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

spark = SparkSession.builder.appName("MusicListeningDataPipeline").getOrCreate()


print("✅ Spark session created successfully.")

# Initialize Spark Session with optimized configurations
spark = (
    SparkSession.builder.appName("COVID-19 Data Pipeline")
    .config("spark.sql.adaptive.enabled", "true")
    .config("spark.sql.adaptive.coalescePartitions.enabled", "true")
    .getOrCreate()
)

print("=" * 80)
print("COVID-19 DATA PROCESSING PIPELINE")
print("=" * 80)
```

```
✅ Spark session created successfully.
================================================================================
COVID-19 DATA PROCESSING PIPELINE
================================================================================
```

In [0]:
```python
# ================================================================================
# LOAD DATA WITH DEFAULT PARTITIONING
# ================================================================================

# Start timing
start_time = time.time()

# Read the dataset
covid_df = spark.read.csv(
    "/databricks-datasets/COVID/covid-19-data/", header=True, inferSchema=True
)

# Trigger action to actually load the data (transformations are lazy!)
row_count = covid_df.count()

# Calculate elapsed time
load_time = time.time() - start_time

# Check the data
print(f"Total rows: {covid_df.count():,}")
print(f"⏱️  Time to load data: {load_time:.5f} seconds")
covid_df.printSchema()
covid_df.show(5)
```

```
Total rows: 1,227,256
⏱️  Time to load data: 1.78491 seconds
root
 |-- date: string (nullable = true)
 |-- county: string (nullable = true)
 |-- state: string (nullable = true)
 |-- fips: string (nullable = true)
 |-- cases: string (nullable = true)
 |-- deaths: string (nullable = true)

+----------+---------+----------+-----+-----+------+
|      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
```

In [0]:

```python
# ================================================================================
# LOAD DATA WITH SPECIFIED NUMBER OF PARTITIONS
# ================================================================================
# Start timing
start_time = time.time()

# Read the dataset with specified number of partitions
covid_df = spark.read.csv(
    "/databricks-datasets/COVID/covid-19-data/", header=True, inferSchema=True
).repartition(
    8
)  # Define number of partitions (adjust based on cluster size)

# Trigger action to actually load the data (transformations are lazy!)
row_count = covid_df.count()

# Calculate elapsed time
load_time = time.time() - start_time

# Check the data
print(f"Total rows: {row_count:,}")
# print(f"Number of partitions: {covid_df.rdd.getNumPartitions()}")
print(f"⏱️  Time to load data: {load_time:.5f} seconds")
covid_df.printSchema()
covid_df.show(5)
```

```
Total rows: 1,227,256
⏱️  Time to load data: 2.20983 seconds
root
 |-- date: string (nullable = true)
 |-- county: string (nullable = true)
 |-- state: string (nullable = true)
 |-- fips: string (nullable = true)
 |-- cases: string (nullable = true)
 |-- deaths: string (nullable = true)

+----------+-----------+----------+-----+-----+------+
|      date|     county|     state| fips|cases|deaths|
+----------+-----------+----------+-----+-----+------+
|2020-01-25|     Orange|California|06059|    1|     0|
|2020-01-27|Los Angeles|California|06037|    1|     0|
|2020-01-28|  Snohomish|Washington|53061|    1|     0|
|2020-01-30|   Maricopa|   Arizona|04013|    1|     0|
|2020-01-31|  Snohomish|Washington|53061|    1|     0|
+----------+-----------+----------+-----+-----+------+
only showing top 5 rows
```

In [0]:
```python
# Checkthe size of the dataset
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]:
```python
# ============================================================================
# CREATE REFERENCE DATA FOR JOIN
# ============================================================================
print("\n" + "=" * 80)
print("CREATING REFERENCE DATA")
print("=" * 80)

# Create a second dataset for join operation (simulate external data)
# This represents state population data
state_population = spark.createDataFrame(
    [
        ("California", 39538223, "West"),
```

```
        ("New York", 20201249, "Northeast"),
        ("Texas", 29145505, "South"),
        ("Florida", 21538187, "South"),
        ("Washington", 7705281, "West"),
        ("Illinois", 12812508, "Midwest"),
    ],
    ["state", "population", "region"],
)

print(f"State population reference data loaded: {state_population.count()} state
state_population.show()
```

```
================================================================================
CREATING REFERENCE DATA
================================================================================
State population reference data loaded: 6 states
+----------+----------+---------+
|     state|population|   region|
+----------+----------+---------+
|California|  39538223|     West|
|  New York|  20201249|Northeast|
|     Texas|  29145505|    South|
|   Florida|  21538187|    South|
|Washington|   7705281|     West|
|  Illinois|  12812508|  Midwest|
+----------+----------+---------+
```

