When implementing joins in Spark DataFrames, these are the give join strategies that the Spark's Catalyst Optimizer will try to infer and selected the most appropriate one based on the data size, distribution and join type. Below are the five join strategies in Spark:

1. Broadcast Hash Join (BHJ)

   If one of the datasets is small enough to fit in the memory of each executor, Spark will "broadcast" it to all executors as a reference to do a hash join on the larger dataset in the partition that the executor is being assigned.

   The advantage is that it is fast for small datasets as it avoids shuffling thru the larger dataset by minimizing the I/O.

   The disadvantage is that if the broadcasted dataset is too large for the executor's memory, it will throw an OutOfMemory (OOM) error.

2. Shuffle Hash Join (SHJ)

   When both dataset is not small enough for a BHJ, both datasets are partitioned (shuffled) based on the join key and making sure the rows have the same join key in the same executor. After the shuffling (default is 200 partitions or per the config), a hash table is created for lookup based on two partitions and a hash join happens in-memory.

   The advantage is that if both datasets are too large for BHJ, but one of the datasets is small enough, the partitions/load is balanced across the executors to perform the join.

   The disadvantage is that it will do a full shuffle (partitioning), which may be affected by I/O bottlenecks. And if the data is skewed, that partition may become a bottleneck when using SHJ.

3. Shuffle Sort-Merge Join (SMJ)

   Very similar to SHJ, but repartitioning on both datasets will happen based on joining keys and sorted within the partitions. After sorting, Spark will then merge the two sorted partitions based on joining key.

   The advantage is good for large datasets that cannot fit in memory and handles data skew better than hash join, SHJ method above.

   The disadvantage is that it is also the same as SHJ, it requires a full shuffle. On top of that, a sort operation needs to happen before merging, hence the compute is expensive.

4. Cartesian Product Join (CPJ)

As the name suggests, it is a join of each record from one dataset with every row in the other dataset.

This has no benefits and usually is costly and can lead to Out Of Memory (OOM) when the memory of the executors runs out.

5. Broadcast Nested Loop Join (BNLJ)

   It is very similar to the above CPJ strategy, but one of the datasets is smaller than the other. The smaller dataset will be broadcasted to all the executor for the non-equi join to happen just like a cartesian product join.

   There is also not much benefits when doing such join. It is costly and extremely slow when the larger dataset is too large. It will also encounter Out Of Memory (OOM) errors when the executors run out of memory.

For the scenario in the question, I will adopt the BHJ method for joining the Dataset A with Dataset B. Dataset B has a size of 10,000 rows, which should be significantly small enough to be broadcasted to the executors for the join to happen. To make tell Spark to use Broadcast Join, I can set the `spark.sql.autoBroadcastJoinThreshold` according to my needs. If needed, a broadcast(small_df) hint can be used to tell Spark to use BHJ strategy when joining the two datasets.

References:

1. https://www.linkedin.com/pulse/spark-join-strategies-mastering-joins-apache-venkatesh-nandikolla-mk4qc/
2. https://oindrila-chakraborty88.medium.com/different-types-of-join-strategies-in-apache-spark-5c0066999d0d