In [0]:
```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.window import Window
from pyspark.sql.types import *
import time
```

Load data

In [0]:
```
# Load the COVID-19 dataset
print("\n[1] Loading COVID-19 Dataset...")
start_time = time.time()

# Load without schema inference to avoid type issues
df_covid = spark.read \
    .option("header", "true") \
    .option("inferSchema", "false") \
    .csv("/databricks-datasets/COVID/covid-19-data/")

# Clean and cast data types using expr with try_cast (handles malformed data gracefully)
df_covid = df_covid \
    .withColumn("date", to_date(col("date"), "yyyy-MM-dd")) \
    .withColumn("cases", coalesce(expr("try_cast(cases as int)"), lit(0))) \
    .withColumn("deaths", coalesce(expr("try_cast(deaths as int)"), lit(0)))

load_time = time.time() - start_time
print(f"Dataset loaded in {load_time:.2f} seconds")
print(f"Initial row count: {df_covid.count():,}")
print(f"Columns: {df_covid.columns}")
```

```
[1] Loading COVID-19 Dataset...
Dataset loaded in 0.04 seconds
Initial row count: 1,227,256
Columns: ['date', 'county', 'state', 'fips', 'cases', 'deaths']
```

In [0]:
```
# Test to see if the data is available

def get_directory_size(path):
    total_size = 0
    try:
```

```
        files = dbutils.fs.ls(path)
        for file in files:
            if file.isDir():
                total_size += get_directory_size(file.path)
            else:
                total_size += file.size
    except Exception as e:
        print(f"Error: {e}")
    return total_size

path = "/databricks-datasets/COVID/covid-19-data/"
size_bytes = get_directory_size(path)
size_mb = size_bytes / (1024 * 1024)
size_gb = size_bytes / (1024 * 1024 * 1024)

print(f"Total size: {size_bytes:,} bytes")
print(f"Total size: {size_mb:.2f} MB")
print(f"Total size: {size_gb:.2f} GB")
```

```
Total size: 2,567,706,254 bytes
Total size: 2448.76 MB
Total size: 2.39 GB
```

In [0]:
```
# Check the data

# Show schema
print("\nDataset Schema:")
df_covid.printSchema()

# Show sample data
print("\nSample Data:")
df_covid.show(5, truncate=False)
```

```
Dataset Schema:
root
 |-- date: date (nullable = true)
 |-- county: string (nullable = true)
 |-- state: string (nullable = true)
 |-- fips: string (nullable = true)
 |-- cases: integer (nullable = false)
 |-- deaths: integer (nullable = false)


Sample Data:
+----------+---------+----------+-----+-----+------+
|date      |county   |state     |fips |cases|deaths|
+----------+---------+----------+-----+-----+------+
|2020-01-21|Snohomish|Washington|53061|1    |0     |
|2020-01-22|Snohomish|Washington|53061|1    |0     |
|2020-01-23|Snohomish|Washington|53061|1    |0     |
|2020-01-24|Cook     |Illinois  |17031|1    |0     |
|2020-01-24|Snohomish|Washington|53061|1    |0     |
+----------+---------+----------+-----+-----+------+
only showing top 5 rows
```

Apply transformations

In [0]:
```python
# TRANSFORMATION 1: Filter Operations (Applied Early for Optimization)

# Filter 1: Select data from 2021 onwards
date_cols = [col for col in df_covid.columns if 'date' in col.lower()]
print(f"Date columns found: {date_cols}")

# Filter 2: Remove null values in key columns
df_covid_valid = df_covid.filter(
    expr("try_to_date(date, 'yyyy-MM-dd') IS NOT NULL")
)

df_filtered = df_covid_valid.filter(col("cases") > 0)

print(f"After filtering: {df_filtered.count():,} rows")
```

```
Date columns found: ['date']
After filtering: 1,224,611 rows
```

