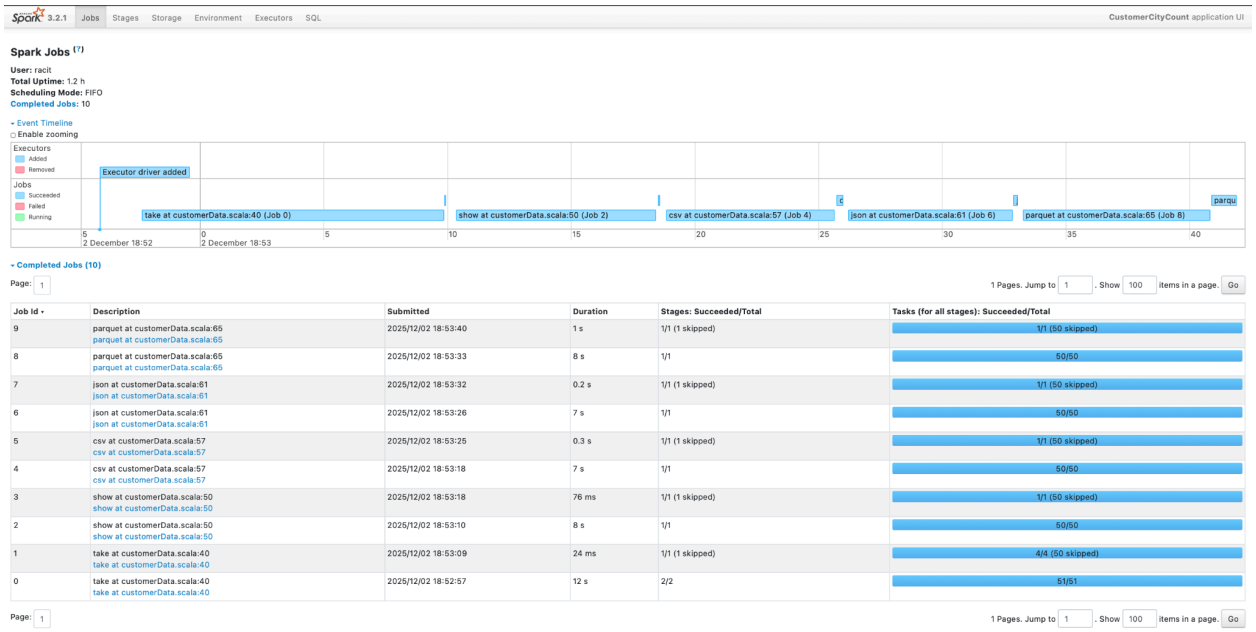
Spark Day 2 Assignment

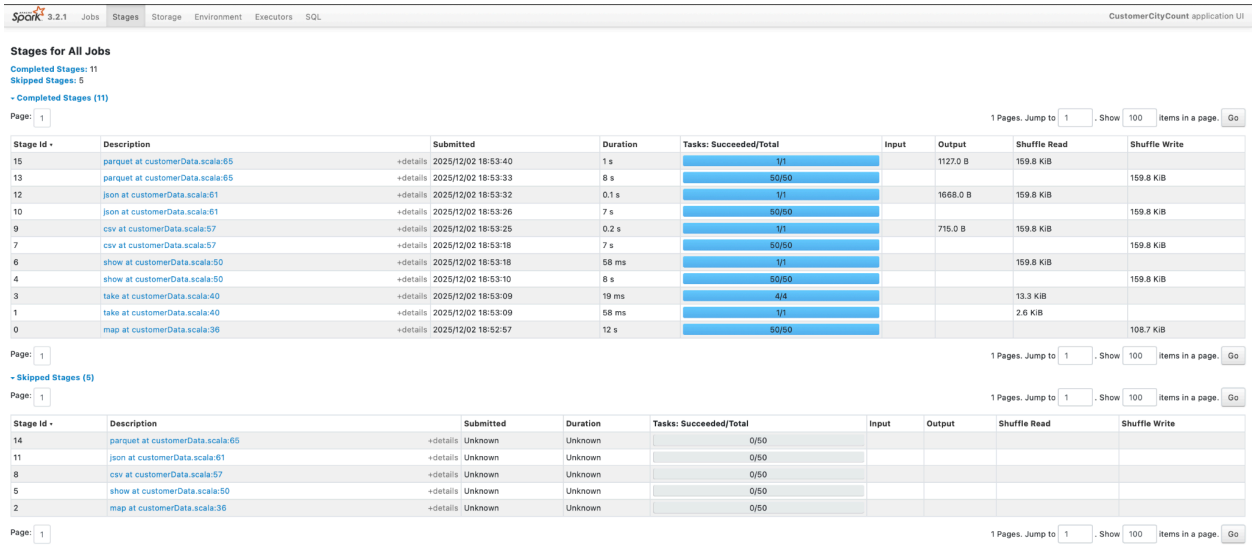
---

1. Customer Data

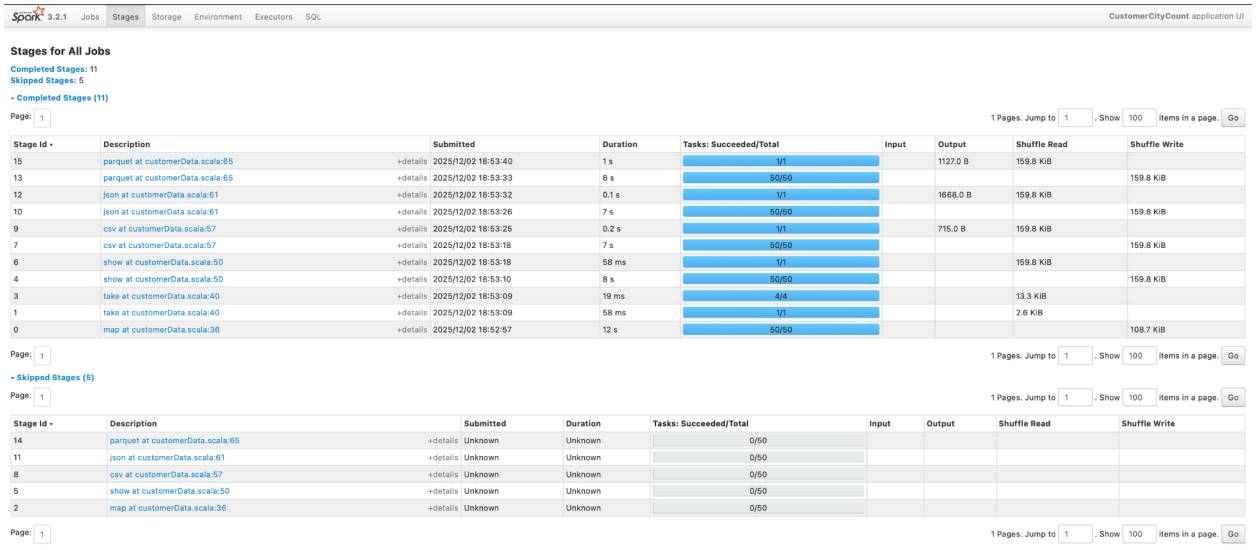
Jobs



Stages



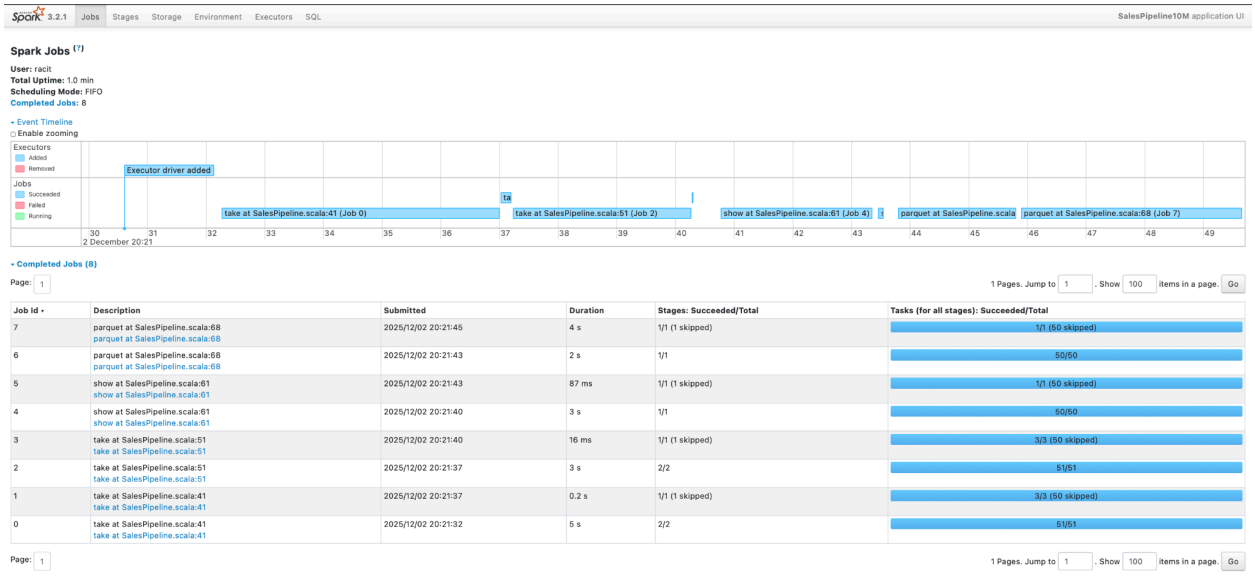
Executors



Observation	CSV	JSON	Parquet
Write Speed	**Fastest** (very low overhead)	Medium	**Slowest** (metadata + encoding overhead)
Output Size (in this pipeline)	**Smallest** (raw text, no metadata)	Largest (verbose text, repeats field names)	Medium (metadata overhead bigger than actual data since only ~50 rows)
Operations Used	-	-	-
Narrow Ops	map (customer generation), toDF, final part of take/show		
Broad Ops	reduceByKey, groupBy("city").count()(shuffle)		

2. Sales Data

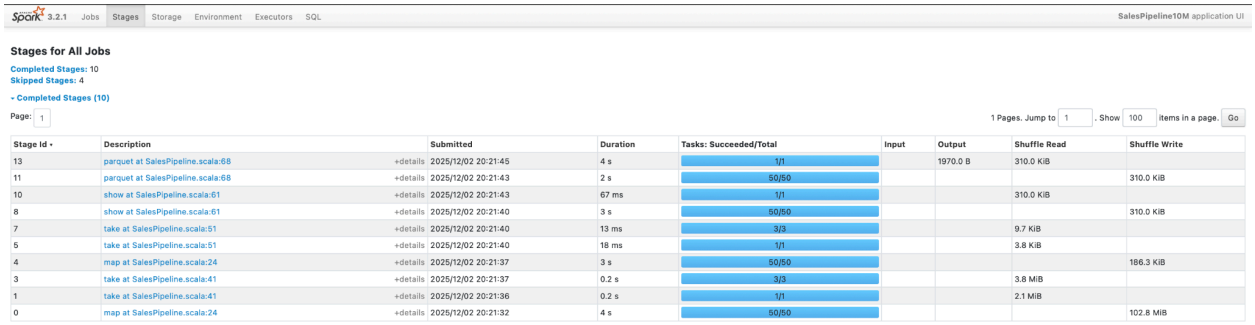
Jobs



Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
7	parquet at SalesPipeline.scala:68 parquet at SalesPipeline.scala:68	2025/12/02 20:21:45	4 s	1/1 (1 skipped)	1/1 (50 skipped)
6	parquet at SalesPipeline.scala:68 parquet at SalesPipeline.scala:68	2025/12/02 20:21:43	2 s	1/1	50/50
5	show at SalesPipeline.scala:61 show at SalesPipeline.scala:61	2025/12/02 20:21:43	87 ms	1/1 (1 skipped)	1/1 (50 skipped)
4	show at SalesPipeline.scala:61 show at SalesPipeline.scala:61	2025/12/02 20:21:40	3 s	1/1	50/50
3	take at SalesPipeline.scala:51 take at SalesPipeline.scala:51	2025/12/02 20:21:40	16 ms	1/1 (1 skipped)	3/3 (50 skipped)