In [0]:

```
# ================================================================================
# VERSION 1: EAGER/INEFFICIENT APPROACH ❌
# ================================================================================
print("\n" + "=" * 80)
print("VERSION 1: EAGER/INEFFICIENT APPROACH (Bad Practices)")
print("=" * 80)
print("Problems:")
print("  ❌ GroupBy BEFORE filtering (processes all 1.2M rows)")
print("  ❌ Join on full dataset BEFORE filtering")
print("  ❌ Column transformations on large dataset")
print("  ❌ Multiple shuffles due to poor ordering")
print("  ❌ No early data reduction")

eager_start = time.time()

# Step 1: GroupBy FIRST (BAD - processes all data)
print("\n[Step 1] Performing groupBy on ENTIRE dataset...")
step1_start = time.time()
eager_grouped = covid_df.groupBy("state", "county", "date").agg(
    sum("cases").alias("total_cases"), sum("deaths").alias("total_deaths")
)
step1_time = time.time() - step1_start
print(f"  Time: {step1_time:.2f}s | Rows: {eager_grouped.count():,}")

# Step 2: Add column transformations BEFORE filtering (BAD - transforms all rows
print("\n[Step 2] Adding column transformations on full dataset...")
step2_start = time.time()
eager_transformed = (
    eager_grouped.withColumn(
        "mortality_rate",
        when(
            col("total_cases") > 0, (col("total_deaths") / col("total_cases")) *
```

```
        ).otherwise(0),
    )
    .withColumn("year", year(col("date")))
    .withColumn("month", month(col("date")))
    .withColumn(
        "case_category",
        when(col("total_cases") < 100, "Low")
        .when(col("total_cases") < 1000, "Medium")
        .otherwise("High"),
    )
)
step2_time = time.time() - step2_start
print(f"  Time: {step2_time:.2f}s")

# Step 3: Join BEFORE filtering (BAD - joins large dataset)
print("\n[Step 3] Joining with population data on full dataset...")
step3_start = time.time()
eager_joined = eager_transformed.join(state_population, on="state", how="left")
step3_time = time.time() - step3_start
print(f"  Time: {step3_time:.2f}s | Rows: {eager_joined.count():,}")

# Step 4: Filter LAST (BAD - after all expensive operations)
print("\n[Step 4] Finally filtering (after all expensive operations)...")
step4_start = time.time()
eager_filtered = eager_joined.filter(
    (col("year") == 2020)
    & (col("state").isin("California", "New York", "Texas", "Florida", "Washingt
)
step4_time = time.time() - step4_start
print(f"  Time: {step4_time:.2f}s | Rows: {eager_filtered.count():,}")

# Step 5: Another filter (BAD - separate operation causes another pass)
print("\n[Step 5] Second filter (separate operation)...")
step5_start = time.time()
eager_final = eager_filtered.filter(col("total_cases") >= 10)
step5_time = time.time() - step5_start
print(f"  Time: {step5_time:.2f}s | Rows: {eager_final.count():,}")

# Step 6: Final aggregation with another groupBy (causes another shuffle)
print("\n[Step 6] Final aggregation (another shuffle)...")
step6_start = time.time()
eager_result = (
    eager_final.groupBy("state", "region", "year", "month")
    .agg(
        sum("total_cases").alias("monthly_cases"),
        sum("total_deaths").alias("monthly_deaths"),
        avg("mortality_rate").alias("avg_mortality_rate"),
        count("*").alias("counties_reported"),
    )
    .orderBy("state", "year", "month")
)

# Trigger execution
eager_count = eager_result.count()
step6_time = time.time() - step6_start
print(f"  Time: {step6_time:.2f}s | Final rows: {eager_count:,}")

eager_total = time.time() - eager_start

print("\n" + "-" * 80)
```

```
print("EAGER APPROACH SUMMARY:")
print(f"  Total execution time: {eager_total:.2f} seconds")
print(f"  Number of shuffles: ~4-5 (groupBy, join, filters, final groupBy)")
print(f"  Data processed: Full dataset multiple times")
print("-" * 80)

# Show sample results
print("\nSample results (Eager approach):")
eager_result.show(10, truncate=False)

# Show execution plan
print("\nEXECUTION PLAN (Eager - Notice multiple stages and shuffles):")
eager_result.explain(mode="simple")
```

```
================================================================================
VERSION 1: EAGER/INEFFICIENT APPROACH (Bad Practices)
================================================================================
Problems:
  ❌ GroupBy BEFORE filtering (processes all 1.2M rows)
  ❌ Join on full dataset BEFORE filtering
  ❌ Column transformations on large dataset
  ❌ Multiple shuffles due to poor ordering
  ❌ No early data reduction

[Step 1] Performing groupBy on ENTIRE dataset...
  Time: 0.00s | Rows: 1,133,113

[Step 2] Adding column transformations on full dataset...
  Time: 0.00s

[Step 3] Joining with population data on full dataset...
  Time: 0.00s | Rows: 1,133,113

[Step 4] Finally filtering (after all expensive operations)...
  Time: 0.00s | Rows: 131,570

[Step 5] Second filter (separate operation)...
  Time: 0.00s | Rows: 115,697

[Step 6] Final aggregation (another shuffle)...
  Time: 0.81s | Final rows: 52

--------------------------------------------------------------------------------
EAGER APPROACH SUMMARY:
  Total execution time: 4.29 seconds
  Number of shuffles: ~4-5 (groupBy, join, filters, final groupBy)
  Data processed: Full dataset multiple times
--------------------------------------------------------------------------------

Sample results (Eager approach):
+----------+------+----+-----+------------+--------------+------------------+---
--------------+
|state     |region|year|month|monthly_cases|monthly_deaths|avg_mortality_rate|cou
nties_reported|
+----------+------+----+-----+------------+--------------+------------------+---
--------------+
|California|West  |2020|2    |44.0        |0.0           |0.0               |4
|
|California|West  |2020|3    |53893.0     |1075.0        |1.953142131176942 |468
|
|California|West  |2020|4    |864363.0    |30166.0       |3.2309658387784115|123
4             |
|California|West  |2020|5    |2475042.0   |99347.0       |3.0996459026429473|141
1             |
|California|West  |2020|6    |4904141.0   |155680.0      |2.24890510173643  |150
2             |
|California|West  |2020|7    |1.140477E7  |232621.0      |1.2139232466490364|167
1             |
|California|West  |2020|8    |1.9168191E7 |349458.0      |1.3111670072947814|168
3             |
|California|West  |2020|9    |2.3157578E7 |438311.0      |1.4751735752790123|168
0             |
|California|West  |2020|10   |2.7163707E7 |522654.0      |1.5523338388611116|173
6             |
```

```
|California|West  |2020|11   |3.1809052E7  |551791.0      |1.3805990606541387|172
0             |
+----------+------+----+-----+-------------+--------------+------------------+---
--------------+
only showing top 10 rows

EXECUTION PLAN (Eager - Notice multiple stages and shuffles):
== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- == Initial Plan ==
   ColumnarToRow
   +- PhotonResultStage
      +- PhotonSort [state#14466 ASC NULLS FIRST, year#15344 ASC NULLS FIRST, mon
th#15346 ASC NULLS FIRST]
         +- PhotonShuffleExchangeSource
            +- PhotonShuffleMapStage ENSURE_REQUIREMENTS, [id=#19072]
               +- PhotonShuffleExchangeSink rangepartitioning(state#14466 ASC NUL
LS FIRST, year#15344 ASC NULLS FIRST, month#15346 ASC NULLS FIRST, 1024)
                  +- PhotonProject [state#14466, region#15355, year#15344, month#
15346, sum(total_cases)#15357 AS monthly_cases#15349, sum(total_deaths)#15358 AS
monthly_deaths#15350, avg(mortality_rate)#15359 AS avg_mortality_rate#15351, coun
t(1)#15356L AS counties_reported#15352L]
                     +- PhotonGroupingAgg(keys=[state#14466, region#15355, year#1
5344, month#15346], functions=[finalmerge_sum(merge sum#15361) AS sum(total_case
s)#15357, finalmerge_sum(merge sum#15363) AS sum(total_deaths)#15358, finalmerge_
avg(merge sum#15366, count#15367L) AS avg(mortality_rate)#15359, finalmerge_count
(merge count#15369L) AS count(1)#15356L])
                        +- PhotonShuffleExchangeSource
                           +- PhotonShuffleMapStage ENSURE_REQUIREMENTS, [id=#190
64]
                              +- PhotonShuffleExchangeSink hashpartitioning(state
#14466, region#15355, year#15344, month#15346, 1024)
                                 +- PhotonGroupingAgg(keys=[state#14466, region#1
5355, year#15344, month#15346], functions=[merge_sum(merge sum#15361) AS sum#1536
1, merge_sum(merge sum#15363) AS sum#15363, merge_avg(merge sum#15366, count#1536
7L) AS (sum#15366, count#15367L), merge_count(merge count#15369L) AS count#15369
L])
                                    +- PhotonProject [state#14466, region#15355,
year#15344, month#15346, sum#15361, sum#15363, sum#15366, count#15367L, count#153
69L]
                                       +- PhotonBroadcastHashJoin [state#14466],
[state#15353], LeftOuter, BuildRight, false, true
                                          :- PhotonGroupingAgg(keys=[state#14466,
year#15344, month#15346], functions=[partial_sum(total_cases#15337) AS sum#15361,
partial_sum(total_deaths#15338) AS sum#15363, partial_avg(mortality_rate#15342) A
S (sum#15366, count#15367L), partial_count(1) AS count#15369L])
                                          :  +- PhotonProject [state#14466, total
_cases#15337, total_deaths#15338, CASE WHEN (total_cases#15337 > 0.0) THEN ((tota
l_deaths#15338 / total_cases#15337) * 100.0) ELSE 0.0 END AS mortality_rate#1534
2, year(cast(date#14464 as date)) AS year#15344, month(cast(date#14464 as date))
AS month#15346]
                                          :     +- PhotonFilter (isnotnull(total_
cases#15337) AND (total_cases#15337 >= 10.0))
                                          :        +- PhotonGroupingAgg(keys=[sta
te#14466, county#14465, date#14464], functions=[finalmerge_sum(merge sum#15373) A
S sum(cases)#15339, finalmerge_sum(merge sum#15375) AS sum(deaths)#15340])
                                          :           +- PhotonShuffleExchangeSou
rce
                                          :              +- PhotonShuffleMapStage
ENSURE_REQUIREMENTS, [id=#19039]
```

```
                                                  :                  +- PhotonShuffleExcha
ngeSink hashpartitioning(state#14466, county#14465, date#14464, 1024)
                                                  :                     +- PhotonGroupingA
gg(keys=[state#14466, county#14465, date#14464], functions=[partial_sum(cast(case
s#14468 as double)) AS sum#15373, partial_sum(cast(deaths#14469 as double)) AS su
m#15375])
                                                  :                        +- PhotonFilter
((isnotnull(date#14464) AND (year(cast(date#14464 as date)) = 2020)) AND state#14
466 IN (California,New York,Texas,Florida,Washington))
                                                  :                           +- PhotonRow
ToColumnar
                                                  :                              +- FileSc
an csv [date#14464,county#14465,state#14466,cases#14468,deaths#14469] Batched: fa
lse, DataFilters: [isnotnull(date#14464), (year(cast(date#14464 as date)) = 202
0), state#14466 IN (California,New Y..., Format: CSV, Location: InMemoryFileIndex
(1 paths)[dbfs:/databricks-datasets/COVID/covid-19-data], PartitionFilters: [], P
ushedFilters: [IsNotNull(date), In(state, [California,Florida,New York,Texas,Wash
ington])], ReadSchema: struct<date:string,county:string,state:string,cases:strin
g,deaths:string>
                                               +- PhotonShuffleExchangeSource
                                                  +- PhotonShuffleMapStage EXECUTOR_BR
OADCAST, [id=#19055]
                                                     +- PhotonShuffleExchangeSink Sing
lePartition
                                                        +- PhotonFilter (isnotnull(sta
te#15353) AND state#15353 IN (California,New York,Texas,Florida,Washington))
                                                           +- PhotonRowToColumnar
                                                              +- LocalTableScan [state
#15353, region#15355]


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

In [0]:

```python
# ============================================================================
# VERSION 2: LAZY/OPTIMIZED APPROACH ✅
# ============================================================================
print("\n" + "=" * 80)
print("VERSION 2: LAZY/OPTIMIZED APPROACH (Best Practices)")
print("=" * 80)
print("Optimizations:")
print("  ✅ Filter EARLY (reduce data volume immediately)")
print("  ✅ Combine filters together (single pass)")
print("  ✅ Join AFTER filtering (smaller dataset)")
print("  ✅ Minimize shuffles through intelligent ordering")
print("  ✅ Use appropriate partitioning")

lazy_start = time.time()

# Step 1: Filter FIRST (GOOD - reduce data early)
print("\n[Step 1] Filtering data FIRST (early reduction)...")
step1_start = time.time()
lazy_filtered = covid_df.filter(
    (year(col("date")) == 2020)
    & (col("state").isin("California", "New York", "Texas", "Florida", "Washingt
    & (col("cases") >= 10)  # Combine multiple filters
)
# Repartition after filtering for optimal parallelism
lazy_filtered = lazy_filtered.repartition(4, "state")
step1_time = time.time() - step1_start
```

```
print(f"  Time: {step1_time:.2f}s | Rows after filter: {lazy_filtered.count():,}

# Step 2: Column transformations on FILTERED data (GOOD - fewer rows)
print("\n[Step 2] Adding column transformations on filtered data...")
step2_start = time.time()
lazy_transformed = (
    lazy_filtered.withColumn("cases_int", col("cases").cast("integer"))
    .withColumn("deaths_int", col("deaths").cast("integer"))
    .withColumn(
        "mortality_rate",
        when(
            col("cases_int") > 0, (col("deaths_int") / col("cases_int")) * 100
        ).otherwise(0),
    )
    .withColumn("year", year(col("date")))
    .withColumn("month", month(col("date")))
    .withColumn(
        "case_category",
        when(col("cases_int") < 100, "Low")
        .when(col("cases_int") < 1000, "Medium")
        .otherwise("High"),
    )
)
step2_time = time.time() - step2_start
print(f"  Time: {step2_time:.2f}s")

# Step 3: GroupBy on FILTERED data (GOOD - much less data to shuffle)
print("\n[Step 3] GroupBy on filtered dataset...")
step3_start = time.time()
lazy_grouped = lazy_transformed.groupBy("state", "county", "date", "year", "mont
    sum("cases_int").alias("total_cases"),
    sum("deaths_int").alias("total_deaths"),
    avg("mortality_rate").alias("avg_mortality_rate"),
    first("case_category").alias("case_category"),
)
step3_time = time.time() - step3_start
print(f"  Time: {step3_time:.2f}s | Rows: {lazy_grouped.count():,}")

# Step 4: Join on SMALL dataset (GOOD - broadcast join possible)
print("\n[Step 4] Joining with population data (small dataset)...")
step4_start = time.time()
# Broadcast the small dimension table
lazy_joined = lazy_grouped.join(broadcast(state_population), on="state", how="le
step4_time = time.time() - step4_start
print(f"  Time: {step4_time:.2f}s")

# Step 5: Final aggregation (GOOD - single optimized operation)
print("\n[Step 5] Final aggregation...")
step5_start = time.time()
lazy_result = (
    lazy_joined.groupBy("state", "region", "year", "month", "population")
    .agg(
        sum("total_cases").alias("monthly_cases"),
        sum("total_deaths").alias("monthly_deaths"),
        avg("avg_mortality_rate").alias("avg_mortality_rate"),
        count("*").alias("counties_reported"),
    )
    .withColumn("cases_per_100k", (col("monthly_cases") / col("population")) * 1
    .orderBy("state", "year", "month")
)
```

```python
# Trigger execution
lazy_count = lazy_result.count()
step5_time = time.time() - step5_start
print(f"  Time: {step5_time:.2f}s | Final rows: {lazy_count:,}")

lazy_total = time.time() - lazy_start

print("\n" + "-" * 80)
print("LAZY/OPTIMIZED APPROACH SUMMARY:")
print(f"  Total execution time: {lazy_total:.2f} seconds")
print(f"  Number of shuffles: ~2 (one groupBy, one final aggregation)")
print(f"  Data processed: Filtered dataset only")
print("-" * 80)

# Show sample results
print("\nSample results (Optimized approach):")
lazy_result.show(10, truncate=False)

# Show execution plan
print("\nEXECUTION PLAN (Optimized - Notice fewer stages):")
lazy_result.explain(mode="simple")
```

```
================================================================================
VERSION 2: LAZY/OPTIMIZED APPROACH (Best Practices)
================================================================================
Optimizations:
  ✅ Filter EARLY (reduce data volume immediately)
  ✅ Combine filters together (single pass)
  ✅ Join AFTER filtering (smaller dataset)
  ✅ Minimize shuffles through intelligent ordering
  ✅ Use appropriate partitioning

[Step 1] Filtering data FIRST (early reduction)...
  Time: 0.00s | Rows after filter: 115,697

[Step 2] Adding column transformations on filtered data...
  Time: 0.00s

[Step 3] GroupBy on filtered dataset...
  Time: 0.00s | Rows: 115,697

[Step 4] Joining with population data (small dataset)...
  Time: 0.00s

[Step 5] Final aggregation...
  Time: 0.76s | Final rows: 52

--------------------------------------------------------------------------------
LAZY/OPTIMIZED APPROACH SUMMARY:
  Total execution time: 2.26 seconds
  Number of shuffles: ~2 (one groupBy, one final aggregation)
  Data processed: Filtered dataset only
--------------------------------------------------------------------------------

Sample results (Optimized approach):
+----------+------+----+-----+----------+-------------+--------------+-----------
-------+-----------------+------------------+
|state     |region|year|month|population|monthly_cases|monthly_deaths|avg_mortali
ty_rate|counties_reported|cases_per_100k     |
+----------+------+----+-----+----------+-------------+--------------+-----------
-------+-----------------+------------------+
|California|West  |2020|2    |39538223  |44           |0             |0.0
|4                |0.11128471808153846|
|California|West  |2020|3    |39538223  |53893        |1075          |1.953142131
1769432|468              |136.3060752629171  |
|California|West  |2020|4    |39538223  |864363       |30166         |3.230965838
7784093|1234             |2186.1452903434733 |
|California|West  |2020|5    |39538223  |2475042      |99347         |3.099645902
642947 |1411             |6259.871618408344  |
|California|West  |2020|6    |39538223  |4904141      |155680        |2.248905101
7364265|1502             |12403.544286752594 |
|California|West  |2020|7    |39538223  |11404770     |232621        |1.213923246
6490384|1671             |28844.92305079062  |
|California|West  |2020|8    |39538223  |19168191     |349458        |1.311167007
2947847|1683             |48480.1529901837   |
|California|West  |2020|9    |39538223  |23157578     |438311        |1.475173575
2790125|1680             |58570.10316320994  |
|California|West  |2020|10   |39538223  |27163707     |522654        |1.552333838
8611123|1736             |68702.3971714662   |
|California|West  |2020|11   |39538223  |31809052     |551791        |1.380599060
6541378|1720             |80451.39509684085  |
+----------+------+----+-----+----------+-------------+--------------+-----------
```