In [0]:
```
# TRANSFORMATION 2: Column Transformations using withColumn

df_transformed = df_filtered \
    .withColumn("year", year(col("date"))) \
    .withColumn("month", month(col("date"))) \
    .withColumn("quarter", quarter(col("date"))) \
    .withColumn("cases_per_death",
                when((col("deaths") > 0) & (col("deaths").isNotNull()),
                     col("cases").cast("double") / col("deaths").cast("double"))
                .otherwise(lit(None).cast("double"))) \
    .withColumn("is_high_cases", when(col("cases") > 10000, "High").otherwise("Low"))

print("New columns added: year, month, quarter, cases_per_death, is_high_cases")
df_transformed.show(5)
```

```
New columns added: year, month, quarter, cases_per_death, is_high_cases
+----------+---------+----------+-----+-----+------+----+-----+-------+---------------+-------------+
|      date|   county|     state| fips|cases|deaths|year|month|quarter|cases_per_death|is_high_cases|
+----------+---------+----------+-----+-----+------+----+-----+-------+---------------+-------------+
|2020-01-21|Snohomish|Washington|53061|    1|     0|2020|    1|      1|           NULL|          Low|
|2020-01-22|Snohomish|Washington|53061|    1|     0|2020|    1|      1|           NULL|          Low|
|2020-01-23|Snohomish|Washington|53061|    1|     0|2020|    1|      1|           NULL|          Low|
|2020-01-24|     Cook|  Illinois|17031|    1|     0|2020|    1|      1|           NULL|          Low|
|2020-01-24|Snohomish|Washington|53061|    1|     0|2020|    1|      1|           NULL|          Low|
+----------+---------+----------+-----+-----+------+----+-----+-------+---------------+-------------+
only showing top 5 rows
```

In [0]:
```
# TRANSFORMATION 3: GroupBy with Aggregations

# Aggregation 1: Monthly statistics by state/region
df_monthly_stats = df_transformed \
    .groupBy("state", "year", "month") \
    .agg(
        sum("cases").alias("total_cases"),
        sum("deaths").alias("total_deaths"),
        avg("cases").alias("avg_daily_cases"),
        max("cases").alias("max_daily_cases"),
        count("*").alias("num_records")
    ) \
    .withColumn("mortality_rate",
                (col("total_deaths") / col("total_cases") * 100).cast("decimal(10,2)"))
```

```
print("Monthly Statistics Aggregated")
df_monthly_stats.show(10)
```

```
Monthly Statistics Aggregated
+--------------+----+-----+-----------+------------+------------------+--------------+-----------+--------
------+
|         state|year|month|total_cases|total_deaths|   avg_daily_cases|max_daily_cases|num_records|mortalit
y_rate|
+--------------+----+-----+-----------+------------+------------------+--------------+-----------+--------
------+
|          Ohio|2020|    5|     847509|       49709| 310.5566141443752|          5862|       2729|
5.87|
|      Nebraska|2020|    5|     305559|        3838|145.78196564885496|          4314|       2096|
1.26|
| West Virginia|2020|    3|        866|           3| 4.224390243902439|            31|        205|
0.35|
|       Montana|2020|    3|       1200|          11| 5.555555555555555|            74|        216|
0.92|
|        Hawaii|2020|    5|      19430|         525|156.69354838709677|           421|        124|
2.70|
|      Nebraska|2020|    3|       1682|          13| 7.715596330275229|           113|        218|
0.77|
|        Oregon|2020|    3|       4621|         129|11.971502590673575|           186|        386|
2.79|
|South Carolina|2020|    5|     269994|       11736| 189.3366058906031|          1619|       1426|
4.35|
|         Texas|2020|    2|         91|           0| 5.055555555555555|            11|         18|
0.00|
| West Virginia|2020|    5|      47442|        1951|28.981062919975564|           308|       1637|
4.11|
+--------------+----+-----+-----------+------------+------------------+--------------+-----------+--------
------+
only showing top 10 rows
```

In [0]:
```
# Aggregation 2: Year-over-year comparison
df_yearly = df_transformed \
    .groupBy("state", "year") \
    .agg(
        sum("cases").alias("yearly_cases"),
        sum("deaths").alias("yearly_deaths"),
        avg("cases").alias("avg_cases_per_day")
    ) \
```

```
    .orderBy("state", "year")

print("\n Yearly Statistics:")
df_yearly.show(10)
```

```
 Yearly Statistics:
+-----+----+------------+-------------+------------------+
|state|year|yearly_cases|yearly_deaths| avg_cases_per_day|
+-----+----+------------+-------------+------------------+
|   01|2020|      526388|            0|1866.6241134751774|
|   01|2021|      552952|            0| 7899.314285714286|
|   02|2020|       13147|            0| 46.95357142857143|
|   02|2021|       17911|            0|255.87142857142857|
|   04|2020|     1048821|            0|3654.4285714285716|
|   04|2021|      937029|            0|13386.128571428571|
|   05|2020|      285868|            0|1010.1342756183745|
|   05|2021|      337571|            0| 4822.442857142857|
|   06|2020|     3065113|            0|10115.884488448844|
|   06|2021|     2940139|            0|42001.985714285714|
+-----+----+------------+-------------+------------------+
only showing top 10 rows
```