2	take at SalesPipeline.scala:51 take at SalesPipeline.scala:51	2025/12/02 20:21:37	3 s	2/2	51/51
1	take at SalesPipeline.scala:41 take at SalesPipeline.scala:41	2025/12/02 20:21:37	0.2 s	1/1 (1 skipped)	3/3 (50 skipped)
0	take at SalesPipeline.scala:41 take at SalesPipeline.scala:41	2025/12/02 20:21:32	5 s	2/2	51/51

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

Stages



Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
13	parquet at SalesPipeline.scala:68 +details	2025/12/02 20:21:45	4 s	1/1		1970.0 B	310.0 KiB	
11	parquet at SalesPipeline.scala:68 +details	2025/12/02 20:21:43	2 s	50/50				310.0 KiB
10	show at SalesPipeline.scala:61 +details	2025/12/02 20:21:43	67 ms	1/1			310.0 KiB	
8	show at SalesPipeline.scala:61 +details	2025/12/02 20:21:40	3 s	50/50				310.0 KiB
7	take at SalesPipeline.scala:51 +details	2025/12/02 20:21:40	13 ms	3/3			9.7 KiB	
5	take at SalesPipeline.scala:51 +details	2025/12/02 20:21:40	18 ms	1/1			3.8 KiB	
4	map at SalesPipeline.scala:24 +details	2025/12/02 20:21:37	3 s	50/50				186.3 KiB
3	take at SalesPipeline.scala:41 +details	2025/12/02 20:21:37	0.2 s	3/3			3.8 MiB	
1	take at SalesPipeline.scala:41 +details	2025/12/02 20:21:36	0.2 s	1/1			2.1 MiB	
0	map at SalesPipeline.scala:24 +details	2025/12/02 20:21:32	4 s	50/50				102.8 MiB

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

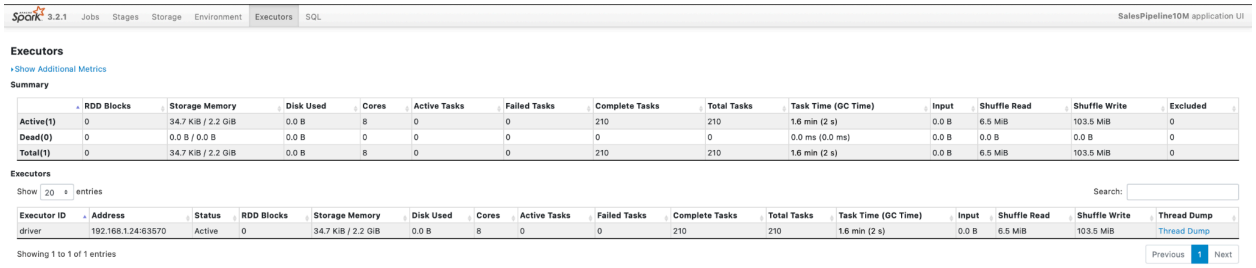
Skipped Stages (4)