```
-------+----------------+------------------+
only showing top 10 rows

EXECUTION PLAN (Optimized - Notice fewer stages):
== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- == Initial Plan ==
   ColumnarToRow
   +- PhotonResultStage
      +- PhotonSort [state#14466 ASC NULLS FIRST, year#15874 ASC NULLS FIRST, mon
th#15876 ASC NULLS FIRST]
         +- PhotonShuffleExchangeSource
            +- PhotonShuffleMapStage ENSURE_REQUIREMENTS, [id=#21171]
               +- PhotonShuffleExchangeSink rangepartitioning(state#14466 ASC NUL
LS FIRST, year#15874 ASC NULLS FIRST, month#15876 ASC NULLS FIRST, 16)
                  +- PhotonProject [state#14466, region#15893, year#15874, month#
15876, population#15892L, sum(total_cases)#15895L AS monthly_cases#15883L, sum(to
tal_deaths)#15896L AS monthly_deaths#15884L, avg(avg_mortality_rate)#15897 AS avg
_mortality_rate#15885, count(1)#15894L AS counties_reported#15886L, ((cast(sum(to
tal_cases)#15895L as double) / cast(population#15892L as double)) * 100000.0) AS
cases_per_100k#15899]
                     +- PhotonGroupingAgg(keys=[state#14466, region#15893, year#1
5874, month#15876, population#15892L], functions=[finalmerge_sum(merge sum#15901
L) AS sum(total_cases)#15895L, finalmerge_sum(merge sum#15903L) AS sum(total_deat
hs)#15896L, finalmerge_avg(merge sum#15906, count#15907L) AS avg(avg_mortality_ra
te)#15897, finalmerge_count(merge count#15909L) AS count(1)#15894L])
                        +- PhotonProject [state#14466, region#15893, year#15874,
month#15876, population#15892L, sum#15901L, sum#15903L, sum#15906, count#15907L,
count#15909L]
                           +- PhotonBroadcastHashJoin [state#14466], [state#1589
1], LeftOuter, BuildRight, false, true
                              :- PhotonGroupingAgg(keys=[state#14466, year#15874,
month#15876], functions=[partial_sum(total_cases#15879L) AS sum#15901L, partial_s
um(total_deaths#15880L) AS sum#15903L, partial_avg(avg_mortality_rate#15881) AS
(sum#15906, count#15907L), partial_count(1) AS count#15909L])
                              :  +- PhotonGroupingAgg(keys=[state#14466, county#1
4465, date#14464, year#15874, month#15876], functions=[sum(cases_int#15868), sum
(deaths_int#15870), avg(mortality_rate#15872)])
                              :     +- PhotonProject [date#14464, county#14465, s
tate#14466, cases_int#15868, deaths_int#15870, CASE WHEN (cases_int#15868 > 0) TH
EN ((cast(deaths_int#15870 as double) / cast(cases_int#15868 as double)) * 100.0)
ELSE 0.0 END AS mortality_rate#15872, year(cast(date#14464 as date)) AS year#1587
4, month(cast(date#14464 as date)) AS month#15876]
                              :        +- PhotonProject [date#14464, county#1446
5, state#14466, cast(cases#14468 as int) AS cases_int#15868, cast(deaths#14469 as
int) AS deaths_int#15870]
                              :           +- PhotonShuffleExchangeSource
                              :              +- PhotonShuffleMapStage REPARTITION
_BY_NUM, [id=#21144]
                              :                 +- PhotonShuffleExchangeSink hash
partitioning(state#14466, 4)
                              :                    +- PhotonFilter ((((isnotnull
(date#14464) AND isnotnull(cases#14468)) AND (year(cast(date#14464 as date)) = 20
20)) AND (cast(cases#14468 as bigint) >= 10)) AND state#14466 IN (California,New
York,Texas,Florida,Washington))
                              :                       +- PhotonRowToColumnar
                              :                          +- FileScan csv [date#14
464,county#14465,state#14466,cases#14468,deaths#14469] Batched: false, DataFilter
s: [isnotnull(date#14464), isnotnull(cases#14468), (year(cast(date#14464 as dat
e)) = 2020), (cast(ca..., Format: CSV, Location: InMemoryFileIndex(1 paths)[dbf
```

```
s:/databricks-datasets/COVID/covid-19-data], PartitionFilters: [], PushedFilters:
[IsNotNull(date), IsNotNull(cases), In(state, [California,Florida,New York,Texas,
Washington])], ReadSchema: struct<date:string,county:string,state:string,cases:st
ring,deaths:string>
                              +- PhotonShuffleExchangeSource
                                 +- PhotonShuffleMapStage EXECUTOR_BROADCAST, [id
=#21160]
                                    +- PhotonShuffleExchangeSink SinglePartition
                                       +- PhotonFilter (isnotnull(state#15891) AN
D state#15891 IN (California,New York,Texas,Florida,Washington))
                                          +- PhotonRowToColumnar
                                             +- LocalTableScan [state#15891, popu
lation#15892L, region#15893]


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

In [0]:
```python
# ================================================================================
# PERFORMANCE COMPARISON
# ================================================================================
print("\n" + "=" * 80)
print("PERFORMANCE COMPARISON")
print("=" * 80)

speedup = eager_total / lazy_total
time_saved = eager_total - lazy_total
percent_improvement = ((eager_total - lazy_total) / eager_total) * 100

print(f"Eager/Inefficient Approach: {eager_total:6.2f} seconds")
print(f"Lazy/Optimized Approach: {lazy_total:6.2f} seconds")
print(f"Time Saved: {time_saved:6.2f} seconds")
print(f"Speedup: {speedup:6.2f}x faster")
print(f"Performance Improvement: {percent_improvement:6.1f}%")
```

```
================================================================================
PERFORMANCE COMPARISON
================================================================================
Eager/Inefficient Approach:   4.29 seconds
Lazy/Optimized Approach:   2.26 seconds
Time Saved:   2.03 seconds
Speedup:   1.90x faster
Performance Improvement:   47.4%
```

In [0]:
```python
# SQL QUERYING
# Make sure we have data in the correct format (extra check)
# First, cast the string columns to integers in the original dataframe
covid_df = covid_df.withColumn("cases", col("cases").cast("integer")).withColumn
    "deaths", col("deaths").cast("integer")
)

# Cast columns and add transformations
enriched_df = (
    covid_df.withColumn(
        "mortality_rate",
        when(col("cases") > 0, (col("deaths") / col("cases")) * 100).otherwise(0
    )
    .withColumn("month", month(col("date")))
    .withColumn("year", year(col("date")))
)
```

```
# Create state_stats view with properly typed columns
state_stats = enriched_df.groupBy("state", "county").agg(
    sum("cases").alias("total_cases"),
    sum("deaths").alias("total_deaths"),
    avg("mortality_rate").alias("avg_mortality_rate"),
    count("*").alias("days_reported"),
)
```

