PHASE 1

```
from pyspark.sql import SparkSession, functions as F
from pyspark.sql.types import IntegerType
from pyspark import StorageLevel

spark = (
    SparkSession.builder
    .appName("Phase1_Ingestion_Cleaning")
    .config("spark.sql.legacy.timeParserPolicy", "LEGACY")
    .getOrCreate()
)
spark.sparkContext.setLogLevel("WARN")

INPUT_CSV = "/path/to/orders.csv"
```

```
def norm_str(col):
    """
    Trim -> collapse multiple spaces -> lowercase.
    (Keep normalized for consistent joins/filters; use initcap later for display if needed.)
    """
    return F.lower(F.regexp_replace(F.trim(col), r"\s+", " "))

def to_clean_amount(col_name: str):
    """
    Keep digits only (strips commas/quotes/words), cast to INT; empty -> NULL.
    """
    digits = F.regexp_replace(F.col(col_name), r"[^0-9]", "")
    return F.when(F.length(digits) > 0, digits.cast(IntegerType())).otherwise(F.lit(None).cast(IntegerType()))

def to_clean_date_any(col):
    """
    Parse heterogeneous date formats safely.
    Tries: yyyy-MM-dd, dd/MM/yyyy, MM/dd/yyyy, yyyy/MM/dd
    Invalid tokens -> NULL.
    """
    return F.coalesce(
        F.to_date(col, "yyyy-MM-dd"),
        F.to_date(col, "dd/MM/yyyy"),
        F.to_date(col, "MM/dd/yyyy"),
        F.to_date(col, "yyyy/MM/dd")
    )
```

```
# --- Task 1: Ingestion (all StringType, no inference) ---
raw_df = (
    spark.read
    .option("header", True)
    .option("inferSchema", False)
    .csv(INPUT_CSV)
)

print("Task 1 — Raw row count:", raw_df.count())
raw_df.show(5, truncate=False)
raw_df.printSchema()
```

Show hidden output

Next steps: Explain error

```
# --- Task 1: Robust ingestion (detect uploaded orders.csv automatically) ---
from pyspark.sql import SparkSession, functions as F
import os, glob, sys

spark = (
    SparkSession.builder
```

```
    .appName("Phase1_Task1_AutoPath")
    .config("spark.sql.legacy.timeParserPolicy", "LEGACY")
    .getOrCreate()
)
spark.sparkContext.setLogLevel("WARN")

def _local_exists(p: str) -> bool:
    try:
        return os.path.exists(p)
    except Exception:
        return False

def _try_dbfs_exists(p: str) -> bool:

    try:

        dbutils
        dbutils.fs.ls(p)
        return True
    except Exception:
        return False

candidates = []

candidates += [os.path.join(os.getcwd(), "orders.csv")]
candidates += glob.glob(os.path.join(os.getcwd(), "**", "orders.csv"), recursive=True)

candidates += [
    "/content/orders.csv",
    "/workspace/orders.csv",
    "/home/jovyan/work/orders.csv",
    "/databricks/driver/orders.csv",
    "/dbfs/FileStore/orders.csv",
    "dbfs:/FileStore/orders.csv",
    "dbfs:/FileStore/tables/orders.csv",
]

candidates = list(dict.fromkeys(candidates))

INPUT_CSV = None
for p in candidates:
    if p.startswith("dbfs:/"):
        if _try_dbfs_exists(p):
            INPUT_CSV = p
            break
    else:
        if _local_exists(p):
            INPUT_CSV = p
            break

if INPUT_CSV is None:

    print(" Could not auto-detect 'orders.csv'.")
    print("  Please verify the exact filename and where it was uploaded.")
    print("   - If you used a notebook file uploader, it’s often next to the notebook.")
    print("   - In Colab: /content/orders.csv")
    print("   - In Databricks (Workspace upload): dbfs:/FileStore/tables/orders.csv")
    raise FileNotFoundError("orders.csv not found in typical locations.")
else:
    print(f" Detected orders.csv at: {INPUT_CSV}")


reader = (
    spark.read
    .option("header", True)
    .option("inferSchema", False)
)


raw_df = reader.csv(INPUT_CSV)

print("Task 1 — Raw row count:", raw_df.count())
raw_df.show(5, truncate=False)
raw_df.printSchema()
```

```
✅ Detected orders.csv at: /content/orders.csv
Task 1 — Raw row count: 300000
+-----------+-----------+-----------+-----------+-----------+-------+----------+---------+
|order_id   |customer_id|city       |category   |product    |amount |order_date|status   |
+-----------+-----------+-----------+-----------+-----------+-------+----------+---------+
|ORD00000000|C000000    | hyderabad | grocery   |Oil        |invalid|01/01/2024|Cancelled|
|ORD00000001|C000001    |Pune       |Grocery    |Sugar      |35430  |2024-01-02|Completed|
|ORD00000002|C000002    |Pune       |Electronics|Mobile     |65358  |2024-01-03|Completed|
|ORD00000003|C000003    |Bangalore  |Electronics|Laptop     |5558   |2024-01-04|Completed|
|ORD00000004|C000004    |Pune       |Home       |AirPurifier|33659  |2024-01-05|Completed|
+-----------+-----------+-----------+-----------+-----------+-------+----------+---------+
only showing top 5 rows
root
 |-- order_id: string (nullable = true)
 |-- customer_id: string (nullable = true)
 |-- city: string (nullable = true)
 |-- category: string (nullable = true)
 |-- product: string (nullable = true)
 |-- amount: string (nullable = true)
 |-- order_date: string (nullable = true)
 |-- status: string (nullable = true)
```

```
# --- Task 2: Normalize text fields (trim/space-collapsed/lowercase) ---

from pyspark.sql import functions as F

def norm_str(col):
    """
    Trim -> collapse multiple spaces -> lowercase.
    Keep normalized for consistent joins/filters.
    For presentation later, you can use F.initcap() on copies.
    """
    return F.lower(F.regexp_replace(F.trim(col), r"\s+", " "))

stage_norm_df = (
    raw_df
    .withColumn("order_id", F.trim("order_id"))
    .withColumn("customer_id", F.trim("customer_id"))
    .withColumn("city", norm_str(F.col("city")))
    .withColumn("category", norm_str(F.col("category")))
    .withColumn("product", norm_str(F.col("product")))
    .withColumn("status", norm_str(F.col("status")))
)

print("Task 2 — Sample normalized rows:")
stage_norm_df.select(
    "order_id", "customer_id", "city", "category", "product", "status"
).show(10, truncate=False)
```

```
Task 2 — Sample normalized rows:
+-----------+-----------+---------+-----------+-----------+---------+
|order_id   |customer_id|city     |category   |product    |status   |
+-----------+-----------+---------+-----------+-----------+---------+
|ORD00000000|C000000    |hyderabad|grocery    |oil        |cancelled|
|ORD00000001|C000001    |pune     |grocery    |sugar      |completed|
|ORD00000002|C000002    |pune     |electronics|mobile     |completed|
|ORD00000003|C000003    |bangalore|electronics|laptop     |completed|
|ORD00000004|C000004    |pune     |home       |airpurifier|completed|
|ORD00000005|C000005    |delhi    |fashion    |jeans      |completed|
|ORD00000006|C000006    |delhi    |grocery    |sugar      |completed|
|ORD00000007|C000007    |pune     |grocery    |rice       |completed|
|ORD00000008|C000008    |bangalore|fashion    |jeans      |completed|
|ORD00000009|C000009    |kolkata  |electronics|laptop     |completed|
+-----------+-----------+---------+-----------+-----------+---------+
only showing top 10 rows
```

```
# --- Task 3: Sanitize 'amount' to integer ---

from pyspark.sql import functions as F
from pyspark.sql.types import IntegerType

def to_clean_amount(col_name: str):
    """
    Keep digits only (strips commas, quotes, text like 'invalid'),
    cast to INT; empty result -> NULL.