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
12	parquet at SalesPipeline.scala:68 +details	Unknown	Unknown	0/50				
9	show at SalesPipeline.scala:61 +details	Unknown	Unknown	0/50				
6	map at SalesPipeline.scala:24 +details	Unknown	Unknown	0/50				
2	map at SalesPipeline.scala:24 +details	Unknown	Unknown	0/50				

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

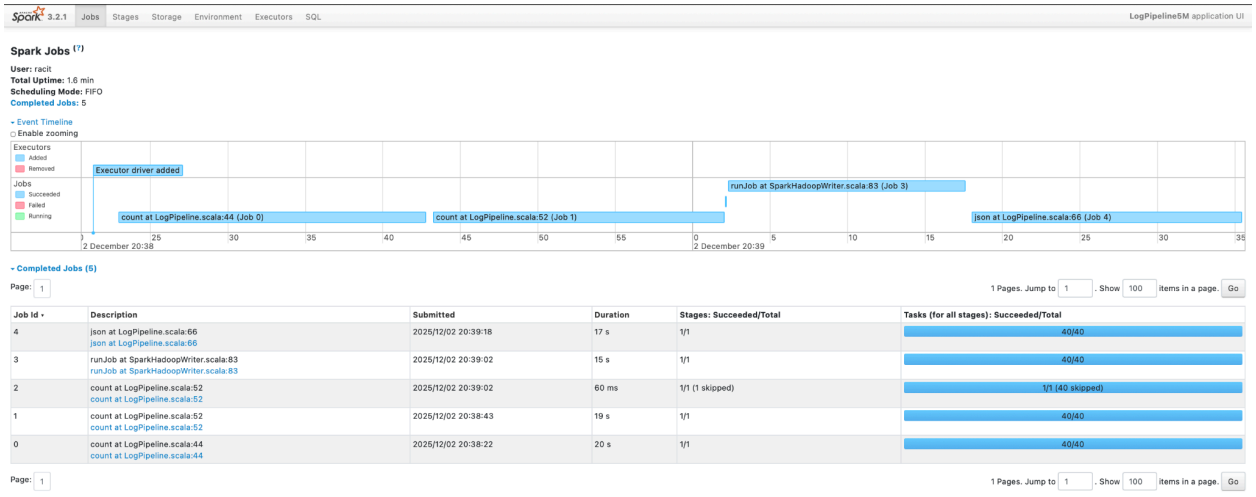
Executors



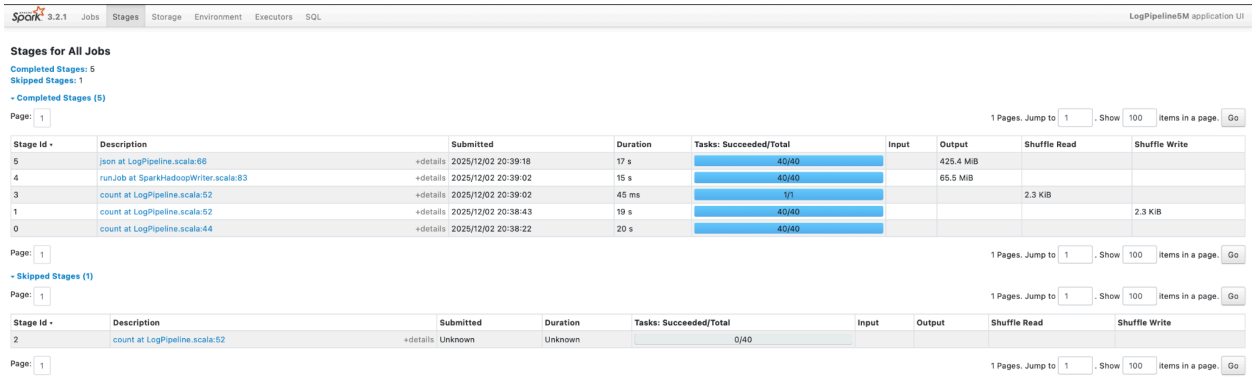
Observation	Summary
Slower Operation	**groupByKey is slower** — shuffles all 10M values, very heavy.
Shuffle Operations	groupByKey, reduceByKey, and DF groupBy all trigger **shuffles**.
Why reduceByKey is Narrow → Broad	First does a narrow **map-side combine**, then a **broad shuffle** → more efficient than groupByKey.