In [0]:

```
# ============================================================================
# SQL QUERY 1: TOP 10 COUNTIES WITH HIGHEST MORTALITY RATES
# ============================================================================

print("\n" + "=" * 80)
print("SQL QUERY 1: TOP 10 COUNTIES WITH HIGHEST MORTALITY RATES")
print("=" * 80)

# ----------------------------------------------------------------------------
# VERSION 1A: EAGER/INEFFICIENT SQL QUERY ❌
# ----------------------------------------------------------------------------
print("\n" + "-" * 80)
print("VERSION 1A: EAGER/INEFFICIENT APPROACH")
print("-" * 80)
print("Problems:")
print("  ❌ Computes statistics on ALL counties first")
print("  ❌ Filters AFTER expensive aggregation")
print("  ❌ No predicate pushdown")
print("  ❌ Processes unnecessary data")

eager_q1_start = time.time()

# Register view
state_stats.createOrReplaceTempView("state_stats_eager")

# INEFFICIENT: No filtering before aggregation, sorting all data
sql_eager_q1 = """
    SELECT
        state,
        county,
        total_cases,
        total_deaths,
        ROUND(avg_mortality_rate, 2) as mortality_rate_pct,
        days_reported
    FROM (
        SELECT
            state,
            county,
            total_cases,
            total_deaths,
            avg_mortality_rate,
            days_reported,
            ROW_NUMBER() OVER (ORDER BY avg_mortality_rate DESC) as rank
        FROM state_stats_eager
    ) ranked
    WHERE total_cases >= 100 AND rank <= 10
"""

print("\n[Executing inefficient query...]")
eager_result_q1 = spark.sql(sql_eager_q1)
eager_q1_time = time.time() - eager_q1_start
```

```
print(f"⏱️  Eager Query 1 Time: {eager_q1_time:.5f} seconds")

print("\nExecution Plan (Eager - notice full table scan and window function):")
eager_result_q1.explain(mode="simple")
```

```
================================================================================
SQL QUERY 1: TOP 10 COUNTIES WITH HIGHEST MORTALITY RATES
================================================================================

--------------------------------------------------------------------------------
VERSION 1A: EAGER/INEFFICIENT APPROACH
--------------------------------------------------------------------------------
Problems:
  ❌ Computes statistics on ALL counties first
  ❌ Filters AFTER expensive aggregation
  ❌ No predicate pushdown
  ❌ Processes unnecessary data

[Executing inefficient query...]
⏱️  Eager Query 1 Time: 0.16685 seconds

Execution Plan (Eager - notice full table scan and window function):
== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- == Initial Plan ==
   ColumnarToRow
   +- PhotonResultStage
      +- PhotonProject [state#14466, county#14465, total_cases#16491L, total_deat
hs#16492L, round(avg_mortality_rate#16493, 2) AS mortality_rate_pct#16894, days_r
eported#16494L]
         +- PhotonFilter ((isnotnull(total_cases#16491L) AND (rank#16893 <= 10))
AND (total_cases#16491L >= 100))
            +- PhotonWindow [state#14466, county#14465, total_cases#16491L, total
_deaths#16492L, avg_mortality_rate#16493, days_reported#16494L, row_number() wind
owspecdefinition(avg_mortality_rate#16493 DESC NULLS LAST, specifiedwindowframe(R
owFrame, unboundedpreceding$(), currentrow$())) AS rank#16893]
               +- PhotonTopK(sortOrder=[avg_mortality_rate#16493 DESC NULLS LAS
T], partitionOrderCount=0)
                  +- PhotonShuffleExchangeSource
                     +- PhotonShuffleMapStage ENSURE_REQUIREMENTS, [id=#24503]
                        +- PhotonShuffleExchangeSink SinglePartition
                           +- PhotonProject [state#14466, county#14465, total_cas
es#16491L, total_deaths#16492L, avg_mortality_rate#16493, days_reported#16494L]
                              +- PhotonFilter (_local_row_number#16912 <= 10)
                                 +- PhotonWindow [state#14466, county#14465, tota
l_cases#16491L, total_deaths#16492L, avg_mortality_rate#16493, days_reported#1649
4L, row_number() windowspecdefinition(avg_mortality_rate#16493 DESC NULLS LAST, s
pecifiedwindowframe(RowFrame, unboundedpreceding$(), currentrow$())) AS _local_ro
w_number#16912]
                                    +- PhotonTopK(sortOrder=[avg_mortality_rate#1
6493 DESC NULLS LAST], partitionOrderCount=0)
                                       +- PhotonGroupingAgg(keys=[state#14466, co
unty#14465], functions=[finalmerge_sum(merge sum#16897L) AS sum(cases)#16885L, fi
nalmerge_sum(merge sum#16899L) AS sum(deaths)#16886L, finalmerge_avg(merge sum#16
902, count#16903L) AS avg(mortality_rate)#16887, finalmerge_count(merge count#169
05L) AS count(1)#16884L])
                                          +- PhotonShuffleExchangeSource
                                             +- PhotonShuffleMapStage ENSURE_REQU
IREMENTS, [id=#24470]
                                                +- PhotonShuffleExchangeSink hash
partitioning(state#14466, county#14465, 1024)
                                                   +- PhotonGroupingAgg(keys=[sta
te#14466, county#14465], functions=[partial_sum(cases#16480) AS sum#16897L, parti
al_sum(deaths#16482) AS sum#16899L, partial_avg(mortality_rate#16485) AS (sum#169
02, count#16903L), partial_count(1) AS count#16905L])
```

```
                                                  +- PhotonProject [county#14
465, state#14466, cases#16480, deaths#16482, CASE WHEN (cases#16480 > 0) THEN ((c
ast(deaths#16482 as double) / cast(cases#16480 as double)) * 100.0) ELSE 0.0 END
AS mortality_rate#16485]
                                                     +- PhotonProject [county
#14465, state#14466, cast(cases#14468 as int) AS cases#16480, cast(deaths#14469 a
s int) AS deaths#16482]
                                                        +- PhotonRowToColumna
r
                                                           +- FileScan csv [c
ounty#14465,state#14466,cases#14468,deaths#14469] Batched: false, DataFilters:
[], Format: CSV, Location: InMemoryFileIndex(1 paths)[dbfs:/databricks-datasets/C
OVID/covid-19-data], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<
county:string,state:string,cases:string,deaths:string>


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

In [0]:

```python
# ------------------------------------------------------------------------------
# VERSION 1B: LAZY/OPTIMIZED SQL QUERY ✅
# ------------------------------------------------------------------------------
print("\n" + "-" * 80)
print("VERSION 1B: LAZY/OPTIMIZED APPROACH")
print("-" * 80)
print("Optimizations:")
print("  ✅ Filters applied early (total_cases >= 100)")
print("  ✅ Uses LIMIT instead of window function")
print("  ✅ Predicate pushdown reduces data scanned")
print("  ✅ Simplified query plan")

lazy_q1_start = time.time()

# Register view
state_stats.createOrReplaceTempView("state_stats_lazy")

# OPTIMIZED: Filter early, simple ORDER BY with LIMIT
sql_lazy_q1 = """
    SELECT
        state,
        county,
        total_cases,
        total_deaths,
        ROUND(avg_mortality_rate, 2) as mortality_rate_pct,
        days_reported
    FROM state_stats_lazy
    WHERE total_cases >= 100
    ORDER BY avg_mortality_rate DESC
    LIMIT 10
"""

print("\n[Executing optimized query...]")
lazy_result_q1 = spark.sql(sql_lazy_q1)
lazy_q1_time = time.time() - lazy_q1_start

print(f"⏱️  Lazy Query 1 Time: {lazy_q1_time:.5f} seconds")

print("\nExecution Plan (Optimized - notice simpler plan):")
lazy_result_q1.explain(mode="simple")
```

```
--------------------------------------------------------------------------------
VERSION 1B: LAZY/OPTIMIZED APPROACH
--------------------------------------------------------------------------------
Optimizations:
  ✅ Filters applied early (total_cases >= 100)
  ✅ Uses LIMIT instead of window function
  ✅ Predicate pushdown reduces data scanned
  ✅ Simplified query plan