    Examples:
      "12,000"   -> 12000
```

```
      " 850 "    -> 850
      "invalid"  -> NULL
      "" or None -> NULL
    """
    digits_only = F.regexp_replace(F.col(col_name), r"[^0-9]", "")
    return F.when(F.length(digits_only) > 0, digits_only.cast(IntegerType())) \
            .otherwise(F.lit(None).cast(IntegerType()))

stage_amt_df = stage_norm_df.withColumn("amount_clean", to_clean_amount("amount"))

print("Task 3 — Amount cleaning sample:")
stage_amt_df.select("amount", "amount_clean").show(15, truncate=False)

null_amt_cnt = stage_amt_df.filter(F.col("amount_clean").isNull()).count()
print("Task 3 — Rows with NULL amount_clean:", null_amt_cnt)
```

```
Task 3 — Amount cleaning sample:
+-------+------------+
|amount |amount_clean|
+-------+------------+
|invalid|NULL        |
|35430  |35430       |
|65358  |65358       |
|5558   |5558        |
|33659  |33659       |
|8521   |8521        |
|42383  |42383       |
|45362  |45362       |
|10563  |10563       |
|63715  |63715       |
|66576  |66576       |
|50318  |50318       |
|84768  |84768       |
|79121  |79121       |
|79469  |79469       |
+-------+------------+
only showing top 15 rows
Task 3 — Rows with NULL amount_clean: 25164
```

```
# --- Task 4: Parse heterogeneous order_date formats into order_date_clean ---

from pyspark.sql import functions as F

def to_clean_date_any(col):
    """
    Try multiple common formats:
      - yyyy-MM-dd  (e.g., 2024-01-15)
      - dd/MM/yyyy  (e.g., 15/01/2024)
      - MM/dd/yyyy  (e.g., 01/15/2024)
      - yyyy/MM/dd  (e.g., 2024/01/15)
    If none match -> NULL
    """
    return F.coalesce(
        F.to_date(col, "yyyy-MM-dd"),
        F.to_date(col, "dd/MM/yyyy"),
        F.to_date(col, "MM/dd/yyyy"),
        F.to_date(col, "yyyy/MM/dd")
    )


stage_date_df = stage_amt_df.withColumn("order_date_clean", to_clean_date_any(F.col("order_date")))

print("Task 4 — Date parsing sample:")
stage_date_df.select("order_date", "order_date_clean").show(15, truncate=False)

null_date_cnt = stage_date_df.filter(F.col("order_date_clean").isNull()).count()
print("Task 4 — Rows with NULL order_date_clean:", null_date_cnt)
```

```
Task 4 — Date parsing sample:
---------------------------------------------------------------------------
DateTimeException                         Traceback (most recent call last)
/tmp/ipython-input-3369566718.py in <cell line: 0>()
     23
     24 print("Task 4 — Date parsing sample:")
---> 25 stage_date_df.select("order_date", "order_date_clean").show(15, truncate=False)
     26
     27 null_date_cnt = stage_date_df.filter(F.col("order_date_clean").isNull()).count()

                                3 frames
/usr/local/lib/python3.12/dist-packages/pyspark/errors/exceptions/captured.py in deco(*a, **kw)
    286                 # Hide where the exception came from that shows a non-Pythonic
    287                 # JVM exception message.
--> 288                 raise converted from None
    289             else:
    290                 raise

DateTimeException: [CANNOT_PARSE_TIMESTAMP] Unparseable date: "01/01/2024". Use `try_to_timestamp` to tolerate invalid input
string and return NULL instead. SQLSTATE: 22007
```

Next steps: Explain error

```
# Make mixed-format parsing tolerant and avoid ANSI errors
spark.conf.set("spark.sql.ansi.enabled", "false")
spark.conf.set("spark.sql.legacy.timeParserPolicy", "LEGACY")
```

```
# --- Task 4: Parse heterogeneous order_date into order_date_clean (bulletproof) ---

from pyspark.sql import functions as F


stage_date_prep_df = stage_amt_df.withColumn("order_date_trim", F.trim(F.col("order_date")))
is_yyyy_mm_dd_dash  = F.col("order_date_trim").rlike(r"^\d{4}-\d{2}-\d{2}$")
is_yyyy_mm_dd_slash = F.col("order_date_trim").rlike(r"^\d{4}/\d{2}/\d{2}$")
is_dd_mm_yyyy_slash = F.col("order_date_trim").rlike(r"^\d{2}/\d{2}/\d{4}$")


parts = F.split(F.col("order_date_trim"), "/")
first  = F.when(is_dd_mm_yyyy_slash, parts.getItem(0).cast("int"))
second = F.when(is_dd_mm_yyyy_slash, parts.getItem(1).cast("int"))


cond_def_ddmm = (first > 12) & (second <= 12)
cond_def_mmdd = (first <= 12) & (second > 12)
cond_ambiguous = (first <= 12) & (second <= 12)

order_date_clean = (
    F.when(is_yyyy_mm_dd_dash,  F.to_date(F.col("order_date_trim"), "yyyy-MM-dd"))
     .when(is_yyyy_mm_dd_slash, F.to_date(F.col("order_date_trim"), "yyyy/MM/dd"))

     .when(is_dd_mm_yyyy_slash & cond_def_ddmm, F.to_date(F.col("order_date_trim"), "dd/MM/yyyy"))

     .when(is_dd_mm_yyyy_slash & cond_def_mmdd, F.to_date(F.col("order_date_trim"), "MM/dd/yyyy"))

     .when(is_dd_mm_yyyy_slash & cond_ambiguous, F.to_date(F.col("order_date_trim"), "dd/MM/yyyy"))

     .otherwise(F.lit(None).cast("date"))
)

stage_date_df = stage_date_prep_df.withColumn("order_date_clean", order_date_clean) \
                                  .drop("order_date_trim")

print("Task 4 — Date parsing sample:")
stage_date_df.select("order_date", "order_date_clean").show(15, truncate=False)

null_date_cnt = stage_date_df.filter(F.col("order_date_clean").isNull()).count()
print("Task 4 — Rows with NULL order_date_clean:", null_date_cnt)
```

```
Task 4 — Date parsing sample:
+----------+----------------+
|order_date|order_date_clean|
```

```
+----------+----------------+
|01/01/2024|2024-01-01      |
|2024-01-02|2024-01-02      |
|2024-01-03|2024-01-03      |
|2024-01-04|2024-01-04      |
|2024-01-05|2024-01-05      |
|2024-01-06|2024-01-06      |
|2024-01-07|2024-01-07      |
|2024-01-08|2024-01-08      |
|2024-01-09|2024-01-09      |
|2024-01-10|2024-01-10      |
|2024-01-11|2024-01-11      |
|12/01/2024|2024-01-12      |
|2024-01-13|2024-01-13      |
|2024/01/14|2024-01-14      |
|2024-01-15|2024-01-15      |
+----------+----------------+
only showing top 15 rows
Task 4 — Rows with NULL order_date_clean: 2595
```

```
dedup_df = stage_date_df.dropDuplicates(["order_id"])
print("Task 5 — Before:", stage_date_df.count(), "After:", dedup_df.count())
```

```
Task 5 — Before: 300000 After: 300000
```

```
# --- Task 5: Drop duplicates on order_id ---

from pyspark.sql import functions as F, Window

before_cnt = stage_date_df.count()

dedup_df = stage_date_df.dropDuplicates(["order_id"])

after_cnt = dedup_df.count()
dup_removed = before_cnt - after_cnt

print("Task 5 — De-duplication summary")
print(f"  Rows before de-dup : {before_cnt}")
print(f"  Rows after  de-dup : {after_cnt}")
print(f"  Duplicates removed : {dup_removed}")
dupe_ids_df = (
    stage_date_df.groupBy("order_id")
    .count()
    .filter(F.col("count") > 1)
    .orderBy(F.desc("count"))
)

print("Task 5 — Sample duplicate order_ids (if any):")
dupe_ids_df.show(10, truncate=False)