3.Log files

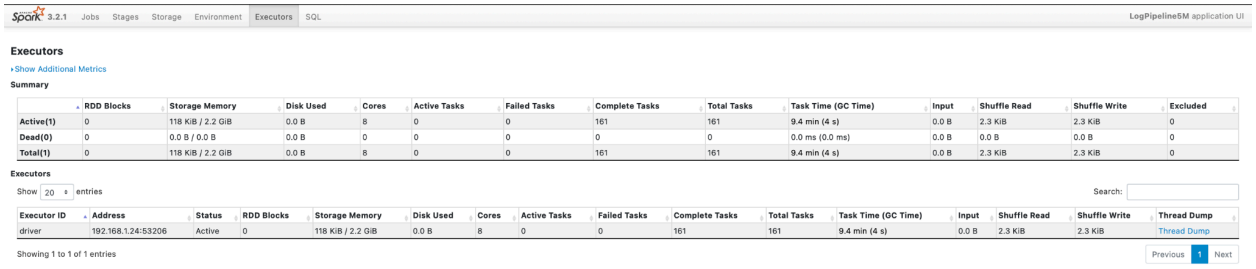
Jobs



Stages



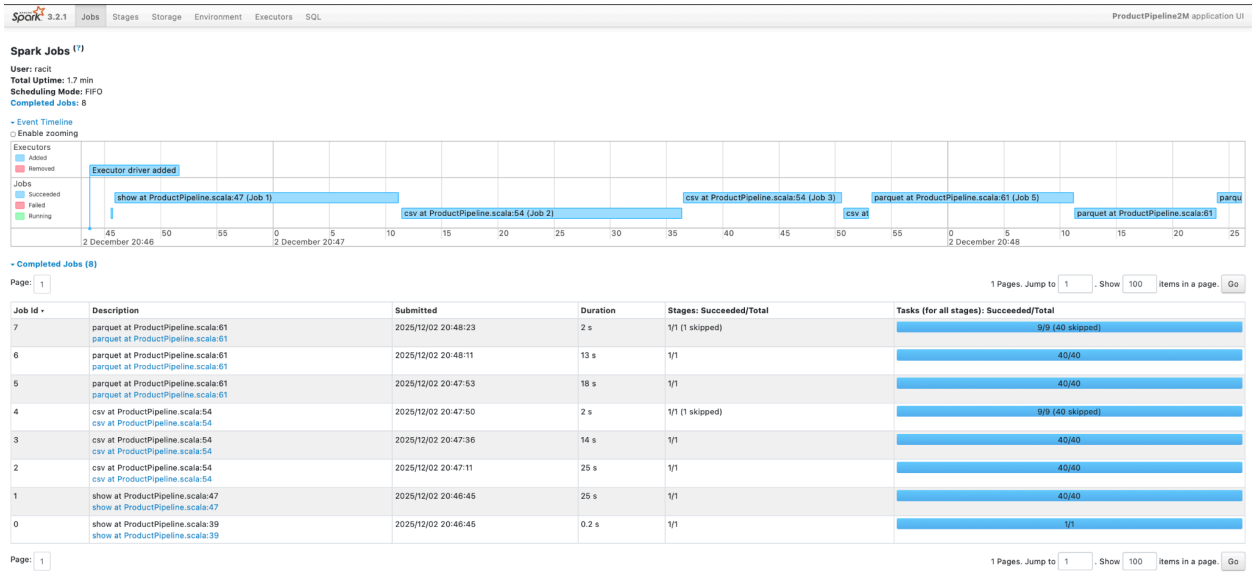
Executors



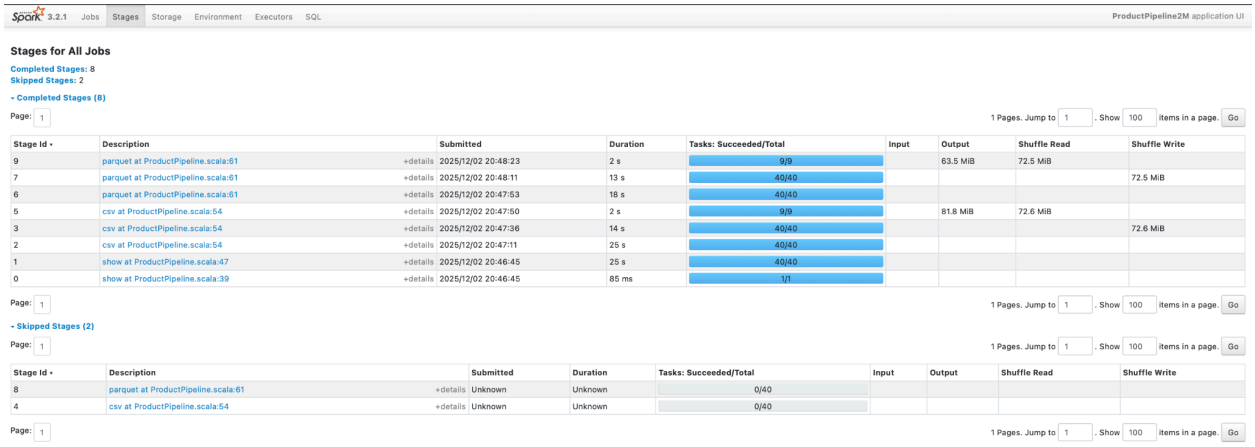
Observation	Summary
Why is plain text slow?	Writes large raw strings → high I/O → many small files.
filter	**Narrow** (no shuffle).
map	**Narrow** (no shuffle).
sort	**Broad** (shuffle required).

4. Product Catalog

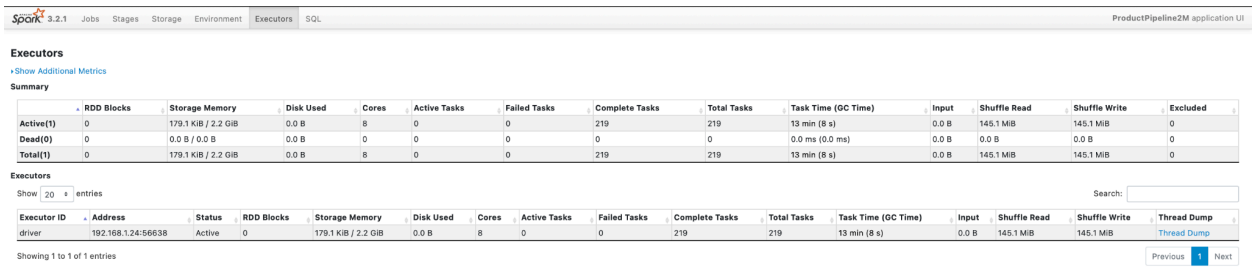
Jobs



Stages



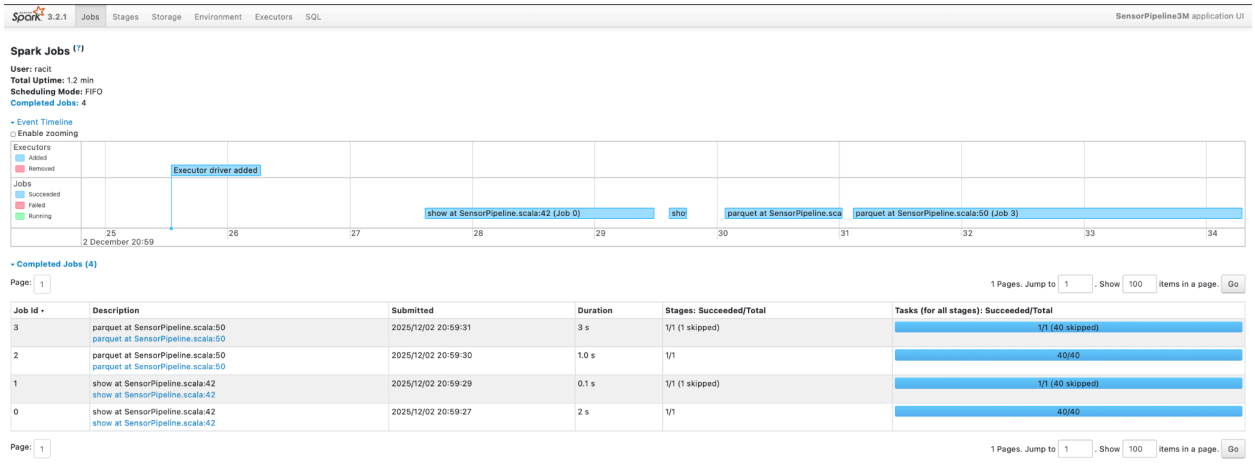
Executors



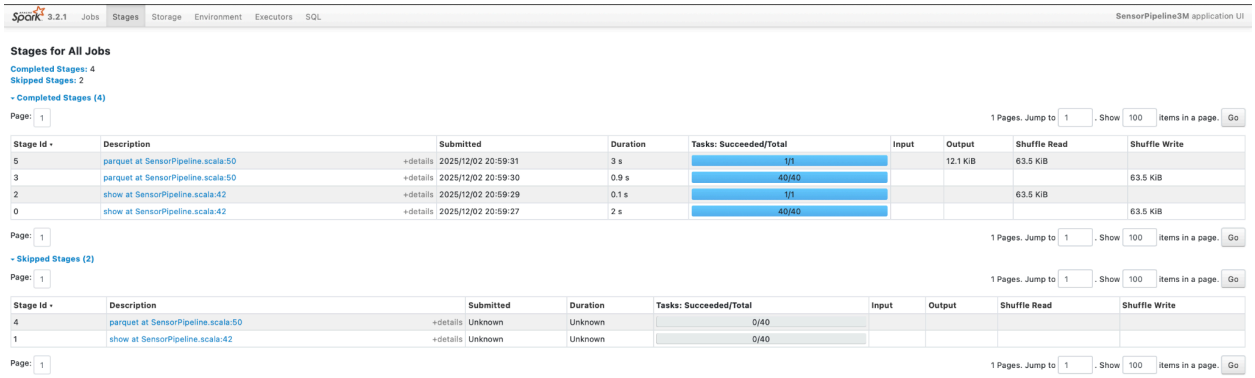
Observation	Summary
CSV vs Parquet Write Speed	**CSV slower** → writes huge plain text. **Parquet faster** → compressed columnar write.
CSV vs Parquet Output Size	**CSV = largest** (raw text). **Parquet = smallest** (compression + encoding).
Why sort causes shuffle?	Sorting needs **global ordering**, so Spark must **shuffle all partitions** → broad transformation.