In [0]:
```
# TRANSFORMATION 4: Window Functions for Ranking
print("\n[5] Applying Window Functions...")

window_spec = Window.partitionBy("year").orderBy(desc("total_cases"))

df_ranked = df_monthly_stats \
    .withColumn("rank_by_cases", rank().over(window_spec)) \
    .filter(col("rank_by_cases") <= 10)

print("Top 10 states by cases per year:")
df_ranked.select("state", "year", "month", "total_cases", "rank_by_cases").show(20)
```

```
[5] Applying Window Functions...
Top 10 states by cases per year:
+----------+----+-----+-----------+-------------+
|     state|year|month|total_cases|rank_by_cases|
+----------+----+-----+-----------+-------------+
|California|2020|   12|   53958843|            1|
|     Texas|2020|   12|   47238530|            2|
|   Florida|2020|   12|   35950671|            3|
|     Texas|2020|   11|   33105809|            4|
|California|2020|   11|   31809169|            5|
|California|2020|   10|   27163983|            6|
|   Florida|2020|   11|   26828632|            7|
|  Illinois|2020|   12|   26821457|            8|
|     Texas|2020|   10|   26765048|            9|
|  New York|2020|   12|   25200181|           10|
|California|2021|    2|  160698825|            1|
|     Texas|2021|    2|  118293231|            2|
|California|2021|    1|   90315276|            3|
|   Florida|2021|    2|   84572548|            4|
|California|2021|    3|   79088430|            5|
|  New York|2021|    2|   71535329|            6|
|     Texas|2021|    1|   64910955|            7|
|     Texas|2021|    3|   59202086|            8|
|  Illinois|2021|    2|   53725198|            9|
|   Florida|2021|    1|   47886474|           10|
+----------+----+-----+-----------+-------------+
```

SQL queries

In [0]:
```
# Register DataFrame as temporary view
df_transformed.createOrReplaceTempView("covid_data")
df_monthly_stats.createOrReplaceTempView("monthly_stats")

# SQL Query 1: Top 10 states with highest total cases
print("\n[6] SQL Query 1: Top 10 States by Total Cases")
query1 = """
    SELECT
        state,
        SUM(cases) as total_cases,
```

```
        SUM(deaths) as total_deaths,
        ROUND(SUM(deaths) / SUM(cases) * 100, 2) as mortality_rate_pct
    FROM covid_data
    WHERE state IS NOT NULL
    GROUP BY state
    ORDER BY total_cases DESC
    LIMIT 10
"""

df_sql1 = spark.sql(query1)
df_sql1.show()
```

```
[6] SQL Query 1: Top 10 States by Total Cases
+--------------+-----------+------------+------------------+
|         state|total_cases|total_deaths|mortality_rate_pct|
+--------------+-----------+------------+------------------+
|    California|  505068688|     7480314|              1.48|
|         Texas|  402592904|     6841453|               1.7|
|       Florida|  313235465|     5413037|              1.73|
|      New York|  272678258|    12719543|              4.66|
|      Illinois|  195082889|     4340764|              2.23|
|       Georgia|  151749782|     2835421|              1.87|
|          Ohio|  134921770|     2465310|              1.83|
|  Pennsylvania|  133343845|     4195967|              3.15|
|    New Jersey|  128802146|     6004001|              4.66|
|North Carolina|  124271360|     1725065|              1.39|
+--------------+-----------+------------+------------------+
```

In [0]:
```
# SQL Query 2: Monthly trend analysis
print("\n[7] SQL Query 2: Monthly Trend Analysis for 2021")
query2 = """
    SELECT
        month,
        COUNT(DISTINCT state) as num_states,
        SUM(total_cases) as monthly_total_cases,
        AVG(mortality_rate) as avg_mortality_rate,
        MAX(max_daily_cases) as peak_cases
    FROM monthly_stats
    WHERE year = 2021
    GROUP BY month
    ORDER BY month
```

```
"""

df_sql2 = spark.sql(query2)
df_sql2.show()
```

```
[7] SQL Query 2: Monthly Trend Analysis for 2021
+-----+----------+-------------------+------------------+----------+
|month|num_states|monthly_total_cases|avg_mortality_rate|peak_cases|
+-----+----------+-------------------+------------------+----------+
|    1|       110|          742182208|          0.798273|   1117346|
|    2|       110|         1291966232|          0.825818|   1192559|
|    3|       110|          643622995|          0.838000|   1208672|
+-----+----------+-------------------+------------------+----------+
```