```

```
Task 5 — De-duplication summary
  Rows before de-dup : 300000
  Rows after  de-dup : 300000
  Duplicates removed : 0
Task 5 — Sample duplicate order_ids (if any):
+--------+-----+
|order_id|count|
+--------+-----+
+--------+-----+
```

```
completed_df = dedup_df.filter(F.col("status") == "completed")
print("Task 6 — Completed row count:", completed_df.count())
```

```
Task 6 — Completed row count: 285000
```

```
# --- Task 6: Keep only 'completed' orders ---

from pyspark.sql import functions as F
completed_df = dedup_df.filter(F.col("status") == F.lit("completed"))
```

```
total_after_dedup = dedup_df.count()
completed_count    = completed_df.count()
filtered_out       = total_after_dedup - completed_count

print("Task 6 — Completed filter summary")
print(f"  Rows after Task 5 (dedup): {total_after_dedup}")
print(f"  Completed rows kept:       {completed_count}")
print(f"  Non-completed filtered:    {filtered_out}")

print("Task 6 — Distinct status values after filter (should be just 'completed'):")
completed_df.select("status").distinct().show(truncate=False)

completed_df.select(
    "order_id","customer_id","status","city","category","product","amount_clean","order_date_clean"
).show(10, truncate=False)
```

```
Task 6 — Completed filter summary
  Rows after Task 5 (dedup): 300000
  Completed rows kept:       285000
  Non-completed filtered:    15000
Task 6 — Distinct status values after filter (should be just 'completed'):
+---------+
|status   |
+---------+
|completed|
+---------+

+-----------+-----------+---------+---------+-----------+-------+------------+----------------+
|order_id   |customer_id|status   |city     |category   |product|amount_clean|order_date_clean|
+-----------+-----------+---------+---------+-----------+-------+------------+----------------+
|ORD00000001|C000001    |completed|pune     |grocery    |sugar  |35430       |2024-01-02      |
|ORD00000007|C000007    |completed|pune     |grocery    |rice   |45362       |2024-01-08      |
|ORD00000008|C000008    |completed|bangalore|fashion    |jeans  |10563       |2024-01-09      |
|ORD00000010|C000010    |completed|bangalore|grocery    |sugar  |66576       |2024-01-11      |
|ORD00000011|C000011    |completed|kolkata  |electronics|tablet |50318       |2024-01-12      |
|ORD00000012|C000012    |completed|bangalore|grocery    |sugar  |84768       |2024-01-13      |
|ORD00000014|C000014    |completed|mumbai   |electronics|tablet |79469       |2024-01-15      |
|ORD00000015|C000015    |completed|pune     |electronics|mobile |81018       |2024-01-16      |
|ORD00000017|C000017    |completed|bangalore|grocery    |oil    |69582       |2024-01-18      |
|ORD00000019|C000019    |completed|mumbai   |electronics|mobile |NULL        |2024-01-20      |
+-----------+-----------+---------+---------+-----------+-------+------------+----------------+
only showing top 10 rows
```

```
# Task 7 preview:
clean_orders_df = (
    completed_df
    .filter(F.col("amount_clean").isNotNull())
    .filter(F.col("order_date_clean").isNotNull())
    .select(
        "order_id", "customer_id", "city", "category", "product",
        F.col("amount_clean").alias("amount"),
        F.col("order_date_clean").alias("order_date")
    )
)
```

```
# --- Task 7: Require valid amount & date; project final Silver schema ---

from pyspark.sql import functions as F


clean_orders_df = (
    completed_df
    .filter(F.col("amount_clean").isNotNull())
    .filter(F.col("order_date_clean").isNotNull())
    .select(
        "order_id",
        "customer_id",
        "city",
        "category",
        "product",
        F.col("amount_clean").alias("amount"),
        F.col("order_date_clean").alias("order_date"),
    )
```

```
)

post_completed_cnt = completed_df.count()
final_clean_cnt    = clean_orders_df.count()
dropped_nulls      = post_completed_cnt - final_clean_cnt

print("Task 7 — Validity filter summary")
print(f"  Rows after Task 6 (completed only): {post_completed_cnt}")
print(f"  Final clean rows (valid amount & date): {final_clean_cnt}")
print(f"  Dropped due to NULL amount/date:       {dropped_nulls}")

print("\nTask 7 — Sample of final Silver rows:")
clean_orders_df.show(15, truncate=False)

print("\nTask 7 — Final Silver schema:")
clean_orders_df.printSchema()
```

```
Task 7 — Validity filter summary
  Rows after Task 6 (completed only): 285000
  Final clean rows (valid amount & date): 258834
  Dropped due to NULL amount/date:       26166

Task 7 — Sample of final Silver rows:
+-----------+-----------+---------+-----------+-----------+------+----------+
|order_id   |customer_id|city     |category   |product    |amount|order_date|
+-----------+-----------+---------+-----------+-----------+------+----------+
|ORD00000001|C000001    |pune     |grocery    |sugar      |35430 |2024-01-02|
|ORD00000007|C000007    |pune     |grocery    |rice       |45362 |2024-01-08|
|ORD00000008|C000008    |bangalore|fashion    |jeans      |10563 |2024-01-09|
|ORD00000010|C000010    |bangalore|grocery    |sugar      |66576 |2024-01-11|
|ORD00000011|C000011    |kolkata  |electronics|tablet     |50318 |2024-01-12|
|ORD00000012|C000012    |bangalore|grocery    |sugar      |84768 |2024-01-13|
|ORD00000014|C000014    |mumbai   |electronics|tablet     |79469 |2024-01-15|
|ORD00000015|C000015    |pune     |electronics|mobile     |81018 |2024-01-16|
|ORD00000017|C000017    |bangalore|grocery    |oil        |69582 |2024-01-18|
|ORD00000022|C000022    |mumbai   |grocery    |sugar      |48832 |2024-01-23|
|ORD00000023|C000023    |hyderabad|electronics|mobile     |12000 |2024-01-24|
|ORD00000024|C000024    |bangalore|home       |mixer      |18082 |2024-01-25|
|ORD00000025|C000025    |bangalore|home       |airpurifier|58248 |2024-01-26|
|ORD00000028|C000028    |mumbai   |grocery    |sugar      |70675 |2024-01-29|
|ORD00000030|C000030    |pune     |home       |airpurifier|52112 |2024-01-31|
+-----------+-----------+---------+-----------+-----------+------+----------+
only showing top 15 rows

Task 7 — Final Silver schema:
root
 |-- order_id: string (nullable = true)
 |-- customer_id: string (nullable = true)
 |-- city: string (nullable = true)
 |-- category: string (nullable = true)
 |-- product: string (nullable = true)
 |-- amount: integer (nullable = true)
 |-- order_date: date (nullable = true)
```

```
# --- Task 8: Optimize Silver for downstream workloads (repartition + persist) ---

from pyspark import StorageLevel

optimized_clean_orders_df = clean_orders_df.repartition("customer_id")


optimized_clean_orders_df.persist(StorageLevel.MEMORY_AND_DISK)

print("Task 8 — Materializing persisted Silver...")