5. Iot Sensor Data

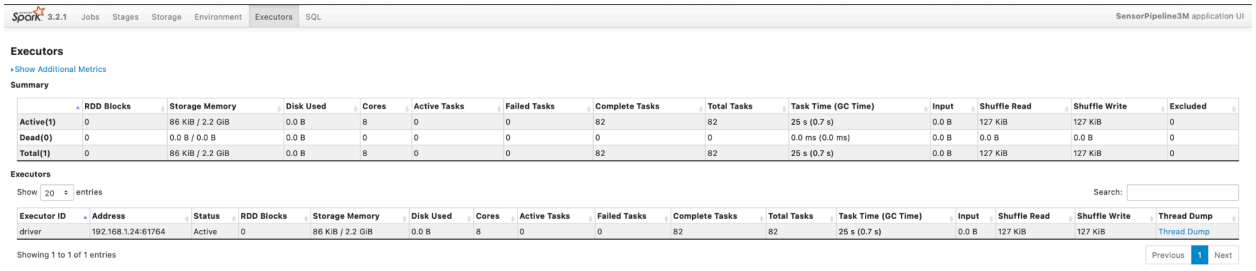
Jobs



Stages



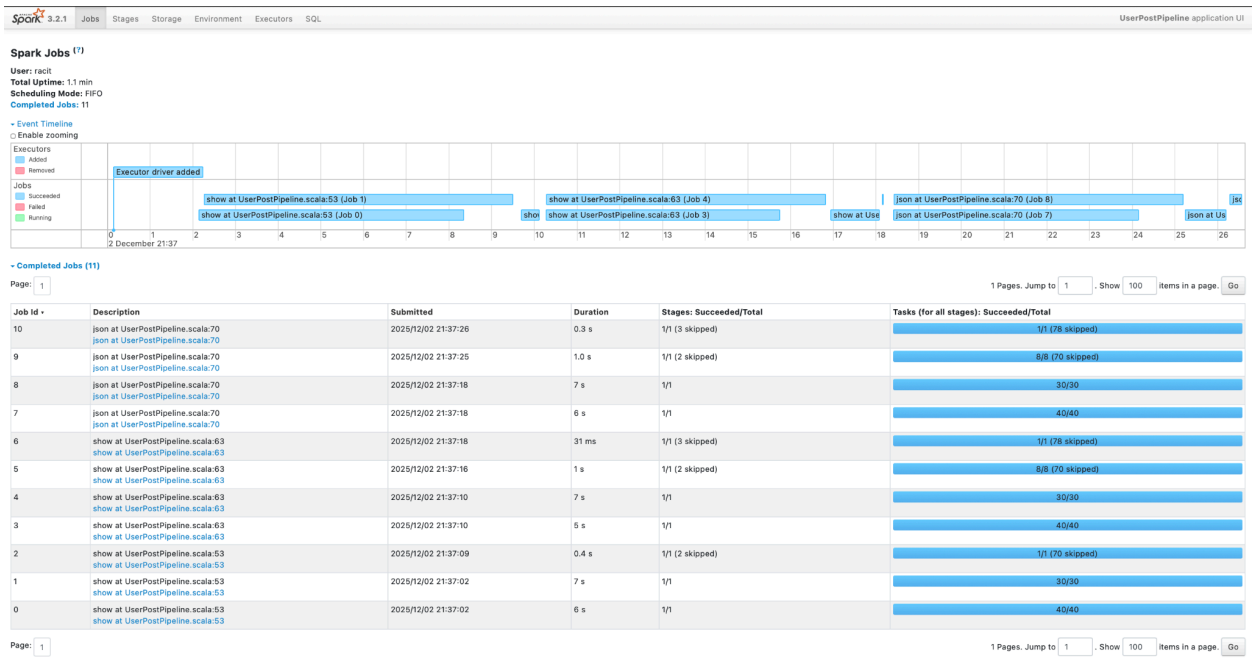
Executors



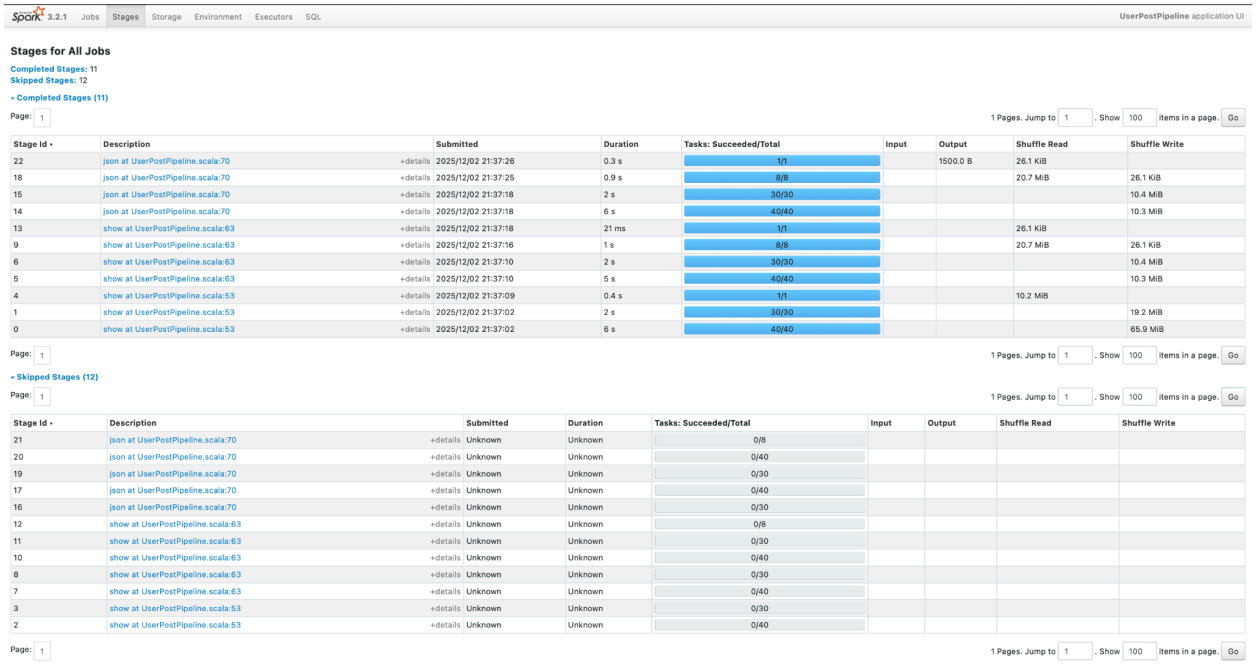
Observation	Summary
Output folders created	**24 folders** (one per hour).
Why groupBy is broad	Needs a **shuffle** to group all same-hour records together.