[Executing optimized query...]
⏱️  Lazy Query 1 Time: 0.15117 seconds

Execution Plan (Optimized - notice simpler plan):
== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- == Initial Plan ==
   ColumnarToRow
   +- PhotonResultStage
      +- PhotonProject [state#14466, county#14465, total_cases#16491L, total_deat
hs#16492L, mortality_rate_pct#17136, days_reported#16494L]
         +- PhotonTopK(sortOrder=[avg_mortality_rate#16493 DESC NULLS LAST], part
itionOrderCount=0)
            +- PhotonShuffleExchangeSource
               +- PhotonShuffleMapStage ENSURE_REQUIREMENTS, [id=#25223]
                  +- PhotonShuffleExchangeSink SinglePartition
                     +- PhotonTopK(sortOrder=[avg_mortality_rate#16493 DESC NULLS
LAST], partitionOrderCount=0)
                        +- PhotonProject [state#14466, county#14465, total_cases#
16491L, total_deaths#16492L, round(avg_mortality_rate#16493, 2) AS mortality_rate
_pct#17136, days_reported#16494L, avg_mortality_rate#16493]
                           +- PhotonFilter (isnotnull(total_cases#16491L) AND (to
tal_cases#16491L >= 100))
                              +- PhotonGroupingAgg(keys=[state#14466, county#1446
5], functions=[finalmerge_sum(merge sum#17138L) AS sum(cases)#17130L, finalmerge_
sum(merge sum#17140L) AS sum(deaths)#17131L, finalmerge_avg(merge sum#17143, coun
t#17144L) AS avg(mortality_rate)#17132, finalmerge_count(merge count#17146L) AS c
ount(1)#17129L])
                                 +- PhotonShuffleExchangeSource
                                    +- PhotonShuffleMapStage ENSURE_REQUIREMENTS,
[id=#25211]
                                       +- PhotonShuffleExchangeSink hashpartition
ing(state#14466, county#14465, 1024)
                                          +- PhotonGroupingAgg(keys=[state#14466,
county#14465], functions=[partial_sum(cases#16480) AS sum#17138L, partial_sum(dea
ths#16482) AS sum#17140L, partial_avg(mortality_rate#16485) AS (sum#17143, count#
17144L), partial_count(1) AS count#17146L])
                                             +- PhotonProject [county#14465, stat
e#14466, cases#16480, deaths#16482, CASE WHEN (cases#16480 > 0) THEN ((cast(death
s#16482 as double) / cast(cases#16480 as double)) * 100.0) ELSE 0.0 END AS mortal
ity_rate#16485]
                                                +- PhotonProject [county#14465, s
tate#14466, cast(cases#14468 as int) AS cases#16480, cast(deaths#14469 as int) AS
deaths#16482]
                                                   +- PhotonRowToColumnar
                                                      +- FileScan csv [county#144
65,state#14466,cases#14468,deaths#14469] Batched: false, DataFilters: [], Format:
CSV, Location: InMemoryFileIndex(1 paths)[dbfs:/databricks-datasets/COVID/covid-1
9-data], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<county:strin
g,state:string,cases:string,deaths:string>
```

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

In [0]:
```python
# ================================================================================
# PERFORMANCE COMPARISON
# ================================================================================
print("\n" + "=" * 80)
print("PERFORMANCE COMPARISON")
print("=" * 80)

speedup_q1 = eager_q1_time / lazy_q1_time
time_saved_q1 = eager_q1_time - lazy_q1_time
percent_improvement_q1 = ((eager_q1_time - lazy_q1_time) / eager_q1_time) * 100

print(f"Eager/Inefficient Approach: {eager_q1_time:6.2f} seconds")
print(f"Lazy/Optimized Approach: {lazy_q1_time:6.2f} seconds")
print(f"Time Saved: {time_saved_q1:6.2f} seconds")
print(f"Speedup: {speedup_q1:6.2f}x faster")
print(f"Performance Improvement: {percent_improvement_q1:6.1f}%")
```

```
================================================================================
PERFORMANCE COMPARISON
================================================================================
Eager/Inefficient Approach:   0.17 seconds
Lazy/Optimized Approach:   0.15 seconds
Time Saved:   0.02 seconds
Speedup:   1.10x faster
Performance Improvement:    9.4%
```

In [0]:
```python
# ================================================================================
# SQL QUERY 2: STATE-LEVEL COMPARISON
# ================================================================================

print("\n" + "=" * 80)
print("SQL QUERY 2: STATE-LEVEL COMPARISON WITH AGGREGATIONS")
print("=" * 80)

# --------------------------------------------------------------------------------
# VERSION 2A: EAGER/INEFFICIENT SQL QUERY ❌
# --------------------------------------------------------------------------------
print("\n" + "-" * 80)
print("VERSION 2A: EAGER/INEFFICIENT APPROACH")
print("-" * 80)
print("Problems:")
print("  ❌ Multiple subqueries with repeated aggregations")
print("  ❌ Computes aggregations separately (inefficient)")
print("  ❌ Multiple passes over the same data")
print("  ❌ Unnecessary intermediate results")

eager_q2_start = time.time()

# INEFFICIENT: Multiple subqueries, repeated computations
sql_eager_q2 = """
    SELECT
        s1.state,
        s1.num_counties,
        s2.state_total_cases,
        s3.state_total_deaths,
        s4.avg_mortality_rate,
```

```
        s5.max_county_cases
    FROM (
        SELECT state, COUNT(DISTINCT county) as num_counties
        FROM state_stats_eager
        GROUP BY state
    ) s1
    LEFT JOIN (
        SELECT state, SUM(total_cases) as state_total_cases
        FROM state_stats_eager
        GROUP BY state
    ) s2 ON s1.state = s2.state
    LEFT JOIN (
        SELECT state, SUM(total_deaths) as state_total_deaths
        FROM state_stats_eager
        GROUP BY state
    ) s3 ON s1.state = s3.state
    LEFT JOIN (
        SELECT state, ROUND(AVG(avg_mortality_rate), 2) as avg_mortality_rate
        FROM state_stats_eager
        GROUP BY state
    ) s4 ON s1.state = s4.state
    LEFT JOIN (
        SELECT state, MAX(total_cases) as max_county_cases
        FROM state_stats_eager
        GROUP BY state
    ) s5 ON s1.state = s5.state
    ORDER BY s2.state_total_cases DESC
"""

print("\n[Executing inefficient query...]")
eager_result_q2 = spark.sql(sql_eager_q2)
eager_count_q2 = eager_result_q2.count()
eager_q2_time = time.time() - eager_q2_start

print(f"⏱️  Eager Query 2 Time: {eager_q2_time:.2f} seconds")

print("\nExecution Plan (Eager - notice multiple scans and joins):")
eager_result_q2.explain(mode="simple")
```

```
================================================================================
SQL QUERY 2: STATE-LEVEL COMPARISON WITH AGGREGATIONS
================================================================================

--------------------------------------------------------------------------------
VERSION 2A: EAGER/INEFFICIENT APPROACH
--------------------------------------------------------------------------------
Problems:
  ❌ Multiple subqueries with repeated aggregations
  ❌ Computes aggregations separately (inefficient)
  ❌ Multiple passes over the same data
  ❌ Unnecessary intermediate results

[Executing inefficient query...]
⏱️  Eager Query 2 Time: 0.80 seconds

Execution Plan (Eager - notice multiple scans and joins):
== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- == Initial Plan ==
   ColumnarToRow
   +- PhotonResultStage
      +- PhotonSort [state_total_cases#17617L DESC NULLS LAST]
         +- PhotonShuffleExchangeSource
            +- PhotonShuffleMapStage ENSURE_REQUIREMENTS, [id=#27470]
               +- PhotonShuffleExchangeSink rangepartitioning(state_total_cases#1
7617L DESC NULLS LAST, 1024)
                  +- PhotonProject [state#14466, num_counties#17616L, state_total
_cases#17617L, state_total_deaths#17618L, avg_mortality_rate#17619, max_county_ca
ses#17620L]
                     +- PhotonShuffledHashJoin [state#14466], [state#17641], Left
Outer, BuildRight
                        :- PhotonProject [state#14466, num_counties#17616L, state
_total_cases#17617L, state_total_deaths#17618L, avg_mortality_rate#17619]
                        :  +- PhotonShuffledHashJoin [state#14466], [state#1763
5], LeftOuter, BuildRight
                        :     :- PhotonProject [state#14466, num_counties#17616L,
state_total_cases#17617L, state_total_deaths#17618L]
                        :     :  +- PhotonShuffledHashJoin [state#14466], [state#
17629], LeftOuter, BuildRight
                        :     :     :- PhotonProject [state#14466, num_counties#1
7616L, state_total_cases#17617L]
                        :     :     :  +- PhotonShuffledHashJoin [state#14466],
[state#17623], LeftOuter, BuildRight
                        :     :     :     :- PhotonGroupingAgg(keys=[state#1446
6], functions=[finalmerge_count(distinct merge count#17655L) AS count(county)#176
45L])
                        :     :     :     :  +- PhotonShuffleExchangeSource
                        :     :     :     :     +- PhotonShuffleMapStage ENSURE_R
EQUIREMENTS, [id=#27370]
                        :     :     :     :        +- PhotonShuffleExchangeSink h
ashpartitioning(state#14466, 1024)
                        :     :     :     :           +- PhotonGroupingAgg(keys=
[state#14466], functions=[partial_count(distinct county#14465) AS count#17655L])
                        :     :     :     :              +- PhotonGroupingAgg(key
s=[state#14466, county#14465], functions=[])
                        :     :     :     :                 +- PhotonShuffleExcha
ngeSource
                        :     :     :     :                    +- PhotonShuffleMa
pStage ENSURE_REQUIREMENTS, [id=#27362]
```