_ = optimized_clean_orders_df.count()

print("Task 8 — Persisted partitions:", optimized_clean_orders_df.rdd.getNumPartitions())
optimized_clean_orders_df.explain()
```

```
                                                  +- *(1) Project [order_id#71, customer_id#72, city#73, category#74, produ
                                                     +- *(1) Project [trim(order_id#17, None) AS order_id#71, trim(custome
                                                        +- FileScan csv [order_id#17,customer_id#18,city#19,category#20,pro
                  +- == Initial Plan ==
                     Exchange hashpartitioning(customer_id#791, 200), REPARTITION_BY_COL, [plan_id=1639]
                     +- Project [order_id#71, customer_id#791, city#793, category#795, product#797, amount_clean#805 AS amount#
                        +- Filter (isnotnull(status#803) AND ((status#803 = completed) AND (isnotnull(amount_clean#805) AND isn
                           +- SortAggregate(key=[order_id#71], functions=[first(customer_id#72, false), first(city#73, false),
                              +- Sort [order_id#71 ASC NULLS FIRST], false, 0
                                 +- Exchange hashpartitioning(order_id#71, 200), ENSURE_REQUIREMENTS, [plan_id=1634]
                                    +- SortAggregate(key=[order_id#71], functions=[partial_first(customer_id#72, false), partia
                                       +- Sort [order_id#71 ASC NULLS FIRST], false, 0
                                          +- Project [order_id#71, customer_id#72, city#73, category#74, product#75, status#76
                                             +- Project [trim(order_id#17, None) AS order_id#71, trim(customer_id#18, None) AS
                                                +- FileScan csv [order_id#17,customer_id#18,city#19,category#20,product#21,amou
   +- == Initial Plan ==
      InMemoryTableScan [order_id#71, customer_id#72, city#73, category#74, product#75, amount#569, order_date#570]
         +- InMemoryRelation [order_id#71, customer_id#72, city#73, category#74, product#75, amount#569, order_date#570], Storage
               +- AdaptiveSparkPlan isFinalPlan=true
               +- == Final Plan ==
                  ResultQueryStage 2
                  +- ShuffleQueryStage 1
                     +- Exchange hashpartitioning(customer_id#791, 200), REPARTITION_BY_COL, [plan_id=1741]
                        +- *(3) Project [order_id#71, customer_id#791, city#793, category#795, product#797, amount_clean#805 AS ar
                           +- *(3) Filter (isnotnull(status#803) AND ((status#803 = completed) AND (isnotnull(amount_clean#805) AN
                              +- SortAggregate(key=[order_id#71], functions=[first(customer_id#72, false), first(city#73, false),
                                 +- *(2) Sort [order_id#71 ASC NULLS FIRST], false, 0
                                    +- AQEShuffleRead coalesced
                                       +- ShuffleQueryStage 0
                                          +- Exchange hashpartitioning(order_id#71, 200), ENSURE_REQUIREMENTS, [plan_id=1688]
                                             +- SortAggregate(key=[order_id#71], functions=[partial_first(customer_id#72, false),
                                                +- *(1) Sort [order_id#71 ASC NULLS FIRST], false, 0
                                                   +- *(1) Project [order_id#71, customer_id#72, city#73, category#74, product#75
                                                      +- *(1) Project [trim(order_id#17, None) AS order_id#71, trim(customer_id#18
                                                         +- FileScan csv [order_id#17,customer_id#18,city#19,category#20,product#2
               +- == Initial Plan ==
                  Exchange hashpartitioning(customer_id#791, 200), REPARTITION_BY_COL, [plan_id=1639]
                  +- Project [order_id#71, customer_id#791, city#793, category#795, product#797, amount_clean#805 AS amount#569, c
                     +- Filter (isnotnull(status#803) AND ((status#803 = completed) AND (isnotnull(amount_clean#805) AND isnotnul
                        +- SortAggregate(key=[order_id#71], functions=[first(customer_id#72, false), first(city#73, false), first
                           +- Sort [order_id#71 ASC NULLS FIRST], false, 0
                              +- Exchange hashpartitioning(order_id#71, 200), ENSURE_REQUIREMENTS, [plan_id=1634]
                                 +- SortAggregate(key=[order_id#71], functions=[partial_first(customer_id#72, false), partial_fir
                                    +- Sort [order_id#71 ASC NULLS FIRST], false, 0
                                       +- Project [order_id#71, customer_id#72, city#73, category#74, product#75, status#76, amoun
                                          +- Project [trim(order_id#17, None) AS order_id#71, trim(customer_id#18, None) AS custom
                                             +- FileScan csv [order_id#17,customer_id#18,city#19,category#20,product#21,amount#22
```

PHASE 2

```
from pyspark.sql import functions as F
from pyspark.sql import Window

orders_df = optimized_clean_orders_df if 'optimized_clean_orders_df' in globals() else clean_orders_df

orders_df.printSchema()
orders_df.show(5, truncate=False)
```

```
root
 |-- order_id: string (nullable = true)
 |-- customer_id: string (nullable = true)
 |-- city: string (nullable = true)
 |-- category: string (nullable = true)
 |-- product: string (nullable = true)
 |-- amount: integer (nullable = true)
 |-- order_date: date (nullable = true)

+-----------+-----------+---------+-----------+-------+------+----------+
|order_id   |customer_id|city     |category   |product|amount|order_date|
+-----------+-----------+---------+-----------+-------+------+----------+
|ORD00000142|C000142    |kolkata  |electronics|tablet |47592 |2024-01-23|
|ORD00000299|C000299    |delhi    |home       |mixer  |12000 |2024-02-29|
|ORD00000433|C000433    |bangalore|electronics|mobile |62262 |2024-01-14|
|ORD00001115|C001115    |kolkata  |grocery    |oil    |23740 |2024-02-05|
|ORD00001875|C001875    |pune     |electronics|laptop |18392 |2024-01-16|
+-----------+-----------+---------+-----------+-------+------+----------+
only showing top 5 rows
```

```
# --- Task 1: Total number of orders per customer ---
orders_per_customer_df = (
    orders_df
    .groupBy("customer_id")
    .agg(F.count("*").alias("order_count"))
)

print("Task 1 — Sample:")
orders_per_customer_df.orderBy(F.desc("order_count")).show(10, truncate=False)
```

```
Task 1 — Sample:
+-----------+-----------+
|customer_id|order_count|
+-----------+-----------+
|C008123    |6          |
|C005981    |6          |
|C006415    |6          |
|C014534    |6          |
|C010172    |6          |
|C007767    |6          |
|C013439    |6          |
|C007929    |6          |
|C005044    |6          |
|C013102    |6          |
+-----------+-----------+
only showing top 10 rows
```

```
# --- Task 2: Total spending per customer ---
spend_per_customer_df = (
    orders_df
    .groupBy("customer_id")
    .agg(F.sum("amount").alias("total_spend"))
)

print("Task 2 — Sample:")
spend_per_customer_df.orderBy(F.desc("total_spend")).show(10, truncate=False)
```

```
Task 2 — Sample:
+-----------+-----------+
|customer_id|total_spend|
+-----------+-----------+
|C043076    |493949     |
|C034689    |486879     |
|C039985    |484057     |
|C026691    |477147     |
|C038979    |477138     |
|C020762    |474717     |
|C044654    |471304     |
|C014292    |468617     |
|C019565    |467523     |
|C045487    |467050     |
+-----------+-----------+
only showing top 10 rows
```

```
# --- Task 3: Average Order Value (AOV) per customer ---
aov_per_customer_df = (
    orders_df
    .groupBy("customer_id")
    .agg((F.sum("amount") / F.count("*")).alias("avg_order_value"))
)

print("Task 3 — Sample:")
aov_per_customer_df.orderBy(F.desc("avg_order_value")).show(10, truncate=False)
```

```
Task 3 — Sample:
+-----------+------------------+
|customer_id|avg_order_value   |
+-----------+------------------+
|C007483    |84969.0           |
|C007752    |84360.0           |
|C020164    |83755.75          |
|C001406    |82452.8           |
|C043076    |82324.83333333333|
```

```
|C030438    |81927.75         |
|C013495    |81694.6          |
|C036697    |81442.6          |
|C004346    |81252.2          |
|C047634    |81235.4          |
+-----------+-----------------+
only showing top 10 rows
```

```
# --- Task 4: First & last purchase dates per customer ---
first_last_dates_df = (
    orders_df
    .groupBy("customer_id")
    .agg(
        F.min("order_date").alias("first_purchase_date"),
        F.max("order_date").alias("last_purchase_date")
    )
)

print("Task 4 — Sample:")
first_last_dates_df.orderBy("first_purchase_date").show(10, truncate=False)
```

```
Task 4 — Sample:
+-----------+-------------------+------------------+