6. Social media posts

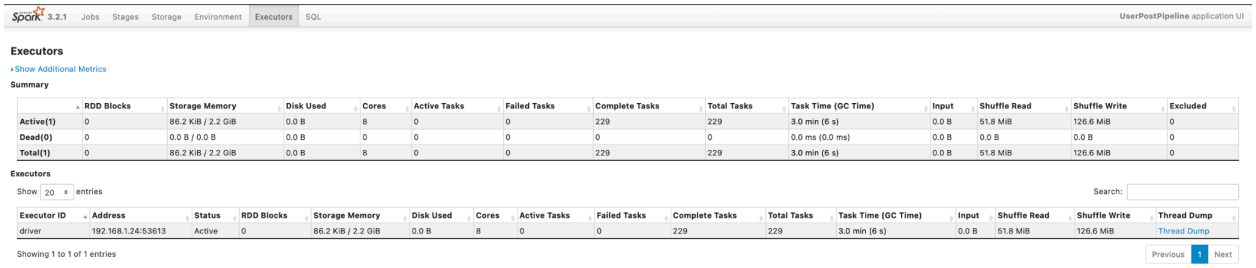
Jobs



Stages



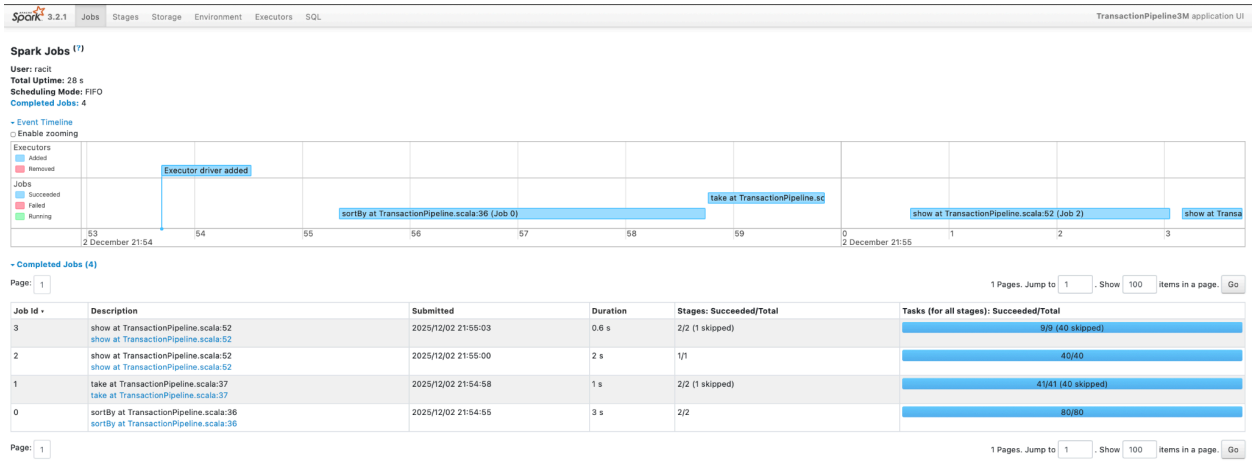
Executors



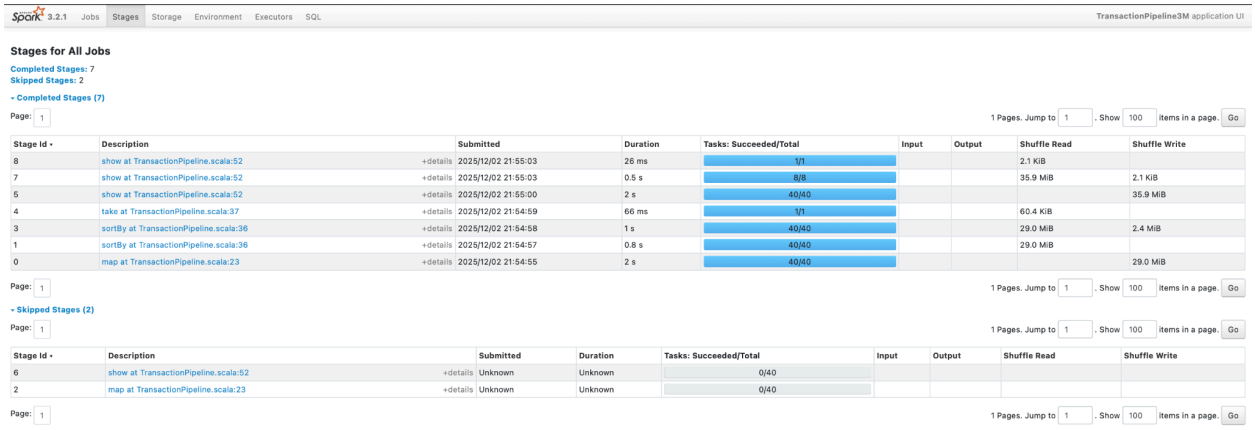
Observation	Summary
Why join is broad?	Needs a **shuffle** to move the same userId records to one partition.
Why DF join is easier?	Auto-optimized by Catalyst; less code than RDD key-value joins.