OPTIMIZATION STRATEGIES

In [0]:
```
# Show Execution Plan
df_monthly_stats.explain(mode="extended")
```

```
== Parsed Logical Plan ==
Project [state#26658, year#26774, month#26776, total_cases#27428L, total_deaths#27429L, avg_daily_cases#274
30, max_daily_cases#27431, num_records#27432L, cast(((cast(total_deaths#27429L as double) / cast(total_case
s#27428L as double)) * cast(100 as double)) as decimal(10,2)) AS mortality_rate#27439]
+- Aggregate [state#26658, year#26774, month#26776], [state#26658, year#26774, month#26776, sum(cases#2677
0) AS total_cases#27428L, sum(deaths#26772) AS total_deaths#27429L, avg(cases#26770) AS avg_daily_cases#274
30, max(cases#26770) AS max_daily_cases#27431, count(1) AS num_records#27432L]
   +- Project [date#26768, county#26657, state#26658, fips#26659, cases#26770, deaths#26772, year#26774, mo
nth#26776, quarter#26778, cases_per_death#26780, CASE WHEN (cases#26770 > 10000) THEN High ELSE Low END AS
is_high_cases#26782]
      +- Project [date#26768, county#26657, state#26658, fips#26659, cases#26770, deaths#26772, year#26774,
month#26776, quarter#26778, CASE WHEN ((deaths#26772 > 0) AND isnotnull(deaths#26772)) THEN (cast(cases#267
70 as double) / cast(deaths#26772 as double)) ELSE cast(null as double) END AS cases_per_death#26780]
         +- Project [date#26768, county#26657, state#26658, fips#26659, cases#26770, deaths#26772, year#267
74, month#26776, quarter(date#26768) AS quarter#26778]
            +- Project [date#26768, county#26657, state#26658, fips#26659, cases#26770, deaths#26772, year#
26774, month(date#26768) AS month#26776]
               +- Project [date#26768, county#26657, state#26658, fips#26659, cases#26770, deaths#26772, ye
ar(date#26768) AS year#26774]
                  +- Filter (cases#26770 > 0)
                     +- Filter isnotnull(try_to_date(date#26768, Some(yyyy-MM-dd), Some(Etc/UTC), false))
                        +- Project [date#26768, county#26657, state#26658, fips#26659, cases#26770, coalesc
e(try_cast(deaths#26661 as int), 0) AS deaths#26772]
                           +- Project [date#26768, county#26657, state#26658, fips#26659, coalesce(try_cast
(cases#26660 as int), 0) AS cases#26770, deaths#26661]
                              +- Project [to_date(date#26656, Some(yyyy-MM-dd), Some(Etc/UTC), true) AS dat
e#26768, county#26657, state#26658, fips#26659, cases#26660, deaths#26661]
                                 +- Relation [date#26656,county#26657,state#26658,fips#26659,cases#26660,de
aths#26661] csv

== Analyzed Logical Plan ==
state: string, year: int, month: int, total_cases: bigint, total_deaths: bigint, avg_daily_cases: double, m
ax_daily_cases: int, num_records: bigint, mortality_rate: decimal(10,2)
Project [state#26658, year#26774, month#26776, total_cases#27428L, total_deaths#27429L, avg_daily_cases#274
30, max_daily_cases#27431, num_records#27432L, cast(((cast(total_deaths#27429L as double) / cast(total_case
s#27428L as double)) * cast(100 as double)) as decimal(10,2)) AS mortality_rate#27439]
+- Aggregate [state#26658, year#26774, month#26776], [state#26658, year#26774, month#26776, sum(cases#2677
0) AS total_cases#27428L, sum(deaths#26772) AS total_deaths#27429L, avg(cases#26770) AS avg_daily_cases#274
30, max(cases#26770) AS max_daily_cases#27431, count(1) AS num_records#27432L]
   +- Project [date#26768, county#26657, state#26658, fips#26659, cases#26770, deaths#26772, year#26774, mo
nth#26776, quarter#26778, cases_per_death#26780, CASE WHEN (cases#26770 > 10000) THEN High ELSE Low END AS
is_high_cases#26782]
```

```
      +- Project [date#26768, county#26657, state#26658, fips#26659, cases#26770, deaths#26772, year#26774,
month#26776, quarter#26778, CASE WHEN ((deaths#26772 > 0) AND isnotnull(deaths#26772)) THEN (cast(cases#267
70 as double) / cast(deaths#26772 as double)) ELSE cast(null as double) END AS cases_per_death#26780]
         +- Project [date#26768, county#26657, state#26658, fips#26659, cases#26770, deaths#26772, year#267
74, month#26776, quarter(date#26768) AS quarter#26778]
            +- Project [date#26768, county#26657, state#26658, fips#26659, cases#26770, deaths#26772, year#
26774, month(date#26768) AS month#26776]
               +- Project [date#26768, county#26657, state#26658, fips#26659, cases#26770, deaths#26772, ye
ar(date#26768) AS year#26774]
                  +- Filter (cases#26770 > 0)
                     +- Filter isnotnull(try_to_date(date#26768, Some(yyyy-MM-dd), Some(Etc/UTC), false))
                        +- Project [date#26768, county#26657, state#26658, fips#26659, cases#26770, coalesc
e(try_cast(deaths#26661 as int), 0) AS deaths#26772]
                           +- Project [date#26768, county#26657, state#26658, fips#26659, coalesce(try_cast