|customer_id|first_purchase_date|last_purchase_date|
+-----------+-------------------+------------------+
|C007641    |2024-01-02         |2024-02-11        |
|C005981    |2024-01-02         |2024-02-11        |
|C003561    |2024-01-02         |2024-02-11        |
|C006101    |2024-01-02         |2024-02-11        |
|C000661    |2024-01-02         |2024-02-11        |
|C023181    |2024-01-02         |2024-02-11        |
|C001461    |2024-01-02         |2024-02-11        |
|C040681    |2024-01-02         |2024-02-11        |
|C018081    |2024-01-02         |2024-02-11        |
|C048961    |2024-01-02         |2024-02-11        |
+-----------+-------------------+------------------+
only showing top 10 rows
```

```
# --- Task 5: Number of distinct cities per customer ---
distinct_cities_df = (
    orders_df
    .groupBy("customer_id")
    .agg(F.countDistinct("city").alias("distinct_cities"))
)

print("Task 5 — Sample:")
distinct_cities_df.orderBy(F.desc("distinct_cities")).show(10, truncate=False)
```

```
Task 5 — Sample:
+-----------+---------------+
|customer_id|distinct_cities|
+-----------+---------------+
|C030787    |6              |
|C011844    |6              |
|C000219    |6              |
|C008703    |6              |
|C004985    |6              |
|C015763    |6              |
|C019052    |6              |
|C033568    |6              |
|C015032    |6              |
|C042376    |6              |
+-----------+---------------+
only showing top 10 rows
```

```
# --- Task 6: Number of distinct categories per customer ---
distinct_categories_df = (
    orders_df
    .groupBy("customer_id")
    .agg(F.countDistinct("category").alias("distinct_categories"))
)

print("Task 6 — Sample:")
```

```
distinct_categories_df.orderBy(F.desc("distinct_categories")).show(10, truncate=False)
```

```
Task 6 — Sample:
+-----------+-------------------+
|customer_id|distinct_categories|
+-----------+-------------------+
|C013083    |4                  |
|C009656    |4                  |
|C011231    |4                  |
|C027788    |4                  |
|C011008    |4                  |
|C010795    |4                  |
|C022453    |4                  |
|C004684    |4                  |
|C005044    |4                  |
|C014534    |4                  |
+-----------+-------------------+
only showing top 10 rows
```

```
# --- Task 7: Customer profile in a single aggregation (efficient) ---
customer_profile_df = (
    orders_df
    .groupBy("customer_id")
    .agg(
        F.count("*").alias("order_count"),
        F.sum("amount").alias("total_spend"),
        (F.sum("amount") / F.count("*")).alias("avg_order_value"),
        F.min("order_date").alias("first_purchase_date"),
        F.max("order_date").alias("last_purchase_date"),
        F.countDistinct("city").alias("distinct_cities"),
        F.countDistinct("category").alias("distinct_categories")
    )
)

customer_profile_df = customer_profile_df.cache()
_ = customer_profile_df.count()

print("Task 7 — Customer profile sample:")
customer_profile_df.orderBy(F.desc("total_spend")).show(10, truncate=False)
customer_profile_df.printSchema()
```

```
Task 7 — Customer profile sample:
+-----------+-----------+-----------+-----------------+-------------------+------------------+---------------+------------------
|customer_id|order_count|total_spend|avg_order_value  |first_purchase_date|last_purchase_date|distinct_cities|distinct_categorie
+-----------+-----------+-----------+-----------------+-------------------+------------------+---------------+------------------
|C043076    |6          |493949     |82324.83333333333|2024-01-17         |2024-02-26        |5              |4
|C034689    |6          |486879     |81146.5          |2024-01-10         |2024-02-19        |4              |3
|C039985    |6          |484057     |80676.16666666667|2024-01-06         |2024-02-15        |3              |4
|C026691    |6          |477147     |79524.5          |2024-01-12         |2024-02-21        |4              |3
|C038979    |6          |477138     |79523.0          |2024-01-20         |2024-02-29        |5              |4
|C020762    |6          |474717     |79119.5          |2024-01-03         |2024-02-12        |5              |3
|C044654    |6          |471304     |78550.66666666667|2024-01-15         |2024-02-24        |5              |2
|C014292    |6          |468617     |78102.83333333333|2024-01-13         |2024-02-22        |5              |3
|C019565    |6          |467523     |77920.5          |2024-01-06         |2024-02-15        |3              |3
|C045487    |6          |467050     |77841.66666666667|2024-01-08         |2024-02-17        |4              |3
+-----------+-----------+-----------+-----------------+-------------------+------------------+---------------+------------------
only showing top 10 rows
root
 |-- customer_id: string (nullable = true)
 |-- order_count: long (nullable = false)
 |-- total_spend: long (nullable = true)
 |-- avg_order_value: double (nullable = true)
 |-- first_purchase_date: date (nullable = true)
 |-- last_purchase_date: date (nullable = true)
 |-- distinct_cities: long (nullable = false)
 |-- distinct_categories: long (nullable = false)
```

PHASE 3

```
import builtins
print = builtins.print
```

```
# --- Task 1: Add customer_segment based on business rules ---

from pyspark.sql import functions as F

segmented_customers_df = (
    customer_profile_df
    .withColumn(
        "customer_segment",
        F.when((F.col("total_spend") >= 200000) & (F.col("order_count") >= 5), "VIP")
         .when(F.col("total_spend") >= 100000, "Premium")
         .otherwise("Regular")
    )
)

print("Task 1 — Sample with segments:")
segmented_customers_df.select(
    "customer_id","order_count","total_spend","customer_segment"
).show(10, truncate=False)
```

```
Task 1 — Sample with segments:
+-----------+-----------+-----------+----------------+
|customer_id|order_count|total_spend|customer_segment|
+-----------+-----------+-----------+----------------+
|C018237    |4          |226546     |Premium         |
|C044374    |6          |224785     |VIP             |
|C001115    |5          |163614     |Premium         |
|C012569    |6          |270399     |VIP             |
|C010142    |5          |245547     |VIP             |
|C018622    |5          |210550     |VIP             |
|C035805    |6          |272408     |VIP             |
|C029047    |6          |218872     |VIP             |
|C040253    |6          |202009     |VIP             |
|C028333    |5          |213761     |VIP             |
+-----------+-----------+-----------+----------------+
only showing top 10 rows
```

```
# --- Task 2: Count customers in each segment ---

segment_counts_df = (
    segmented_customers_df
    .groupBy("customer_segment")
    .count()
    .orderBy(F.desc("count"))
)

print("Task 2 — Segment counts:")
segment_counts_df.show(truncate=False)
```

```
Task 2 — Segment counts:
+----------------+-----+
|customer_segment|count|
+----------------+-----+
|VIP             |32815|
|Premium         |13978|
|Regular         |707  |
+----------------+-----+
```

```
# --- Task 3: Sanity checks ---

from pyspark.sql import functions as F

vip_violations = segmented_customers_df.filter(
    (F.col("customer_segment") == "VIP") &
    ~((F.col("total_spend") >= 200_000) & (F.col("order_count") >= 5))
).count()

prem_violations = segmented_customers_df.filter(
    (F.col("customer_segment") == "Premium") &
    ~( (F.col("total_spend") >= 100_000) & ~((F.col("total_spend") >= 200_000) & (F.col("order_count") >= 5)) )
).count()

print("Task 3 — Sanity checks")
```

```
print(f"  VIP violations:     {vip_violations}")
print(f"  Premium violations: {prem_violations}")

print("\nSample VIPs:")
segmented_customers_df.filter(F.col("customer_segment")=="VIP") \
    .select("customer_id","order_count","total_spend").orderBy(F.desc("total_spend")).show(10, truncate=False)

print("\nSample Premiums:")
segmented_customers_df.filter(F.col("customer_segment")=="Premium") \
    .select("customer_id","order_count","total_spend").orderBy(F.desc("total_spend")).show(10, truncate=False)
```

```
Task 3 — Sanity checks
  VIP violations:     0
  Premium violations: 0

Sample VIPs:
+-----------+-----------+-----------+
|customer_id|order_count|total_spend|
+-----------+-----------+-----------+
|C043076    |6          |493949     |
|C034689    |6          |486879     |
|C039985    |6          |484057     |
|C026691    |6          |477147     |
|C038979    |6          |477138     |
|C020762    |6          |474717     |
|C044654    |6          |471304     |
|C014292    |6          |468617     |
|C019565    |6          |467523     |
|C045487    |6          |467050     |
+-----------+-----------+-----------+
only showing top 10 rows

Sample Premiums:
+-----------+-----------+-----------+
|customer_id|order_count|total_spend|
+-----------+-----------+-----------+
|C007483    |4          |339876     |