7. Financial Transactions

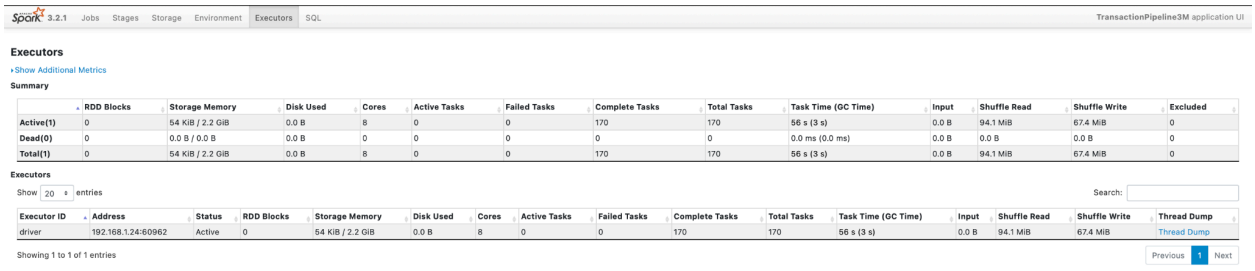
Jobs



Stages



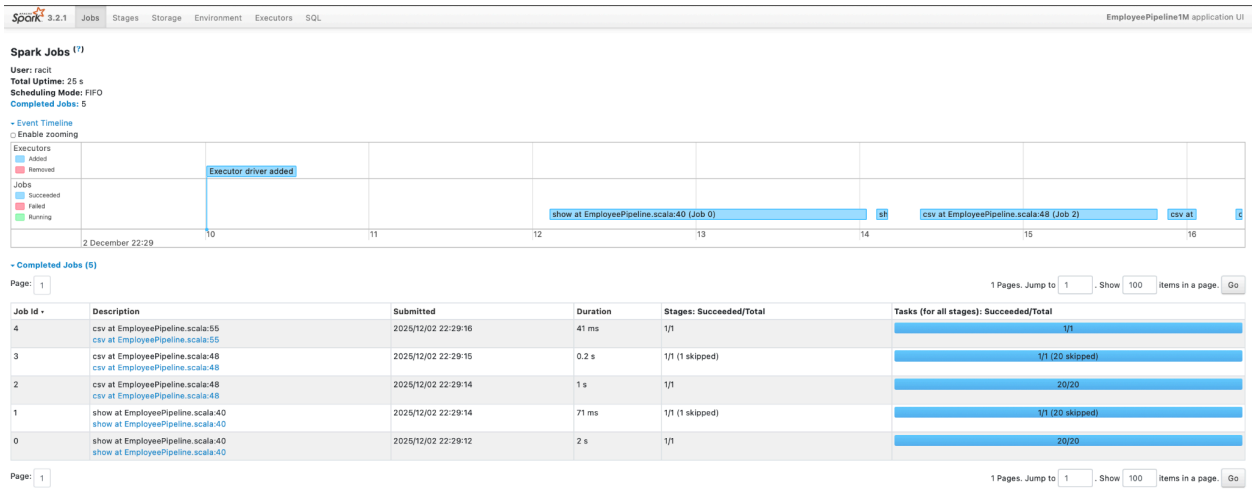
Executors



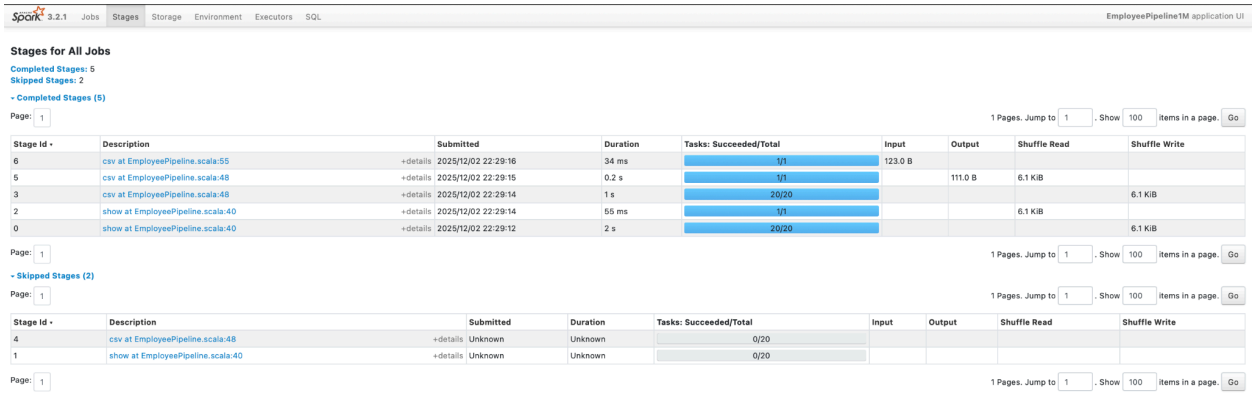
Observation	Summary
Sorting	Broad — needs a full shuffle.
Grouping	Broad — keys must be shuffled together.
Why DF faster?	Catalyst + Tungsten give optimized execution and lower overhead.

8. Employee Data

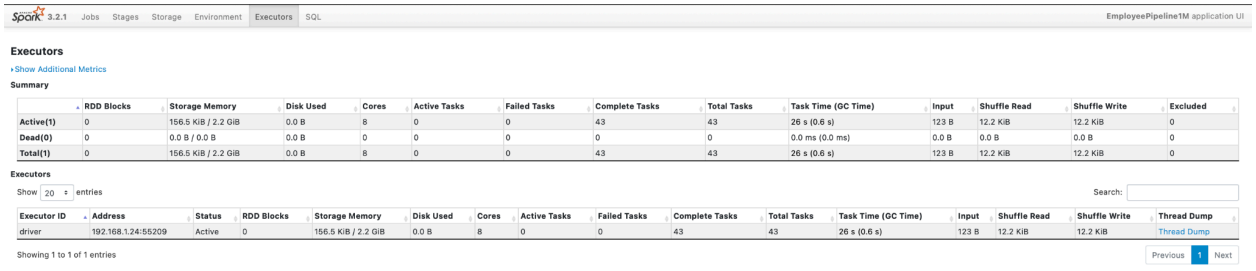
Jobs



Stages



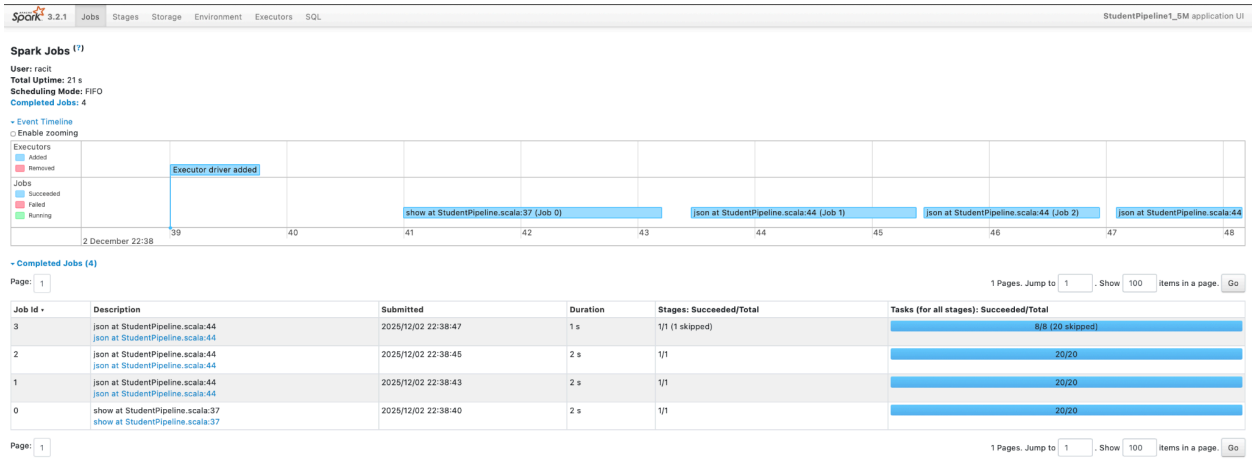
Executors



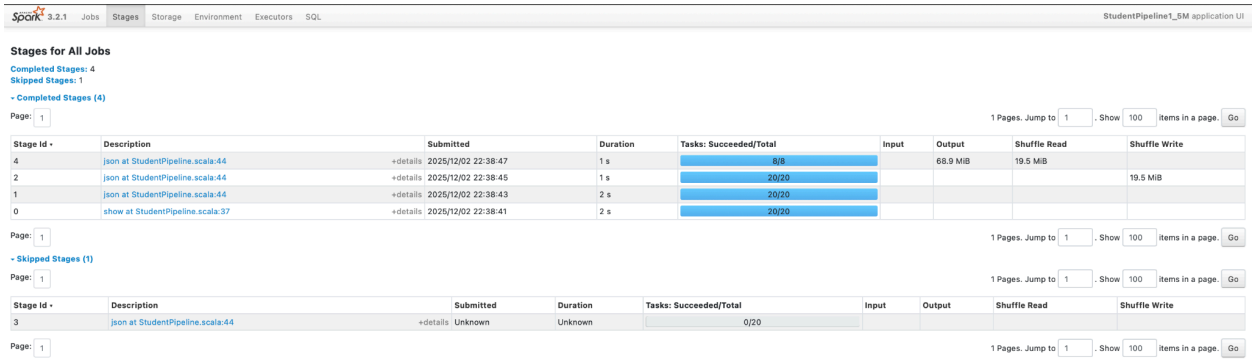
Observation	Summary
CSV schema inference	CSV loads **all columns as STRING**, because CSV has no schema/metadata.
Parquet behavior	Parquet **preserves original data types** since it stores full schema internally.