(cases#26660 as int), 0) AS cases#26770, deaths#26661]
                              +- Project [to_date(date#26656, Some(yyyy-MM-dd), Some(Etc/UTC), true) AS dat
e#26768, county#26657, state#26658, fips#26659, cases#26660, deaths#26661]
                                 +- Relation [date#26656,county#26657,state#26658,fips#26659,cases#26660,de
aths#26661] csv

== Optimized Logical Plan ==
Project [state#26658, year#26774, month#26776, total_cases#27428L, total_deaths#27429L, avg_daily_cases#274
30, max_daily_cases#27431, num_records#27432L, cast(((cast(total_deaths#27429L as double) / cast(total_case
s#27428L as double)) * 100.0) as decimal(10,2)) AS mortality_rate#27439]
+- Aggregate [state#26658, year#26774, month#26776], [state#26658, year#26774, month#26776, sum(cases#2677
0) AS total_cases#27428L, sum(deaths#26772) AS total_deaths#27429L, (sum(cast(cases#26770 as double)) / cas
t(count(1) as double)) AS avg_daily_cases#27430, max(cases#26770) AS max_daily_cases#27431, count(1) AS num
_records#27432L]
   +- Project [state#26658, cases#26770, deaths#26772, year(date#26768) AS year#26774, month(date#26768) AS
month#26776]
      +- Project [cast(gettimestamp(date#26656, yyyy-MM-dd, TimestampType, try_to_date, Some(Etc/UTC), tru
e) as date) AS date#26768, state#26658, coalesce(try_cast(cases#26660 as int), 0) AS cases#26770, coalesce
(try_cast(deaths#26661 as int), 0) AS deaths#26772]
         +- Filter ((coalesce(try_cast(cases#26660 as int), 0) > 0) AND isnotnull(cast(gettimestamp(cast(ge
ttimestamp(date#26656, yyyy-MM-dd, TimestampType, try_to_date, Some(Etc/UTC), true) as date), yyyy-MM-dd, T
imestampType, try_to_date, Some(Etc/UTC), false) as date)))
            +- Relation [date#26656,county#26657,state#26658,fips#26659,cases#26660,deaths#26661] csv

== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- == Initial Plan ==
   ColumnarToRow
```

```
   +- PhotonResultStage
      +- PhotonProject [state#26658, year#26774, month#26776, total_cases#27428L, total_deaths#27429L, avg_
daily_cases#27430, max_daily_cases#27431, num_records#27432L, cast(((cast(total_deaths#27429L as double) /
cast(total_cases#27428L as double)) * 100.0) as decimal(10,2)) AS mortality_rate#27439]
         +- PhotonGroupingAgg(limit=None, keys=[state#26658, year#26774, month#26776], functions=[finalmerg
e_sum(merge sum#27460L) AS sum(cases)#27434L, finalmerge_sum(merge sum#27699L) AS sum(deaths)#27435L, final
merge_sum(merge sum#28104) AS sum(cases)#28101, finalmerge_count(merge count#28106L) AS count(1)#28102L, fi
nalmerge_max(merge max#27701) AS max(cases)#27437], output=[state#26658, year#26774, month#26776, total_cas
es#27428L, total_deaths#27429L, avg_daily_cases#27430, max_daily_cases#27431, num_records#27432L])
            +- PhotonShuffleExchangeSource
               +- PhotonShuffleMapStage ENSURE_REQUIREMENTS, [id=#29532]
                  +- PhotonShuffleExchangeSink hashpartitioning(state#26658, year#26774, month#26776, 1024)
                     +- PhotonGroupingAgg(limit=None, keys=[state#26658, year#26774, month#26776], function
s=[partial_sum(cases#26770) AS sum#27460L, partial_sum(deaths#26772) AS sum#27699L, partial_sum(cast(cases#
26770 as double)) AS sum#28104, partial_count(1) AS count#28106L, partial_max(cases#26770) AS max#27701], o
utput=[state#26658, year#26774, month#26776, sum#27460L, sum#27699L, sum#28104, count#28106L, max#27701])
                        +- PhotonProject [state#26658, cases#26770, deaths#26772, year(date#26768) AS year#
26774, month(date#26768) AS month#26776]
                           +- PhotonProject [cast(gettimestamp(date#26656, yyyy-MM-dd, TimestampType, try_t
o_date, Some(Etc/UTC), true) as date) AS date#26768, state#26658, coalesce(try_cast(cases#26660 as int), 0)
AS cases#26770, coalesce(try_cast(deaths#26661 as int), 0) AS deaths#26772]
                              +- PhotonFilter ((coalesce(try_cast(cases#26660 as int), 0) > 0) AND isnotnul
l(cast(gettimestamp(cast(gettimestamp(date#26656, yyyy-MM-dd, TimestampType, try_to_date, Some(Etc/UTC), tr
ue) as date), yyyy-MM-dd, TimestampType, try_to_date, Some(Etc/UTC), false) as date)))
                                 +- PhotonRowToColumnar
                                    +- FileScan csv [date#26656,state#26658,cases#26660,deaths#26661] Batch
ed: false, DataFilters: [(coalesce(try_cast(cases#26660 as int), 0) > 0), isnotnull(cast(gettimestamp(cast
(gettimestamp(d..., Format: CSV, Location: InMemoryFileIndex(1 paths)[dbfs:/databricks-datasets/COVID/covid
-19-data], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<date:string,state:string,cases:strin
g,deaths:string>

== Photon Explanation ==
The query is fully supported by Photon.
```

