databricks8.3 Optimizing Delta

Optimizing Delta

In this notebook, you'll see some examples of how you can optimize your queries using Delta Engine, which is built-in to the Databricks Runtime 7.0. It is also part of open source Delta Lake (https://delta.io/).

The data contains information about US-based flight schedules from 2008. It is made available to us via Databricks Datasets (https://docs.databricks.com/data/databricks-datasets.html).

First, we will create a standard table using Parquet format and then we'll run a query to observe the timing.

Then, we'll run the same query on a Delta table using Delta Engine optimizations and compare the two

```
%run ../Includes/Classroom-Setup
```

Mounting course-specific datasets to **/mnt/training**...
Datasets are already mounted to **/mnt/training** from **s3a://databricks-corp-training/common**

```
res1: Boolean = true
```

```
res2: Boolean = true
```

Create a Parquet table

Run the command below to create a Parquet table.

```
DROP TABLE IF EXISTS flights;
-- Create a standard table and import US based flights for year 2008
-- USING Clause: Specify parquet format for a standard table
-- PARTITIONED BY clause: Orginize data based on "Origin" column (Originating
Airport code).
-- FROM Clause: Import data from a csv file.
CREATE TABLE flights
USING
  parquet
PARTITIONED BY
  (Origin)
SELECT
  _c0 AS Year,
  _c1 AS MONTH,
  _c2 AS DayofMonth,
  _c3 AS DayOfWeek,
  _c4 AS DepartureTime,
  _c5 AS CRSDepartureTime,
  _c6 AS ArrivalTime,
  _c7 AS CRSArrivalTime,
  _c8 AS UniqueCarrier,
  _c9 AS FlightNumber,
  _c10 AS TailNumber,
  _c11 AS ActualElapsedTime,
  _c12 AS CRSElapsedTime,
  _c13 AS AirTime,
  _c14 AS ArrivalDelay,
  _c15 AS DepartureDelay,
  _c16 AS Origin,
  _c17 AS Destination,
  _c18 AS Distance,
  _c19 AS TaxiIn,
  _c20 AS TaxiOut,
  _c21 AS Cancelled,
  _c22 AS CancellationCode,
  _c23 AS Diverted,
  _c24 AS CarrierDelay,
  _c25 AS WeatherDelay,
  _c26 AS NASDelay,
  _c27 AS SecurityDelay,
  _c28 AS LateAircraftDelay
FROM                                -- This table is being read in directly from
a csv file.
  csv.`dbfs:/databricks-datasets/asa/airlines/2008.csv`
```

OK

Highest monthly total (Parquet)

Run the query to get the top 20 cities with the highest monthly total flights on the first day of the week. Be sure to note the time when the query finishes.

```
SELECT Month, Origin, count(*) as TotalFlights
FROM flights
WHERE DayOfWeek = 1
GROUP BY Month, Origin
ORDER BY TotalFlights DESC
LIMIT 20;
```

	Month	Origin	TotalFlights
1	6	ATL	6046
2	3	ATL	6019
3	12	ATL	5800
4	9	ATL	5722
5	6	ORD	5241
6	3	ORD	5072
7	9	ORD	4931

Showing all 20 rows.