|C007752    |4          |337440     |
|C020164    |4          |335023     |
|C030438    |4          |327711     |
|C001997    |4          |323359     |
|C024675    |4          |319973     |
|C015586    |4          |318066     |
|C031453    |4          |316414     |
|C002776    |4          |315636     |
|C000510    |4          |314014     |
+-----------+-----------+-----------+
only showing top 10 rows
```

PHASE 4

```
#task 1
from pyspark.sql import functions as F
from pyspark.sql.window import Window

base = customer_profile_df

top10_overall = (
    base.withColumn(
            "rank",
            F.dense_rank().over(
                Window.orderBy(F.col("total_spend").desc(), F.col("order_count").desc())
            )
        )
        .filter("rank <= 10")
)

top10_overall.select("customer_id", "total_spend", "order_count", "rank").orderBy("rank").show(10, truncate=False)
```

```
+-----------+-----------+-----------+----+
|customer_id|total_spend|order_count|rank|
+-----------+-----------+-----------+----+
|C043076    |493949     |6          |1   |
|C034689    |486879     |6          |2   |
|C039985    |484057     |6          |3   |
|C026691    |477147     |6          |4   |
|C038979    |477138     |6          |5   |
|C020762    |474717     |6          |6   |
```

```
|C044654    |471304     |6          |7   |
|C014292    |468617     |6          |8   |
|C019565    |467523     |6          |9   |
|C045487    |467050     |6          |10  |
+-----------+-----------+-----------+----+
```

```
from pyspark.sql import functions as F
from pyspark.sql.window import Window

df = optimized_clean_orders_df

# --- Phase 4 · Task 2: Build per-city rankings ---
cust_city_rank = (
    df.groupBy("city", "customer_id")
      .agg(F.sum("amount").alias("city_total_spend"))
      .withColumn(
          "rank",
          F.dense_rank().over(
              Window.partitionBy("city").orderBy(F.col("city_total_spend").desc())
          )
      )
)
```

```
# --- Phase 4 · Task 3: Extract Top 3 per city from the rankings ---
top3_per_city = cust_city_rank.filter("rank <= 3")

top3_per_city.orderBy("city", "rank").show(100, truncate=False)
```

```
+---------+-----------+----------------+----+
|city     |customer_id|city_total_spend|rank|
+---------+-----------+----------------+----+
|bangalore|C011518    |332527          |1   |
|bangalore|C024935    |315622          |2   |
|bangalore|C025451    |303208          |3   |
|chennai  |C028121    |340890          |1   |
|chennai  |C027841    |287392          |2   |
|chennai  |C030712    |284466          |3   |
|delhi    |C016309    |325001          |1   |
|delhi    |C022599    |314625          |2   |
|delhi    |C018688    |306692          |3   |
|hyderabad|C032833    |318097          |1   |
|hyderabad|C023269    |292791          |2   |
|hyderabad|C013263    |291679          |3   |
|kolkata  |C032246    |304480          |1   |
|kolkata  |C022131    |296888          |2   |
|kolkata  |C028450    |296653          |3   |
|mumbai   |C048696    |334732          |1   |
|mumbai   |C047887    |307401          |2   |
|mumbai   |C022721    |306800          |3   |
|pune     |C002564    |315172          |1   |
|pune     |C023148    |310061          |2   |
|pune     |C032428    |305759          |3   |
+---------+-----------+----------------+----+
```

```
#task 4
from pyspark.sql import functions as F
from pyspark.sql.window import Window

base = segmented_customers_df if 'segmented_customers_df' in globals() else customer_profile_df

top10_overall = (
    base.withColumn(
            "rank",
            F.dense_rank().over(
                Window.orderBy(F.col("total_spend").desc(), F.col("order_count").desc())
            )
        )
        .filter("rank <= 10")
)

cols = ["customer_id", "total_spend", "order_count", "rank"]
if "customer_segment" in base.columns: cols.insert(3, "customer_segment")
```

```
top10_overall.select(*cols).orderBy("rank").show(10, truncate=False)
```

```
+-----------+-----------+-----------+----------------+----+
|customer_id|total_spend|order_count|customer_segment|rank|
+-----------+-----------+-----------+----------------+----+
|C043076    |493949     |6          |VIP             |1   |
|C034689    |486879     |6          |VIP             |2   |
|C039985    |484057     |6          |VIP             |3   |
|C026691    |477147     |6          |VIP             |4   |
|C038979    |477138     |6          |VIP             |5   |
|C020762    |474717     |6          |VIP             |6   |
|C044654    |471304     |6          |VIP             |7   |
|C014292    |468617     |6          |VIP             |8   |
|C019565    |467523     |6          |VIP             |9   |
|C045487    |467050     |6          |VIP             |10  |
+-----------+-----------+-----------+----------------+----+
```

PHASE 5

```
from pyspark.sql import functions as F

# Use optimized silver if present, else the plain silver
orders = optimized_clean_orders_df if 'optimized_clean_orders_df' in globals() else clean_orders_df

# 5.1) Loyal customers (≥ 3 distinct dates AND ≥ 2 distinct categories)
loyal = (
    orders.groupBy("customer_id")
          .agg(
              F.countDistinct("order_date").alias("distinct_purchase_dates"),
              F.countDistinct("category").alias("distinct_categories")
          )
          .withColumn(
              "is_loyal",
              (F.col("distinct_purchase_dates") >= 3) & (F.col("distinct_categories") >= 2)
          )
)

# 5.2) Loyal customers per city
loyal_per_city = (
    orders.join(loyal, "customer_id", "inner")
          .groupBy("city")
          .agg(F.countDistinct(F.when(F.col("is_loyal"), F.col("customer_id"))).alias("loyal_customers"))
          .orderBy(F.desc("loyal_customers"))
)

# 5.3) Revenue split: loyal vs non-loyal
loyal_revenue_split = (
    orders.join(loyal.select("customer_id","is_loyal"), "customer_id", "left")
          .withColumn("is_loyal", F.coalesce(F.col("is_loyal"), F.lit(False)))
          .groupBy("is_loyal")
          .agg(
              F.sum("amount").alias("revenue"),
              F.count("*").alias("orders")
          )
          .orderBy(F.desc("is_loyal"))
)

print("Phase 5 · Loyal customers (flag):")
loyal.orderBy(F.desc("is_loyal")).show(10, truncate=False)

print("Phase 5 · Loyal customers per city:")
loyal_per_city.show(truncate=False)

print("Phase 5 · Revenue split (loyal vs non-loyal):")
loyal_revenue_split.show(truncate=False)
```

```
Phase 5 · Loyal customers (flag):
+-----------+----------------------+-------------------+--------+
|customer_id|distinct_purchase_dates|distinct_categories|is_loyal|
+-----------+----------------------+-------------------+--------+
|C029861    |3                     |3                  |true    |
|C009248    |3                     |4                  |true    |
|C009707    |3                     |3                  |true    |
|C012569    |3                     |3                  |true    |
```

```
|C048710    |3                    |2                  |true    |
|C035805    |3                    |4                  |true    |
|C034669    |3                    |4                  |true    |
|C040253    |3                    |4                  |true    |
|C031105    |3                    |3                  |true    |
|C045645    |3                    |2                  |true    |
+-----------+---------------------+-------------------+--------+
only showing top 10 rows
Phase 5 · Loyal customers per city:
+---------+---------------+
|city     |loyal_customers|
+---------+---------------+
|hyderabad|26748          |
|delhi    |26635          |
|pune     |26601          |
|chennai  |26562          |
|kolkata  |26448          |
|mumbai   |26416          |
|bangalore|26393          |
+---------+---------------+

Phase 5 · Revenue split (loyal vs non-loyal):
+--------+-----------+------+
|is_loyal|revenue    |orders|
+--------+-----------+------+
|true    |11185423978|255342|
|false   |150209020  |3492  |
+--------+-----------+------+
```

PHASE 6

```
from pyspark.sql import functions as F
from pyspark.sql.window import Window

orders = optimized_clean_orders_df if 'optimized_clean_orders_df' in globals() else clean_orders_df

# T1) Monthly revenue per city
monthly_revenue_city = (
    orders.withColumn("month", F.date_trunc("month", "order_date"))
          .groupBy("city", "month")
          .agg(F.sum("amount").alias("monthly_revenue"))
)

# T2) Monthly order count per category
monthly_orders_category = (
    orders.withColumn("month", F.date_trunc("month", "order_date"))
          .groupBy("category", "month")
          .agg(F.count("*").alias("monthly_order_count"))
)

# T3) MoM trend for city revenue (absolute & % change)
w_city = Window.partitionBy("city").orderBy("month")
monthly_revenue_city_trend = (
    monthly_revenue_city
        .withColumn("prev_rev", F.lag("monthly_revenue").over(w_city))
        .withColumn("mom_change", F.col("monthly_revenue") - F.col("prev_rev"))
