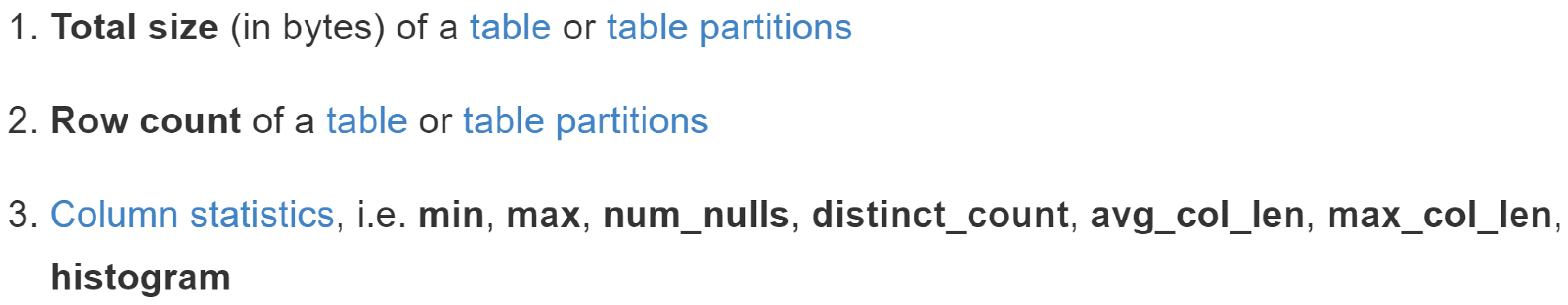
Optimization in Spark

- **Spark 1.x**: Catalyst Optimizer (e.g., predicate pushdown and projection pushdown) and Tungsten Project (CPU, cache, and memory efficiency, eliminate the overhead of JVM objects and garbage collection).

- **Spark 2.x**: Cost-Based Optimizer (CBO) to improve queries with multiple joins., using table statistics to determine the most efficient query execution plan.



1. **Total size** (in bytes) of a table or table partitions
2. **Row count** of a table or table partitions
3. Column statistics, i.e. **min**, **max**, **num_nulls**, **distinct_count**, **avg_col_len**, **max_col_len**, **histogram**

Optimization in Spark

- **Spark 3.x**: Adaptive Query Execution (AQE) to Speed Up Spark SQL at Runtime, based on runtime statistics collected during the execution of the query.

- It resolve the biggest drawback of CBO, by making the balance between the stats collection overhead and the estimation accuracy.

- AQE is false by default in Spark 3.0 and can be enabled by

```
spark.conf.set("spark.sql.adaptive.enabled",true)
```

Author: Amin Karami (PhD, FHEA) amin.karami@ymail.com

Adaptive Query Execution Features

1. Dynamically coalescing Post Shuffle Partitions

2. Dynamically optimizing Skew Join

3. Dynamically switching join strategies (converting sort-merge join to broadcast join or shuffled hash join)

Source:
https://spark.apache.org/docs/latest/sql-performance-tuning.html#adaptive-query-execution

1) Coalescing Post Shuffle Partitions

- When a Big Data developer applies a shuffling operation (such as `GroupBy`), the developer need to re-partition to increase or coalesce to decrease the partitions. The default number of partitions after shuffling is 200.

- It is impossible to guess the optimal number of partitions. The manual configuration is `spark.sql.shuffle.partitions`.

- AQE dynamically compute the statistics on your data during shuffling/sorting to determine the optimal number of partitions (**V3.0**).

2) Optimizing Skew Join

- Data skew can severely downgrade the performance of join queries. Operations such as join perform very slow on these partitions (**V3.0**).

- This feature dynamically handles skew by splitting (and duplicating if needed) skewed tasks into roughly evenly sized tasks.

3) Converting sort-merge join to broadcast join

- This strategy can be used only when one of the join tables is small enough to fit into memory within the broadcast threshold **(V3.2)**.

- In this case, AQE re-plans the join strategy at runtime and applies broadcast hash join.

```
spark.conf.set("spark.sql.adaptive.enabled",True)
spark.conf.set("spark.sql.join.preferSortMergeJoin", "false")
spark.conf.set("spark.sql.adaptive.autoBroadcastJoinThreshold", “100m")
# default: 10485760 bytes (10m)
```

3) Converting sort-merge join to shuffled hash join

- It shuffles both datasets (**Shuffle phase**). So the same keys from both sides end up in the same partition. Then, the smallest of the two will be hashed into buckets and a hash join is performed within the partition (**Hash Join phase**) **(V3.2)**.

- AQE converts sort-merge join to shuffled hash join when all post shuffle partitions are smaller than a threshold. It does not work with massive data or heavily skewed data. NOT recommended.

```
spark.conf.set("spark.sql.adaptive.enabled",True)
spark.conf.set("spark.sql.join.preferSortMergeJoin", "false")
spark.conf.set("spark.sql.adaptive.maxShuffledHashJoinLocalMapThreshold
", "100m")
```