9. Student Scores

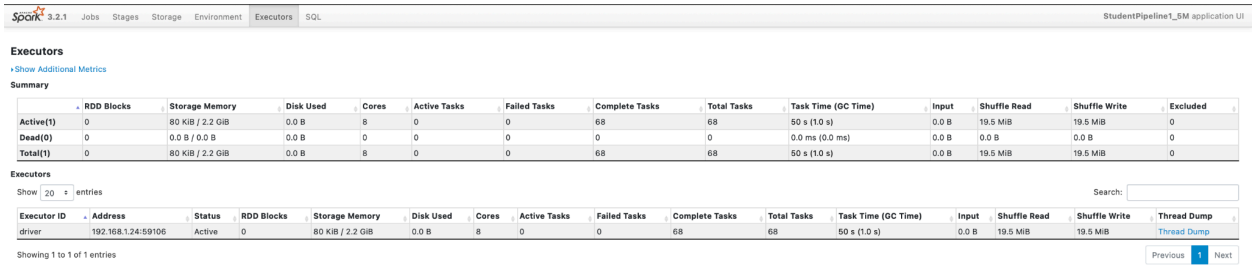
Jobs



Stages



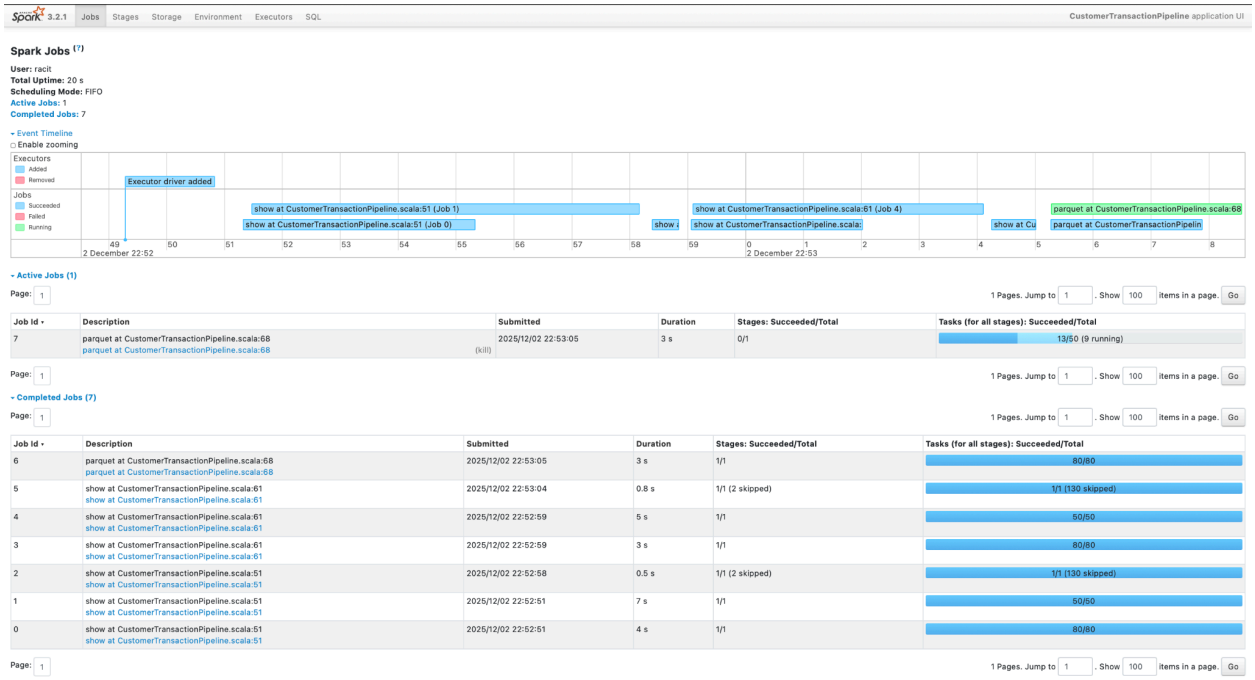
Executor



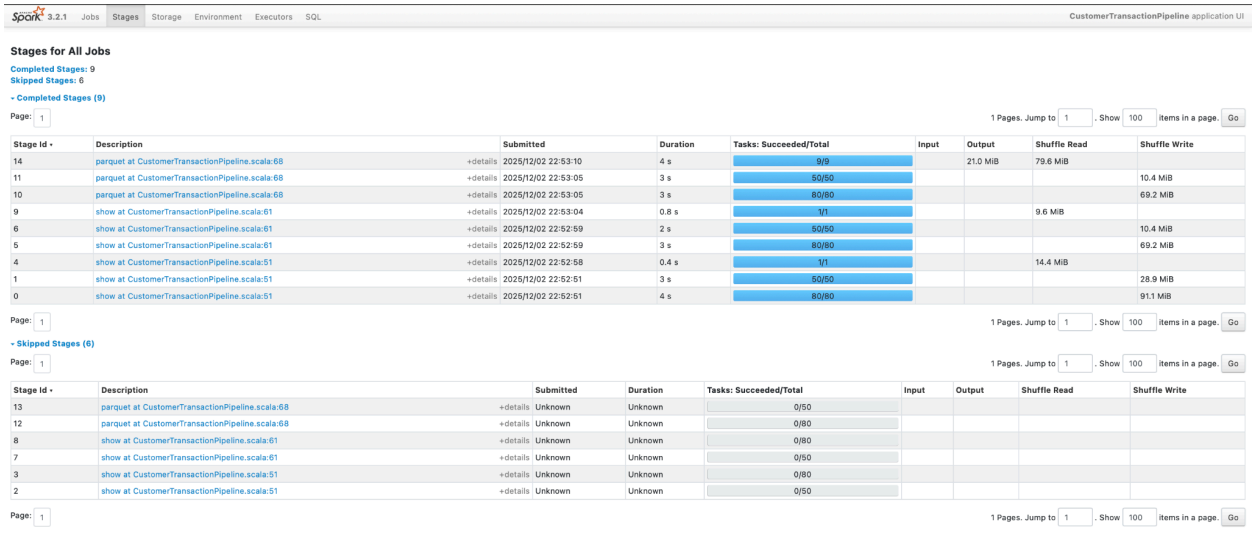
Observation	Summary
Sorting	Broad — requires a full shuffle to globally order all students by score.
JSON Writing	Slowest format — verbose text, repeated field names, large output size.

10. Customer Transaction

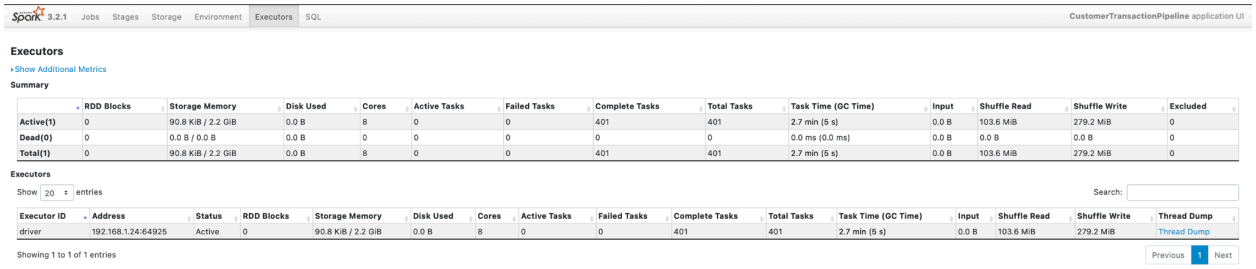
Jobs



Stages



Executors



Observation	Summary
Why joins & groupBy heavy?	Both need **full shuffle** so same keys go to the same partition → biggest data movement.
Why Parquet best?	**Columnar + compressed + schema-aware** → smallest size and fastest for analytics.