In [0]:
```
# Demonstrate Filter Pushdown
print("\n[9] Demonstrating Filter Pushdown:")
optimized_query = df_transformed \
    .filter(col("year") == 2021) \
    .filter(col("state") == "California") \
    .select("date", "state", "cases", "deaths")
```

```
print("\nOptimized Query Plan (filters pushed down):")
optimized_query.explain(mode="formatted")
```

```
[9] Demonstrating Filter Pushdown:

Optimized Query Plan (filters pushed down):
== Physical Plan ==
* ColumnarToRow (6)
+- PhotonResultStage (5)
   +- PhotonProject (4)
      +- PhotonFilter (3)
         +- PhotonRowToColumnar (2)
            +- Scan csv  (1)


(1) Scan csv
Output [4]: [date#26656, state#26658, cases#26660, deaths#26661]
Batched: false
Location: InMemoryFileIndex [dbfs:/databricks-datasets/COVID/covid-19-data]
PushedFilters: [IsNotNull(state), EqualTo(state,California)]
ReadSchema: struct<date:string,state:string,cases:string,deaths:string>

(2) PhotonRowToColumnar
Input [4]: [date#26656, state#26658, cases#26660, deaths#26661]

(3) PhotonFilter
Input [4]: [date#26656, state#26658, cases#26660, deaths#26661]
Arguments: ((((isnotnull(state#26658) AND (state#26658 = California)) AND (coalesce(try_cast(cases#26660 as
int), 0) > 0)) AND (year(cast(gettimestamp(date#26656, yyyy-MM-dd, TimestampType, try_to_date, Some(Etc/UT
C), true) as date)) = 2021)) AND isnotnull(cast(gettimestamp(cast(gettimestamp(date#26656, yyyy-MM-dd, Time
stampType, try_to_date, Some(Etc/UTC), true) as date), yyyy-MM-dd, TimestampType, try_to_date, Some(Etc/UT
C), false) as date)))

(4) PhotonProject
Input [4]: [date#26656, state#26658, cases#26660, deaths#26661]
Arguments: [cast(gettimestamp(date#26656, yyyy-MM-dd, TimestampType, try_to_date, Some(Etc/UTC), true) as d
ate) AS date#26768, state#26658, coalesce(try_cast(cases#26660 as int), 0) AS cases#26770, coalesce(try_cas
t(deaths#26661 as int), 0) AS deaths#26772]

(5) PhotonResultStage
Input [4]: [date#26768, state#26658, cases#26770, deaths#26772]

(6) ColumnarToRow [codegen id : 1]
Input [4]: [date#26768, state#26658, cases#26770, deaths#26772]
```

```
== Photon Explanation ==
The query is fully supported by Photon.
```

Demonstrating Transformations (Lazy) vs Actions (Eager)

In [0]:
```python
# Transformations (Lazy - no execution yet)
print("\nApplying TRANSFORMATIONS (Lazy - not executed yet):")
lazy_df = df_transformed \
    .filter(col("year") == 2021) \
    .select("state", "date", "cases") \
    .filter(col("cases") > 5000)

print("Transformations defined (no computation performed)")
print(f"DataFrame type: {type(lazy_df)}")
```

```
Applying TRANSFORMATIONS (Lazy - not executed yet):
Transformations defined (no computation performed)
DataFrame type: <class 'pyspark.sql.connect.dataframe.DataFrame'>
```