        .withColumn("mom_change_pct", F.round(F.col("mom_change") / F.col("prev_rev") * 100.0, 2))
)


print("Phase 6 · Monthly revenue per city:")
monthly_revenue_city.orderBy("city", "month").show(20, truncate=False)

print("Phase 6 · Monthly orders per category:")
monthly_orders_category.orderBy("category", "month").show(20, truncate=False)

print("Phase 6 · MoM trend (city revenue):")
monthly_revenue_city_trend.orderBy("city", "month").show(20, truncate=False)
```

```
Phase 6 · Monthly revenue per city:
+---------+-------------------+---------------+
|city     |month              |monthly_revenue|
+---------+-------------------+---------------+
|bangalore|2024-01-01 00:00:00|822339117      |
|bangalore|2024-02-01 00:00:00|792163305      |
|chennai  |2024-01-01 00:00:00|818567389      |
```

```
|chennai  |2024-02-01 00:00:00|796361427      |
|delhi    |2024-01-01 00:00:00|817332633      |
|delhi    |2024-02-01 00:00:00|805877007      |
|hyderabad|2024-01-01 00:00:00|833063605      |
|hyderabad|2024-02-01 00:00:00|796252807      |
|kolkata  |2024-01-01 00:00:00|824920456      |
|kolkata  |2024-02-01 00:00:00|785096186      |
|mumbai   |2024-01-01 00:00:00|816636150      |
|mumbai   |2024-02-01 00:00:00|795736235      |
|pune     |2024-01-01 00:00:00|833507124      |
|pune     |2024-02-01 00:00:00|797779557      |
+---------+-------------------+---------------+

Phase 6 · Monthly orders per category:
+-----------+-------------------+-------------------+
|category   |month              |monthly_order_count|
+-----------+-------------------+-------------------+
|electronics|2024-01-01 00:00:00|33063              |
|electronics|2024-02-01 00:00:00|31889              |
|fashion    |2024-01-01 00:00:00|32509              |
|fashion    |2024-02-01 00:00:00|31810              |
|grocery    |2024-01-01 00:00:00|32986              |
|grocery    |2024-02-01 00:00:00|31761              |
|home       |2024-01-01 00:00:00|33136              |
|home       |2024-02-01 00:00:00|31680              |
+-----------+-------------------+-------------------+

Phase 6 · MoM trend (city revenue):
+---------+-------------------+---------------+---------+----------+--------------+
|city     |month              |monthly_revenue|prev_rev |mom_change|mom_change_pct|
+---------+-------------------+---------------+---------+----------+--------------+
|bangalore|2024-01-01 00:00:00|822339117      |NULL     |NULL      |NULL          |
|bangalore|2024-02-01 00:00:00|792163305      |822339117|-30175812 |-3.67         |
|chennai  |2024-01-01 00:00:00|818567389      |NULL     |NULL      |NULL          |
|chennai  |2024-02-01 00:00:00|796361427      |818567389|-22205962 |-2.71         |
|delhi    |2024-01-01 00:00:00|817332633      |NULL     |NULL      |NULL          |
|delhi    |2024-02-01 00:00:00|805877007      |817332633|-11455626 |-1.4          |
|hyderabad|2024-01-01 00:00:00|833063605      |NULL     |NULL      |NULL          |
|hyderabad|2024-02-01 00:00:00|796252807      |833063605|-36810798 |-4.42         |
|kolkata  |2024-01-01 00:00:00|824920456      |NULL     |NULL      |NULL          |
|kolkata  |2024-02-01 00:00:00|785096186      |824920456|-39824270 |-4.83         |
|mumbai   |2024-01-01 00:00:00|816636150      |NULL     |NULL      |NULL          |
|mumbai   |2024-02-01 00:00:00|795736235      |816636150|-20899915 |-2.56         |
|pune     |2024-01-01 00:00:00|833507124      |NULL     |NULL      |NULL          |
|pune     |2024-02-01 00:00:00|797779557      |833507124|-35727567 |-4.29         |
+---------+-------------------+---------------+---------+----------+--------------+
```

PHASE 7

```
#t1
from pyspark.sql import functions as F
from pyspark.sql.window import Window

orders = optimized_clean_orders_df if 'optimized_clean_orders_df' in globals() else clean_orders_df

if 'customer_profile_df' not in globals():
    customer_profile_df = (
        orders.groupBy("customer_id")
              .agg(
                  F.count("*").alias("order_count"),
                  F.sum("amount").alias("total_spend"),
                  (F.sum("amount")/F.count("*")).alias("avg_order_value"),
                  F.min("order_date").alias("first_purchase_date"),
                  F.max("order_date").alias("last_purchase_date"),
                  F.countDistinct("city").alias("distinct_cities"),
                  F.countDistinct("category").alias("distinct_categories"),
              )
    )

if 'cust_city_rank_df' not in globals():
    cust_city_rank_df = (
        orders.groupBy("city","customer_id")
              .agg(F.sum("amount").alias("city_total_spend"),
                   F.count("*").alias("city_order_count"))
              .withColumn("spend_rank_in_city",
                          F.dense_rank().over(
                              Window.partitionBy("city")
                                    .orderBy(F.col("city_total_spend").desc(),
                                             F.col("city_order_count").desc())
```

```
                            ))
    )

print("Reused DFs:", ["orders (silver)", "customer_profile_df", "cust_city_rank_df"])
```

```
Reused DFs: ['orders (silver)', 'customer_profile_df', 'cust_city_rank_df']
```

```
#t2
from pyspark import StorageLevel


orders = orders.persist(StorageLevel.MEMORY_AND_DISK)
customer_profile_df = customer_profile_df.persist(StorageLevel.MEMORY_AND_DISK)
cust_city_rank_df = cust_city_rank_df.persist(StorageLevel.MEMORY_AND_DISK)

_ = orders.count(), customer_profile_df.count(), cust_city_rank_df.count()
print("Cached: orders, customer_profile_df, cust_city_rank_df")
```

```
Cached: orders, customer_profile_df, cust_city_rank_df
```

```
#t3
print("\n== Explain: Customer aggregation ==")
customer_profile_df.explain(True)

print("\n== Explain: Per-city window ranking ==")
cust_city_rank_df.explain(True)
```

```
                                        +- SortAggregate(key=[order_id#71], functions=[first(customer_id#72, false), first(city#73
```

```
                                          +- Sort [order_id#71 ASC NULLS FIRST], false, 0
                                             +- Exchange hashpartitioning(order_id#71, 200), ENSURE_REQUIREMENTS, [plan_id=1634]
                                                +- SortAggregate(key=[order_id#71], functions=[partial_first(customer_id#72, false
                                                   +- Sort [order_id#71 ASC NULLS FIRST], false, 0
                                                      +- Project [order_id#71, customer_id#72, city#73, category#74, product#75,
                                                         +- Project [trim(order_id#17, None) AS order_id#71, trim(customer_id#18,
                                                            +- FileScan csv [order_id#17,customer_id#18,city#19,category#20,produc
```

```
#t4


print("\nShuffle indicators:")
print("- customer_profile_df came from groupBy('customer_id') → shuffle")
print("- cust_city_rank_df came from groupBy('city','customer_id') + window(partitionBy 'city') → shuffle")
print("- Any joins without broadcast on one side → shuffle")


print("spark.sql.shuffle.partitions =", spark.conf.get("spark.sql.shuffle.partitions"))
```

```
Shuffle indicators:
- customer_profile_df came from groupBy('customer_id') → shuffle
- cust_city_rank_df came from groupBy('city','customer_id') + window(partitionBy 'city') → shuffle
- Any joins without broadcast on one side → shuffle
spark.sql.shuffle.partitions = 200
```

```
#t5

customer_profile_df = customer_profile_df.repartition("customer_id")

monthly_revenue_city = (
    orders.withColumn("month", F.date_trunc("month","order_date"))
          .groupBy("city","month")
          .agg(F.sum("amount").alias("monthly_revenue"))
          .repartition("city","month")
          .persist(StorageLevel.MEMORY_AND_DISK)
)
_ = monthly_revenue_city.count()

print("Repartitioned: customer_profile_df by 'customer_id'; monthly_revenue_city by ('city','month')")
print("customer_profile_df partitions:", customer_profile_df.rdd.getNumPartitions())
print("monthly_revenue_city partitions:", monthly_revenue_city.rdd.getNumPartitions())
```

```
Repartitioned: customer_profile_df by 'customer_id'; monthly_revenue_city by ('city','month')
customer_profile_df partitions: 200
monthly_revenue_city partitions: 200
```

PHASE 8

```
from pyspark.sql import functions as F

# T1) Small lookup dimension
segment_lookup = spark.createDataFrame(
    [(1, "VIP"), (2, "Premium"), (3, "Regular")],
    ["segment_code", "segment_label"]
)

# T2) Broadcast join to avoid shuffle
segmented_with_code = (
    segmented_customers_df.alias("c")
    .join(
        F.broadcast(segment_lookup.alias("l")),
        F.col("c.customer_segment") == F.col("l.segment_label"),
        "left"
    )
    .select("c.*", "segment_code")
)

# T3) Verify BroadcastHashJoin in the plan
print("== Phase 8: Explain plan (look for BroadcastHashJoin) ==")
segmented_with_code.explain(True)
