Spark Joins

Data Exchange Strategies in Spark

When Spark joins datasets, it needs to move data between workers (executors). How it moves data depends on the join strategy. There are two main ways:

1. Broadcast

- The small table is sent (broadcast) to all executors.
- Each executor keeps a copy of this small table in memory.
- The **large table** stays where it is, no shuffling
- **Benefit:** Very fast because it avoids heavy shuffling.

2. Shuffle

- Both datasets are shuffled across all executors.
- All rows with the same join key end up on the same executor.
- Every executor may send data to every other executor.
- **Cost:** Expensive because of network traffic and I/O.

Spark Join Strategies

When deciding how to join datasets in Spark, there are 3 considerations

1. **Table Size** – How big is each dataset?
2. **Join Condition** – Are we joining with `=` or something else like `<` or `>`?
3. **Join Type** – Inner, outer, left, right, etc.

---

1. Broadcast Hash Join (BHJ)

How it works:

- Build a hash table on the smaller dataset using the join key.
- broadcast the small dataset to all workers.
- Look up the join key in the hash table for each row in the larger dataset.

Good for:

- Small dataset fits in memory
- Equality (`=`) joins only
- Any join type except full outer
- Avoids shuffling large data

---

2. Shuffle Hash Join (SHJ)

How it works:

- Like BHJ, but data is **shuffled** so rows with the same join key end up in the same worker.

Good for:

- Small dataset fits in memory per partition
- Equality (`=`) joins only
- All join types, including full outer
- Risk of skew if join key is uneven
- Use BHJ if possible to avoid shuffle.

---

3. Sort Merge Join (SMJ)

How it works:

- Sort both datasets by the join key.
- Merge the sorted data without checking every combination.

Good for:

- Large datasets
- Equality (`=`) joins only
- Any join type
- No risk of hash table OOM

---

4. Broadcast Nested Loop Join (BNLJ)

How it works:

- For each row in one dataset, check all rows in the small dataset.
- Can handle any join condition, not just `=`.

Good for:

- Small dataset fits in memory
- Equality or non-equality joins
- Any join type
- Expensive because it loops through all rows

---

5. Cartesian Product Join (CPJ)

How it works:

- Combines every row from one dataset with every row in the other.

Good for:

- Both datasets must be small

- Only inner joins
- Very slow and memory-heavy

Our Case

- We have `topX` items per geo, size = X * unique geo id, which is likely much larger than `geoNames` dataset (10k).
- Join is on equality (`geographical_location_oid`).
- Simple inner join is enough.

Use BHJ

Why:

- `geoNames` is tiny → safe to broadcast
- Join is equality → hash table works
- Avoids shuffling the larger `topX` dataset
- Fast and memory-efficient