In [0]:
```python
# Actions (Eager - triggers execution)
print("\nExecuting ACTIONS (Eager - triggers computation):")

print("\nAction 1: count()")
start = time.time()
count = lazy_df.count()
action_time1 = time.time() - start
print(f"Count: {count:,} (executed in {action_time1:.3f}s)")

print("\nAction 2: show()")
start = time.time()
lazy_df.show(5)
action_time2 = time.time() - start
print(f"Show completed (executed in {action_time2:.3f}s)")

print("\nAction 3: collect()")
start = time.time()
collected = lazy_df.limit(100).collect()
action_time3 = time.time() - start
print(f"Collected {len(collected)} rows (executed in {action_time3:.3f}s)")
```

```
Executing ACTIONS (Eager - triggers computation):

Action 1: count()
Count: 85,962 (executed in 0.745s)

Action 2: show()
+-------+----------+-----+
|  state|      date|cases|
+-------+----------+-----+
|Alabama|2021-01-01|13823|
|Alabama|2021-01-01| 9584|
|Alabama|2021-01-01| 7194|
|Alabama|2021-01-01| 6978|
|Alabama|2021-01-01| 6542|
+-------+----------+-----+
only showing top 5 rows
Show completed (executed in 0.679s)

Action 3: collect()
Collected 100 rows (executed in 0.493s)
```

In [0]:
```
print("\nDefining multiple transformations (chained operations):")
start_lazy = time.time()

# Chain multiple transformations - NO execution happens yet!
lazy_df = df_transformed \
    .filter(col("year") == 2021) \
    .filter(col("state").isNotNull()) \
    .select("state", "date", "cases", "deaths") \
    .filter(col("cases") > 5000) \
    .withColumn("high_mortality", col("deaths") > 100) \
    .filter(col("high_mortality") == True)

lazy_time = time.time() - start_lazy

print(f"6 transformations defined in {lazy_time*1000:.2f} milliseconds")
print(f"DataFrame object created: {type(lazy_df)}")
```

```
Defining multiple transformations (chained operations):
6 transformations defined in 0.67 milliseconds
DataFrame object created: <class 'pyspark.sql.connect.dataframe.DataFrame'>
```

In [0]:
```
# Show the optimized execution plan
print("Execution Plan (Spark's optimization):")
lazy_df.explain(mode="simple")
```

```
Execution Plan (Spark's optimization):
== Physical Plan ==
*(1) ColumnarToRow
+- PhotonResultStage
   +- PhotonProject [state#26658, date#26768, cases#26770, deaths#26772, (deaths#26772 > 100) AS high_morta
lity#28192]
      +- PhotonProject [state#26658, cast(gettimestamp(date#26656, yyyy-MM-dd, TimestampType, try_to_date,
Some(Etc/UTC), true) as date) AS date#26768, coalesce(try_cast(cases#26660 as int), 0) AS cases#26770, coal
esce(try_cast(deaths#26661 as int), 0) AS deaths#26772]
         +- PhotonFilter (((((isnotnull(state#26658) AND (coalesce(try_cast(cases#26660 as int), 0) > 0)) A
ND (coalesce(try_cast(cases#26660 as int), 0) > 5000)) AND (coalesce(try_cast(deaths#26661 as int), 0) > 10
0)) AND (year(cast(gettimestamp(date#26656, yyyy-MM-dd, TimestampType, try_to_date, Some(Etc/UTC), true) as
date)) = 2021)) AND isnotnull(cast(gettimestamp(cast(gettimestamp(date#26656, yyyy-MM-dd, TimestampType, tr
y_to_date, Some(Etc/UTC), true) as date), yyyy-MM-dd, TimestampType, try_to_date, Some(Etc/UTC), false) as
date)))
            +- PhotonRowToColumnar
               +- FileScan csv [date#26656,state#26658,cases#26660,deaths#26661] Batched: false, DataFilter
s: [isnotnull(state#26658), (coalesce(try_cast(cases#26660 as int), 0) > 0), (coalesce(try_cast(case..., Fo
rmat: CSV, Location: InMemoryFileIndex(1 paths)[dbfs:/databricks-datasets/COVID/covid-19-data], PartitionFi
lters: [], PushedFilters: [IsNotNull(state)], ReadSchema: struct<date:string,state:string,cases:string,deat
hs:string>


== Photon Explanation ==
The query is fully supported by Photon.
```

