**SPARK QUESTIONS – HOMEWORK TASK**

**1. Why there are 200 partitions in wide transformation?**

In Spark, optimization is achieved through lazy evaluation, which means tasks are deferred until needed. This process is divided into two main parts: Actions and Transformations. Transformations are further categorized into Narrow Transformations and Wide Transformations.

Wide Transformations involve moving data between different executors. By default, when wide transformations are performed, Spark uses 200 partitions to handle data shuffling and sorting operations.

Data shuffling is necessary for certain operations like grouping, reducing, joining, and sorting. The number of partitions is crucial for distributing data effectively. Depending on the situation, having too few large partitions can lead to issues like disk spills and out-of-memory errors, while having too many small partitions can overwhelm the driver and cause high network overhead. Hence, Spark uses 200 partitions for wide transformations by default because it's a good balance for handling tasks like sorting and shuffling data across different parts of the system. This number helps to distribute the workload effectively without causing problems like running out of memory or creating too much network traffic.

**2. Mention various persist methods in Spark.**

Spark provides a convenient way to work on the dataset by persisting it in memory across operations. While persisting an RDD, each node stores any partitions of it that it computes in memory. Now, we can also reuse them in other tasks on that dataset.

1. **MEMORY\_ONLY:** Stores data in memory as Java objects. If data doesn't fit in memory, some partitions are recomputed when needed.
2. **MEMORY\_AND\_DISK:** Stores data in memory as Java objects. If data doesn't fit in memory, spills over to disk and reads from disk when needed.
3. **MEMORY\_ONLY\_SER:** Stores data in memory as serialized Java objects, which is more space-efficient than deserialized objects.
4. **MEMORY\_AND\_DISK\_SER:** Similar to MEMORY\_ONLY\_SER but spills over to disk if data doesn't fit in memory.
5. **DISK\_ONLY:** Stores data only on disk.
6. **MEMORY\_ONLY\_2, MEMORY\_AND\_DISK\_2, etc.**: Same as above but replicates each partition on two cluster nodes for fault tolerance.
7. **OFF\_HEAP (experimental):** Stores data in off-heap memory as serialized Java objects.