Create a Delta Table

Run the query below to compare Delta to Parquet. Note, this is the exact same command running on the exact same cluster configuration. Recall that the two operations take roughly the same amount of "work" from Spark. We have to read in a huge csv file, partition it by origin, and store it in a new, columnar format. Plus, Delta is creating a transaction log and tagging the files with important and useful metadata!

```
DROP TABLE IF EXISTS flights;
-- Create a standard table and import US based flights for year 2008
-- USING Clause: Specify "delta" format instead of the standard parquet format
-- PARTITIONED BY clause: Orginize data based on "Origin" column (Originating
Airport code).
-- FROM Clause: Import data from a csv file.
CREATE TABLE flights
USING
  delta
PARTITIONED BY
  (Origin)
SELECT
  _c0 AS Year,
  _c1 AS MONTH,
  _c2 AS DayofMonth,
  _c3 AS DayOfWeek,
  _c4 AS DepartureTime,
  _c5 AS CRSDepartureTime,
  _c6 AS ArrivalTime,
  _c7 AS CRSArrivalTime,
  _c8 AS UniqueCarrier,
  _c9 AS FlightNumber,
  _c10 AS TailNumber,
  _c11 AS ActualElapsedTime,
  _c12 AS CRSElapsedTime,
  _c13 AS AirTime,
  _c14 AS ArrivalDelay,
  _c15 AS DepartureDelay,
  _c16 AS Origin,
  _c17 AS Destination,
  _c18 AS Distance,
  _c19 AS TaxiIn,
  _c20 AS TaxiOut,
  _c21 AS Cancelled,
  _c22 AS CancellationCode,
  _c23 AS Diverted,
  _c24 AS CarrierDelay,
  _c25 AS WeatherDelay,
  _c26 AS NASDelay,
  _c27 AS SecurityDelay,
  _c28 AS LateAircraftDelay
FROM
  csv.`dbfs:/databricks-datasets/asa/airlines/2008.csv`;
```

Query returned no results

Optimize your table

If your organization continuously writes data to a Delta table, it will over time accumulate a large number of files, especially if you add data in small batches. For analysts, a common complaint in querying data lakes is read efficiency; and having a large collection of small files to sift through everytime data is queried can create performance problems. Ideally, a large number of small files should be rewritten into a smaller number of larger files on a regular basis, which will improve the speed of read queries from a table. This is known as compaction. You can compact a table using the `OPTIMIZE` command shown below.

Z-ordering co-locates column information (recall that Delta is columnar storage). Co-locality is used by Delta Lake data-skipping algorithms to dramatically reduce the amount of data that needs to be read. You can specify multiple columns for ZORDER BY as a comma-separated list. However, the effectiveness of the locality drops with each additional column. Read more about optimizing Delta tables here (https://docs.databricks.com/spark/latest/spark-sql/language-manual/delta-optimize.html).

```
OPTIMIZE flights ZORDER BY (DayofWeek);
```

	path	metrics
1	null	▸ {"numFilesAdded": 300, "numFilesRemoved": 2308, "filesAdded": {"min": 8696 "totalFiles": 300, "totalSize": 115236424}, "filesRemoved": {"min": 6376, "max": 1 "totalFiles": 2308, "totalSize": 144926126}, "partitionsOptimized": 304, "zOrderSt "minCubeSize(107374182400)", "inputCubeFiles": {"num": 0, "size": 0}, "inputOth "inputNumCubes": 0, "mergedFiles": {"num": 2308, "size": 144926126}, "numOut "numBatches": 1, "totalConsideredFiles": 2312, "totalFilesSkipped": 4, "preserve

Showing all 1 rows.

Rerun the query

Run the query below to compare performance for a standard Parquet table with an optimized Delta table.

```
SELECT
*
FROM
health_tracker_data_2020_silver
LIMIT 5;SELECT Month, Origin, count(*) as TotalFlights
FROM flights
WHERE DayOfWeek = 1
GROUP BY Month, Origin
ORDER BY TotalFlights DESC
LIMIT 20;
```

	Month	Origin	TotalFlights
1	6	ATL	6046
2	3	ATL	6019
3	12	ATL	5800
4	9	ATL	5722
5	6	ORD	5241
6	3	ORD	5072
7	9	ORD	4931

Showing all 20 rows.

Delta Cache

Using the Delta cache is an excellent way to optimize performance. Note: The Delta cache is *not* the same as caching in Apache Spark, which we talked about in Module 4. One notable difference is that the Delta cache is stored entirely on the local disk, so that memory is not taken away from other operations within Spark. When enabled, the Delta cache automatically creates a copy of a remote file in local storage so that successive reads are significantly sped up. Unfortunately, to enable it, you must choose a cluster type that is not available in Databricks Community Edition.

To better understand the differences between Delta caching and Apache Spark caching, please read, "Delta and Apache Spark caching." (https://docs.databricks.com/delta/optimizations/delta-cache.html#delta-and-apache-spark-caching)

```
%run ../Includes/Classroom-Cleanup
```