Caching

Advantages

- Speed up results when a table is used in multiple queries.
- Gives the node faster access by storing it locally in memory.

```
CACHE TABLE table;
```

- Caching can be lazy, e.g. not right now but until required.

```
CACHE LAZY TABLE table;
```

- When no longer needed, is good idea to uncache.

```
UNCACHE TABLE table;
```

Python analogues

```
df.cache()
df.unpersist()
```

Delta Lake caching

- For tables in Delta Lake, caching is automatic and it happens in *disk*, not in *memory*.
- Delta Lake caching can be set when choosing the worker type when you configure your cluster.
- Some types of workers support it, but not enabled by default, see docs.

Example: PySpark

```
%sql
use taxidata;
create table yellow_tripdata as select * from yellow_tripdata_2021_01_csv;
```

```
df = spark.read.table('yellow_tripdata')

from pyspark.sql.functions import sum
distance_sum = df \
                .select(["trip_distance","payment_type", "store_and_fwd_flag"])\
                .filter(df.trip_distance>2)\
                .groupBy("payment_type", "store_and_fwd_flag")\
                .agg(sum("trip_distance"))
```

Example: PySpark (cont.)

```
trips_cached =  df \
                .select(["trip_distance","payment_type", "store_and_fwd_flag"])\
                .filter(df.trip_distance>2)\
                .cache()
trips_cached.count()

distance_sum_cached = trips_cached \
                      .groupBy("payment_type", "store_and_fwd_flag")\
                      .agg(sum("trip_distance"))
```

Example: PySpark (cont.)

```
distance_sum.show()
distance_sum_cached.show()
```

Comparison with SparkSQL

```
sql = spark.sql(' \
         SELECT payment_type, store_and_fwd_flag, SUM(trip_distance)\
         FROM yellow_tripdata \
         WHERE trip_distance > 2 \
         GROUP BY payment_type, store_and_fwd_flag;')

sql.show()
```