**Q1) Why do we have 200 partitions in wide transformation?**

When we perform a wide transformation (group by, join, window function, sort) there is a shuffle(redistribution) of data. During this shuffle, new partitions get created or removed. E.g If we use row\_number() function it will reduce the number of partition to 1.

The smaller size of partitions (more partitions) will increase the parallel running jobs, which can improve performance, but too small of a partition will cause overhead and increase the GC time. Larger partitions (fewer number of partitions) will decrease the number of jobs running in parallel.

* Performance: Having too many partitions can lead to overhead in task scheduling and communication overhead due to excessive partitioning.
* Resource Utilization: Each partition is processed by a task, and managing a large number of partitions can consume significant memory and processing resources.
* Optimization: Limiting the number of partitions can help in optimizing certain operations, especially those involving wide transformations, by reducing the amount of data shuffling and improving performance.
* Ease of Management: Working with a limited number of partitions can simplify resource management and tuning for the Spark cluster.

**Q2) Explore persist option. How will it work?**

By persisting a dataset, you can keep the data in memory or on disk so that it can be quickly retrieved the next time it is needed.

This can significantly improve the performance of iterative or multiple operations by avoiding re-evaluation of the data lineage and recomputation of RDDs or DataFrames.

When you call persist(), Spark evaluates the DataFrame or RDD up to that point and stores the partitions in memory or disk based on the storage level specified.

The storage levels include

MEMORY\_ONLY,MEMORY\_AND\_DISK, MEMORY\_ONLY\_SER, MEMORY\_AND\_DISK\_SER, DISK\_ONLY, and others, each with different trade-offs in terms of performance and fault tolerance.

1. persist() computes the data lineage up to that point and stores partitions in memory or disk based on chosen storage level.
2. Cached partitions are stored on worker nodes; overflow may be spilled to disk depending on storage level.
3. Lazy evaluation persists data without immediate computation; actual computation occurs upon action.
4. Persisted data can be reused across multiple actions or transformations, enhancing performance.
5. Caches can be explicitly invalidated with unpersist() or dropped due to memory pressure or storage policies.