Streaming Joins & Statefulness :-

⇒ Spark Structured Streaming is a Stateful Stream processing engine.

⇒ Most streaming joins are considered to be Stateful, because their progress are tracked in batches, and incrementally updated.

State Explosion :-
It is the condition, when state is not properly discarded, and so it scales towards infinity over time.

⇒ Tuning state information is must in near-realtime processing.

⇒ Components in a Stateful Stream:

⇒ These components decide when the records of the stream are materialised :-

1> Output Modes :-

append, complete, update.

=> Output modes interact with watermarks, to decide how long to wait to write results, and what results should be written after each batch processed.

append :- Waits for all records being processed and then records are updated.

complete :- Gets the up-to-date, but potencially partially processed records, with each trigger. aggregated

update :- Only o/p values that changed since last trigger.

Statefulness vs. Query Progress

- Many operations as specifically stateful (stream-stream joins, deduplication, aggregation)
- Some operations just need to store incremental query progress and are not stateful (appends with simple transformations, stream-static joins, merge)
- Progress and state are stored in checkpoints and managed by driver during query processing

=> When Streaming, many things can grow in size over time :-

No of intermediate states static dataframes size, no. of records in each batch

These can affect the scalability of streaming jobs.

⇒ During near-real time stream processing, memory spill degrades the efficiency.

1) Input Parameters :-

- Amount of data in each micro batch.
- Optimal microbatch size leads to optimal cluster usage without spill.

Per Trigger Settings

- File Source → .format("json" / "csv" / "avro")
  - maxFilesPerTrigger
- Delta Lake and Auto Loader → .format("delta") / .format("cloudFiles")
  - maxFilesPerTrigger
  - maxBytesPerTrigger
- Kafka
  - maxOffsetsPerTrigger

Shuffle Partitions with Structured Streaming

- Should match the number of cores in the largest cluster size that might be used in production
  - Number of shuffle partitions == max parallelism
- Cannot be changed without new checkpoint
  - Will lose query progress and state information
- Higher shuffle partitions == more files written
- Best practice: use Delta Live Tables for streaming jobs with variable volume

spark.conf.set("spark.sql.shuffle.partitions", sc.defaultParallelism)
else, default
200

DLT allows autoscaling

Ideally:-

1 shuffle partition per executor core

⇒ Tuning Per Trigger settings should be done after or in sync with cluster size and no. of shuffle partitions.

Tuning maxFilesPerTrigger

Base it on shuffle partition size

- **Rule of thumb 1:** Optimal shuffle partition size ~100–200 MB
- **Rule of thumb 2:** Set shuffle partitions equal to # of cores
- Use Spark UI to tune `maxFilesPerTrigger` until you get ~100–200 MB per partition
- Note: Size on disk is **not** a good proxy for size in memory
  - Reason is that file size is different from the size in cluster memory

when working with highly compressed file formats like Parquet, data loaded into memory will be much larger than size on disc.

⇒ Limiting the State Dimension:-

⇒ It means limiting the no. of records that

needs to be considered when processing logic for each microBatch of input data.

⇒ If not managed, will lead to State Explosion

⇒ Important things to control State Explosion :-

(i) Always watermark queries that uses the state store.

(ii) Reducing the granularity / freq. of sampling. Reduce the total no. of unique keys, for which the state information is retained for.

State Information is kept in memory in the JVM, by default.

RocksDB can be used to offload this State Information, to attached SSD volume storage.

This is better as Garbage Collection takes huge time.

Stream-Static Join & Merge

- Join driven by streaming data
- Join triggers shuffle
- Join itself is stateless
- Control state information with predicate
- Goal is to broadcast static table to streaming data
- Broadcasting puts all data on each node

1. Main input stream

```
salesSDF = (
  spark
    .readStream
    .format("delta")
    .table("sales")
)
```

2. Join item category lookup

```
itemSalesSDF = (
  salesSDF
    .join(
      spark.table("items")
       .filter("category='Food'"), # Predicate
      on=["item_id"]
    )
)
```

Avoiding shuffle through Broadcasting drastically improves Stream throughput.
Thus, Static side of Stream-Static join must be kept small, and broadcasted.

Predicates = Join condition / expression that forms the basis of Joins.

Why are output parameters important?

- Streaming jobs tend to create many small files
  - Reading a folder with many small files is slow
  - Poor performance for downstream jobs, self-joins, and merge
- Output type can impact state information retained
- Merge statements with full table scans increase state

For .outputMode("complete"), entire state info is retained in memory, so that aggregation based on entire table can be done.

Delta Lake Output Optimizations

- Optimized Writes → reduces no. of o/p files.
- Auto Compaction → identifies partitions in target tbl that will benefit from compaction.
- delta.tuneFileSizesForRewrites → optimises file size for more efficient execution.
- Insert-only merge