLECT\_LIST(STRUCT(timestamp, eventName, currentTime)) as events  
 FROM raw\_video\_events  
 GROUP BY userId, videoId, sessionId  
 HAVING SUM(CASE WHEN eventName = 'video\_pause'   
 AND currentTime < LAG(currentTime)   
 THEN 1 ELSE 0 END) > 0  
""").show(truncate=False)

## 📈 Performance Benchmarks

**Typical Performance (Databricks Standard Cluster):** - 1M events → ~2-3 minutes - 10M events → ~15-20 minutes - 100M events → ~2-3 hours

**Optimization Tips:** 1. **Partitioning:** Partition output table by date python aggregator.save\_results(result, partition\_by=["firstWatchDate"])

1. **Caching:** Script already caches intermediate results
2. **Cluster Size:** Use autoscaling cluster

* Min workers: 2  
  Max workers: 8

1. **Delta Lake:** Output as Delta table for ACID + Time Travel

## 🔄 Migration from KQL to PySpark

**KQL Equivalents in PySpark:**

| **KQL** | **PySpark** |
| --- | --- |
| serialize | Window.orderBy() |
| prev() | lag() over window |
| next() | lead() over window |
| summarize | groupBy().agg() |
| extend | withColumn() |
| where | filter() |
| mv-expand | explode() |

## 📚 Additional Resources

* **Main Script:** databricks\_video\_aggregation.py
* **Example Notebook:** databricks\_example\_notebook.py
* **KQL Queries:** video\_analytics\_kql.md (for Azure Log Analytics)
* **Implementation Guide:** implementation\_roadmap.md

## 🎯 Next Steps

1. ✅ Upload script to Databricks
2. ✅ Run example notebook with sample data
3. ✅ Validate output for known users
4. ✅ Point script to your real raw events
5. ✅ Schedule as daily job
6. ✅ Connect BI tool to output table
7. ✅ Setup data quality alerts
8. ✅ Iterate based on business feedback

**Happy Analyzing! 🚀**