In [0]:
```
print("\nExecuting ACTIONS (Eager - triggers computation):")

# ACTION 1: count()
print("ACTION 1: count() - counts all rows")
start = time.time()
count = lazy_df.count()
action_time1 = time.time() - start
print(f"Result: {count:,} rows")
print(f"Execution time: {action_time1:.3f} seconds")
print(f"Now data was actually processed!\n")

# ACTION 2: show()
```

```
print("ACTION 2: show() - displays sample rows")
start = time.time()
lazy_df.show(5)
action_time2 = time.time() - start
print(f"Execution time: {action_time2:.3f} seconds")
print(f"Note: Slightly faster because Spark only needs 5 rows\n")

# ACTION 3: collect()
print("ACTION 3: collect() - brings data to driver")
start = time.time()
collected = lazy_df.limit(10).collect()
action_time3 = time.time() - start
print(f"Collected {len(collected)} rows to driver memory")
print(f"Execution time: {action_time3:.3f} seconds")
```

```
Executing ACTIONS (Eager - triggers computation):
ACTION 1: count() - counts all rows
Result: 66,791 rows
Execution time: 0.981 seconds
Now data was actually processed!

ACTION 2: show() - displays sample rows
+-------+----------+-----+------+--------------+
|  state|      date|cases|deaths|high_mortality|
+-------+----------+-----+------+--------------+
|Alabama|2021-01-01|13823|   169|          true|
|Alabama|2021-01-01| 9584|   157|          true|
|Alabama|2021-01-01|53058|   719|          true|
|Alabama|2021-01-01|22590|   179|          true|
|Alabama|2021-01-01|26151|   508|          true|
+-------+----------+-----+------+--------------+
only showing top 5 rows
Execution time: 0.832 seconds
Note: Slightly faster because Spark only needs 5 rows

ACTION 3: collect() - brings data to driver
Collected 10 rows to driver memory
Execution time: 0.478 seconds
```

In [0]:
```
#Compare the performance Lazy vs Eager:

print("\nPerformance Comparison:\n")
```

```
print(f"{'Operation':<40} {'Time':<15} {'Data Processed'}")
print("-" * 70)
print(f"{'Define all transformations (LAZY)':<40} {lazy_time*1000:>6.2f} ms      {'None (0 bytes)'}")
print(f"{'Execute count() action':<40} {action_time1*1000:>6.0f} ms      {f'{count:,} rows'}")
print(f"{'Execute show() action':<40} {action_time2*1000:>6.0f} ms      {'5 rows (optimized)'}")
print(f"{'Execute collect() action':<40} {action_time3*1000:>6.0f} ms      {'10 rows'}")
```

```
Performance Comparison:

Operation                                Time            Data Processed
----------------------------------------------------------------------
Define all transformations (LAZY)          0.67 ms      None (0 bytes)
Execute count() action                      981 ms      66,791 rows
Execute show() action                       832 ms      5 rows (optimized)
Execute collect() action                    478 ms      10 rows
```

Write results

In [0]:
```
# Use a Unity Catalog volume (recommended) to save the parquet files:
output_base = "/Volumes/de/de/de/covid_pipeline_output"

output_path_parquet = f"{output_base}/monthly_stats_parquet"
df_monthly_stats.write.mode("overwrite").parquet(output_path_parquet)

output_path_partitioned = f"{output_base}/yearly_stats_partitioned"
df_yearly.write.mode("overwrite").partitionBy("year").parquet(output_path_partitioned)

output_path_delta = f"{output_base}/top_states_delta"
df_sql1.write.format("delta").mode("overwrite").save(output_path_delta)
```

In [0]:
```
print(f"""
Pipeline Execution Complete!

Data Processed:
- Original Records: {df_covid.count():,}
- Filtered Records: {df_filtered.count():,}
- Monthly Aggregations: {df_monthly_stats.count():,}
- Yearly Aggregations: {df_yearly.count():,}

Optimizations Applied:
✓ Early filtering to reduce data volume
```

```
✓ Column pruning (selecting only needed columns)
✓ Predicate pushdown demonstrated
✓ Partitioned writes for efficient storage
✓ Serverless compute automatic optimizations

Output Locations:
- {output_path_parquet}
- {output_path_partitioned}
- {output_path_delta}
""")
```

```
Pipeline Execution Complete!

Data Processed:
- Original Records: 1,227,256
- Filtered Records: 1,224,611
- Monthly Aggregations: 1,442
- Yearly Aggregations: 220

Optimizations Applied:
✓ Early filtering to reduce data volume
✓ Column pruning (selecting only needed columns)
✓ Predicate pushdown demonstrated
✓ Partitioned writes for efficient storage
✓ Serverless compute automatic optimizations

Output Locations:
- /Volumes/de/de/de/covid_pipeline_output/monthly_stats_parquet
- /Volumes/de/de/de/covid_pipeline_output/yearly_stats_partitioned
- /Volumes/de/de/de/covid_pipeline_output/top_states_delta
```