```
                        :     :     :     :                        +- PhotonShuffl
eExchangeSink hashpartitioning(state#14466, county#14465, 1024)
                        :     :     :     :                           +- PhotonGro
upingAgg(keys=[state#14466, county#14465], functions=[])
                        :     :     :     :                              +- Photon
RowToColumnar
                        :     :     :     :                                 +- Fil
eScan csv [county#14465,state#14466] Batched: false, DataFilters: [], Format: CS
V, Location: InMemoryFileIndex(1 paths)[dbfs:/databricks-datasets/COVID/covid-19-
data], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<county:string,
state:string>
                        :     :     :     +- PhotonGroupingAgg(keys=[state#1762
3], functions=[finalmerge_sum(merge sum#17657L) AS sum(total_cases)#17646L])
                        :     :     :        +- PhotonShuffleExchangeSource
                        :     :     :           +- PhotonShuffleMapStage ENSURE_R
EQUIREMENTS, [id=#27383]
                        :     :     :              +- PhotonShuffleExchangeSink h
ashpartitioning(state#17623, 1024)
                        :     :     :                 +- PhotonGroupingAgg(keys=
[state#17623], functions=[partial_sum(total_cases#16491L) AS sum#17657L])
                        :     :     :                    +- PhotonProject [state#
17623, cast(cast(cases#17625 as int) as bigint) AS total_cases#16491L]
                        :     :     :                       +- PhotonFilter isnot
null(state#17623)
                        :     :     :                          +- PhotonRowToColu
mnar
                        :     :     :                             +- FileScan csv
[state#17623,cases#17625] Batched: false, DataFilters: [isnotnull(state#17623)],
Format: CSV, Location: InMemoryFileIndex(1 paths)[dbfs:/databricks-datasets/COVI
D/covid-19-data], PartitionFilters: [], PushedFilters: [IsNotNull(state)], ReadSc
hema: struct<state:string,cases:string>
                        :     :     +- PhotonGroupingAgg(keys=[state#17629], func
tions=[finalmerge_sum(merge sum#17660L) AS sum(total_deaths)#17647L])
                        :     :        +- PhotonShuffleExchangeSource
                        :     :           +- PhotonShuffleMapStage ENSURE_REQUIRE
MENTS, [id=#27403]
                        :     :              +- PhotonShuffleExchangeSink hashpar
titioning(state#17629, 1024)
                        :     :                 +- PhotonGroupingAgg(keys=[state#
17629], functions=[partial_sum(total_deaths#16492L) AS sum#17660L])
                        :     :                    +- PhotonProject [state#17629,
cast(cast(deaths#17632 as int) as bigint) AS total_deaths#16492L]
                        :     :                       +- PhotonFilter isnotnull(s
tate#17629)
                        :     :                          +- PhotonRowToColumnar
                        :     :                             +- FileScan csv [stat
e#17629,deaths#17632] Batched: false, DataFilters: [isnotnull(state#17629)], Form
at: CSV, Location: InMemoryFileIndex(1 paths)[dbfs:/databricks-datasets/COVID/cov
id-19-data], PartitionFilters: [], PushedFilters: [IsNotNull(state)], ReadSchema:
struct<state:string,deaths:string>
                        :     +- PhotonGroupingAgg(keys=[state#17635], functions=
[finalmerge_avg(merge sum#17664, count#17665L) AS avg(avg_mortality_rate)#17648])
                        :        +- PhotonShuffleExchangeSource
                        :           +- PhotonShuffleMapStage ENSURE_REQUIREMENTS,
[id=#27431]
                        :              +- PhotonShuffleExchangeSink hashpartition
ing(state#17635, 1024)
                        :                 +- PhotonGroupingAgg(keys=[state#1763
5], functions=[partial_avg(avg_mortality_rate#16493) AS (sum#17664, count#17665
L)])
```

```
                        :                    +- PhotonGroupingAgg(keys=[state#176
35, county#17634], functions=[finalmerge_avg(merge sum#16902, count#16903L) AS av
g(mortality_rate)#16887])
                        :                       +- PhotonShuffleExchangeSource
                        :                          +- PhotonShuffleMapStage ENSUR
E_REQUIREMENTS, [id=#27423]
                        :                             +- PhotonShuffleExchangeSin
k hashpartitioning(state#17635, county#17634, 1024)
                        :                                +- PhotonGroupingAgg(key
s=[state#17635, county#17634], functions=[partial_avg(mortality_rate#16485) AS (s
um#16902, count#16903L)])
                        :                                   +- PhotonProject [cou
nty#17634, state#17635, CASE WHEN (cases#16480 > 0) THEN ((cast(deaths#16482 as d
ouble) / cast(cases#16480 as double)) * 100.0) ELSE 0.0 END AS mortality_rate#164
85]
                        :                                      +- PhotonProject
[county#17634, state#17635, cast(cases#17637 as int) AS cases#16480, cast(deaths#
17638 as int) AS deaths#16482]
                        :                                         +- PhotonFilter
isnotnull(state#17635)
                        :                                            +- PhotonRow
ToColumnar
                        :                                               +- FileSc
an csv [county#17634,state#17635,cases#17637,deaths#17638] Batched: false, DataFi
lters: [isnotnull(state#17635)], Format: CSV, Location: InMemoryFileIndex(1 path
s)[dbfs:/databricks-datasets/COVID/covid-19-data], PartitionFilters: [], PushedFi
lters: [IsNotNull(state)], ReadSchema: struct<county:string,state:string,cases:st
ring,deaths:string>
                     +- PhotonGroupingAgg(keys=[state#17641], functions=[final
merge_max(merge max#17671L) AS max(total_cases)#17649L])
                        +- PhotonShuffleExchangeSource
                           +- PhotonShuffleMapStage ENSURE_REQUIREMENTS, [id=#
27458]
                              +- PhotonShuffleExchangeSink hashpartitioning(st
ate#17641, 1024)
                                 +- PhotonGroupingAgg(keys=[state#17641], func
tions=[partial_max(total_cases#16491L) AS max#17671L])
                                    +- PhotonGroupingAgg(keys=[state#17641, co
unty#17640], functions=[finalmerge_sum(merge sum#16897L) AS sum(cases)#16885L])
                                       +- PhotonShuffleExchangeSource
                                          +- PhotonShuffleMapStage ENSURE_REQU
IREMENTS, [id=#27450]
                                             +- PhotonShuffleExchangeSink hash
partitioning(state#17641, county#17640, 1024)
                                                +- PhotonGroupingAgg(keys=[sta
te#17641, county#17640], functions=[partial_sum(cases#16480) AS sum#16897L])
                                                   +- PhotonProject [county#17
640, state#17641, cast(cases#17643 as int) AS cases#16480]
                                                      +- PhotonFilter isnotnul
l(state#17641)
                                                         +- PhotonRowToColumna
r
                                                            +- FileScan csv [c
ounty#17640,state#17641,cases#17643] Batched: false, DataFilters: [isnotnull(stat
e#17641)], Format: CSV, Location: InMemoryFileIndex(1 paths)[dbfs:/databricks-dat
asets/COVID/covid-19-data], PartitionFilters: [], PushedFilters: [IsNotNull(stat
e)], ReadSchema: struct<county:string,state:string,cases:string>
```

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

In [0]:

```python
# ------------------------------------------------------------------------------
# VERSION 2B: LAZY/OPTIMIZED SQL QUERY ✅
# ------------------------------------------------------------------------------
print("\n" + "-" * 80)
print("VERSION 2B: LAZY/OPTIMIZED APPROACH")
print("-" * 80)
print("Optimizations:")
print("  ✅ Single GROUP BY with all aggregations together")
print("  ✅ One pass over data")
print("  ✅ No unnecessary joins")
print("  ✅ Efficient execution plan")

lazy_q2_start = time.time()

# OPTIMIZED: Single GROUP BY with all aggregations
sql_lazy_q2 = """
    SELECT
        state,
        COUNT(DISTINCT county) as num_counties,
        SUM(total_cases) as state_total_cases,
        SUM(total_deaths) as state_total_deaths,
        ROUND(AVG(avg_mortality_rate), 2) as avg_mortality_rate,
        MAX(total_cases) as max_county_cases
    FROM state_stats_lazy
    GROUP BY state
    ORDER BY state_total_cases DESC
"""

print("\n[Executing optimized query...]")
lazy_result_q2 = spark.sql(sql_lazy_q2)
lazy_count_q2 = lazy_result_q2.count()
lazy_q2_time = time.time() - lazy_q2_start

print(f"⏱️  Lazy Query 2 Time: {lazy_q2_time:.2f} seconds")

print("\nExecution Plan (Optimized - notice single scan):")
lazy_result_q2.explain(mode="simple")
```

```
--------------------------------------------------------------------------------
VERSION 2B: LAZY/OPTIMIZED APPROACH
--------------------------------------------------------------------------------
Optimizations:
  ✅ Single GROUP BY with all aggregations together
  ✅ One pass over data
  ✅ No unnecessary joins
  ✅ Efficient execution plan

[Executing optimized query...]
⏱️  Lazy Query 2 Time: 0.73 seconds

Execution Plan (Optimized - notice single scan):
== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- == Initial Plan ==
   ColumnarToRow
   +- PhotonResultStage
      +- PhotonSort [state_total_cases#17835L DESC NULLS LAST]
         +- PhotonShuffleExchangeSource
            +- PhotonShuffleMapStage ENSURE_REQUIREMENTS, [id=#28003]
               +- PhotonShuffleExchangeSink rangepartitioning(state_total_cases#1
7835L DESC NULLS LAST, 1024)
                  +- PhotonGroupingAgg(keys=[state#14466], functions=[finalmerge_
sum(merge sum#17847L) AS sum(total_cases)#17840L, finalmerge_sum(merge sum#17849
L) AS sum(total_deaths)#17841L, finalmerge_avg(merge sum#17852, count#17853L) AS
avg(avg_mortality_rate)#17842, finalmerge_max(merge max#17855L) AS max(total_case
s)#17843L, finalmerge_count(distinct merge count#17845L) AS count(county)#17839
L])
                     +- PhotonShuffleExchangeSource
                        +- PhotonShuffleMapStage ENSURE_REQUIREMENTS, [id=#27997]
                           +- PhotonShuffleExchangeSink hashpartitioning(state#14
466, 1024)
                              +- PhotonGroupingAgg(keys=[state#14466], functions=
[merge_sum(merge sum#17847L) AS sum#17847L, merge_sum(merge sum#17849L) AS sum#17
849L, merge_avg(merge sum#17852, count#17853L) AS (sum#17852, count#17853L), merg
e_max(merge max#17855L) AS max#17855L, partial_count(distinct county#14465) AS co
unt#17845L])
                                 +- PhotonGroupingAgg(keys=[state#14466, county#1
4465], functions=[merge_sum(merge sum#17847L) AS sum#17847L, merge_sum(merge sum#
17849L) AS sum#17849L, merge_avg(merge sum#17852, count#17853L) AS (sum#17852, co
unt#17853L), merge_max(merge max#17855L) AS max#17855L])
                                    +- PhotonGroupingAgg(keys=[state#14466, count
y#14465], functions=[partial_sum(total_cases#16491L) AS sum#17847L, partial_sum(t
otal_deaths#16492L) AS sum#17849L, partial_avg(avg_mortality_rate#16493) AS (sum#
17852, count#17853L), partial_max(total_cases#16491L) AS max#17855L])
                                       +- PhotonGroupingAgg(keys=[state#14466, co
unty#14465], functions=[finalmerge_sum(merge sum#17138L) AS sum(cases)#17130L, fi
nalmerge_sum(merge sum#17140L) AS sum(deaths)#17131L, finalmerge_avg(merge sum#17
143, count#17144L) AS avg(mortality_rate)#17132])
                                          +- PhotonShuffleExchangeSource
                                             +- PhotonShuffleMapStage ENSURE_REQU
IREMENTS, [id=#27985]
                                                +- PhotonShuffleExchangeSink hash
partitioning(state#14466, county#14465, 1024)
                                                   +- PhotonGroupingAgg(keys=[sta
te#14466, county#14465], functions=[partial_sum(cases#16480) AS sum#17138L, parti
al_sum(deaths#16482) AS sum#17140L, partial_avg(mortality_rate#16485) AS (sum#171
43, count#17144L)])
                                                      +- PhotonProject [county#14
```

```
465, state#14466, cases#16480, deaths#16482, CASE WHEN (cases#16480 > 0) THEN ((c
ast(deaths#16482 as double) / cast(cases#16480 as double)) * 100.0) ELSE 0.0 END
AS mortality_rate#16485]
                                                              +- PhotonProject [county
#14465, state#14466, cast(cases#14468 as int) AS cases#16480, cast(deaths#14469 a
s int) AS deaths#16482]
                                                                 +- PhotonRowToColumna
r
                                                                    +- FileScan csv [c
ounty#14465,state#14466,cases#14468,deaths#14469] Batched: false, DataFilters:
[], Format: CSV, Location: InMemoryFileIndex(1 paths)[dbfs:/databricks-datasets/C
OVID/covid-19-data], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<
county:string,state:string,cases:string,deaths:string>


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

In [0]:
```python
# ================================================================================
# PERFORMANCE COMPARISON
# ================================================================================
print("\n" + "=" * 80)
print("PERFORMANCE COMPARISON")
print("=" * 80)

speedup_q2 = eager_q2_time / lazy_q2_time
time_saved_q2 = eager_q2_time - lazy_q2_time
percent_improvement_q2 = ((eager_q2_time - lazy_q2_time) / eager_q2_time) * 100

print(f"Eager/Inefficient Approach: {eager_q2_time:6.2f} seconds")
print(f"Lazy/Optimized Approach: {lazy_q2_time:6.2f} seconds")
print(f"Time Saved: {time_saved_q2:6.2f} seconds")
print(f"Speedup: {speedup_q2:6.2f}x faster")
print(f"Performance Improvement: {percent_improvement_q2:6.1f}%")
```

```
================================================================================
PERFORMANCE COMPARISON
================================================================================
Eager/Inefficient Approach:   0.80 seconds
Lazy/Optimized Approach:   0.73 seconds
Time Saved:   0.07 seconds
Speedup:   1.10x faster
Performance Improvement:    8.8%
```

In [0]:
```python
# Write results to a destination - my VOLUME on Databricks
output_base = "/Volumes/pyspark_assn_aj463/table1_output_schema/table1_volume_te

output_path_state_pop = f"{output_base}/output_path_state_pop"
state_population.write.mode("overwrite").parquet(output_path_state_pop)
```

In [0]:
```python
# writing all outputs
output_path_eager_result = f"{output_base}/eager_result"
eager_result.write.mode("overwrite").parquet(output_path_eager_result)

output_path_eager_result = f"{output_base}/eager_result"
state_population.write.mode("overwrite").parquet(output_path_eager_result)

output_path_lazy_result = f"{output_base}/lazy_result"
lazy_result.write.mode("overwrite").parquet(output_path_lazy_result)
```