```

```
# T4) Quick peek
segmented_with_code.select(
    "customer_id", "total_spend", "order_count", "customer_segment", "segment_code"
).orderBy(F.desc("total_spend")).show(10, truncate=False)
```

```
               +- == Initial Plan ==
                  HashAggregate(keys=[customer_id#72], functions=[first(count(1)#2644L, true), first(sum(amount)#2646L, true),
                  +- Exchange hashpartitioning(customer_id#72, 200), ENSURE_REQUIREMENTS, [plan_id=2091]
                     +- HashAggregate(keys=[customer_id#72], functions=[partial_first(count(1)#2644L, true) FILTER (WHERE (gid#
                        +- HashAggregate(keys=[customer_id#72, city#2640, category#2641, gid#2639], functions=[count(1), sum(ar
                           +- Exchange hashpartitioning(customer_id#72, city#2640, category#2641, gid#2639, 200), ENSURE_REQUI
                              +- HashAggregate(keys=[customer_id#72, city#2640, category#2641, gid#2639], functions=[partial_co
                                 +- Expand [[customer_id#72, null, null, 0, amount#569, order_date#570], [customer_id#72, city#
                                    +- InMemoryTableScan [customer_id#72, city#73, category#74, amount#569, order_date#570]
                                          +- InMemoryRelation [order_id#71, customer_id#72, city#73, category#74, product#75, a
                                                +- AdaptiveSparkPlan isFinalPlan=true
                                 +- == Final Plan ==
                                    ResultQueryStage 2
                                    +- ShuffleQueryStage 1
                                       +- Exchange hashpartitioning(customer_id#791, 200), REPARTITION_BY_COL, [plan_id=1741]
                                          +- *(3) Project [order_id#71, customer_id#791, city#793, category#795, product#797, a
                                             +- *(3) Filter (isnotnull(status#803) AND ((status#803 = completed) AND (isnotnul
                                                +- SortAggregate(key=[order_id#71], functions=[first(customer_id#72, false), f
                                                   +- *(2) Sort [order_id#71 ASC NULLS FIRST], false, 0
                                                      +- AQEShuffleRead coalesced
                                                         +- ShuffleQueryStage 0
                                                            +- Exchange hashpartitioning(order_id#71, 200), ENSURE_REQUIREMENTS
                                                               +- SortAggregate(key=[order_id#71], functions=[partial_first(cu
                                                                  +- *(1) Sort [order_id#71 ASC NULLS FIRST], false, 0
                                                                     +- *(1) Project [order_id#71, customer_id#72, city#73, cat
                                                                        +- *(1) Project [trim(order_id#17, None) AS order_id#7
                                                                           +- FileScan csv [order_id#17,customer_id#18,city#19
                                 +- == Initial Plan ==
                                    Exchange hashpartitioning(customer_id#791, 200), REPARTITION_BY_COL, [plan_id=1639]
                                    +- Project [order_id#71, customer_id#791, city#793, category#795, product#797, amount_clean
                                       +- Filter (isnotnull(status#803) AND ((status#803 = completed) AND (isnotnull(amount_cle
                                          +- SortAggregate(key=[order_id#71], functions=[first(customer_id#72, false), first(c
                                             +- Sort [order_id#71 ASC NULLS FIRST], false, 0
                                                +- Exchange hashpartitioning(order_id#71, 200), ENSURE_REQUIREMENTS, [plan_id=1
                                                   +- SortAggregate(key=[order_id#71], functions=[partial_first(customer_id#72
                                                      +- Sort [order_id#71 ASC NULLS FIRST], false, 0
                                                         +- Project [order_id#71, customer_id#72, city#73, category#74, product
                                                            +- Project [trim(order_id#17, None) AS order_id#71, trim(customer_
                                                               +- FileScan csv [order_id#17,customer_id#18,city#19,category#20
      +- BroadcastExchange HashedRelationBroadcastMode(List(input[1, string, false]),false), [plan_id=7150]
         +- Filter isnotnull(segment_label#12342)
            +- Scan ExistingRDD[segment_code#12341L,segment_label#12342]

+-----------+-----------+-----------+----------------+------------+
|customer_id|total_spend|order_count|customer_segment|segment_code|
+-----------+-----------+-----------+----------------+------------+
|C043076    |493949     |6          |VIP             |1           |
|C034689    |486879     |6          |VIP             |1           |
|C039985    |484057     |6          |VIP             |1           |
|C026691    |477147     |6          |VIP             |1           |
|C038979    |477138     |6          |VIP             |1           |
|C020762    |474717     |6          |VIP             |1           |
|C044654    |471304     |6          |VIP             |1           |
|C014292    |468617     |6          |VIP             |1           |
|C019565    |467523     |6          |VIP             |1           |
|C045487    |467050     |6          |VIP             |1           |
+-----------+-----------+-----------+----------------+------------+
only showing top 10 rows
```

PHASE 9

```
#T1
from pyspark.sql import functions as F

base = segmented_customers_df if 'segmented_customers_df' in globals() else customer_profile_df

sorted_customers = base.orderBy(F.col("total_spend").desc(), F.col("order_count").desc())


sorted_cols = ["customer_id", "total_spend", "order_count"]
if "customer_segment" in base.columns: sorted_cols.insert(3, "customer_segment")
sorted_customers.select(*sorted_cols).show(20, truncate=False)
```

```
+-----------+-----------+-----------+----------------+
|customer_id|total_spend|order_count|customer_segment|
+-----------+-----------+-----------+----------------+
|C043076    |493949     |6          |VIP             |
|C034689    |486879     |6          |VIP             |
|C039985    |484057     |6          |VIP             |
|C026691    |477147     |6          |VIP             |
|C038979    |477138     |6          |VIP             |
|C020762    |474717     |6          |VIP             |
|C044654    |471304     |6          |VIP             |
|C014292    |468617     |6          |VIP             |
|C019565    |467523     |6          |VIP             |
|C045487    |467050     |6          |VIP             |
|C046747    |464951     |6          |VIP             |
|C004490    |463923     |6          |VIP             |
|C038296    |463147     |6          |VIP             |
|C005286    |463098     |6          |VIP             |
|C022754    |462987     |6          |VIP             |
|C047145    |458602     |6          |VIP             |
|C026261    |458067     |6          |VIP             |
|C029753    |457258     |6          |VIP             |
|C037602    |456932     |6          |VIP             |
|C017673    |454569     |6          |VIP             |
+-----------+-----------+-----------+----------------+
only showing top 20 rows
```

```
#T2

orders = optimized_clean_orders_df if 'optimized_clean_orders_df' in globals() else clean_orders_df

electronics_buyers = orders.filter(F.col("category") == "electronics").select("customer_id").distinct()
grocery_buyers     = orders.filter(F.col("category") == "grocery").select("customer_id").distinct()
both_sets = electronics_buyers.join(grocery_buyers, on="customer_id", how="inner")
only_one_set = (
    electronics_buyers.join(grocery_buyers, on="customer_id", how="left_anti")
    .unionByName(
        grocery_buyers.join(electronics_buyers, on="customer_id", how="left_anti")
    )
)

print("Phase 9 · Customers in BOTH (Electronics ∩ Grocery):")
both_sets.show(20, truncate=False)

print("Phase 9 · Customers in ONLY ONE (Electronics ⊕ Grocery):")
only_one_set.show(20, truncate=False)
```

```
Phase 9 · Customers in BOTH (Electronics ∩ Grocery):
+-----------+
|customer_id|
+-----------+
|C000142    |
|C000299    |
|C000433    |
|C000884    |
|C001019    |
|C001154    |
|C001875    |
|C002484    |
|C002512    |
|C003053    |
|C003128    |
|C003194    |
|C003563    |
|C004242    |
|C004713    |
|C004804    |
|C005049    |
|C005119    |
|C005311    |
|C005519    |
+-----------+
only showing top 20 rows
Phase 9 · Customers in ONLY ONE (Electronics ⊕ Grocery):
+-----------+
|customer_id|
+-----------+
|C003484    |
|C004744    |
|C005781    |
|C006324    |
```

```
|C006738    |
|C007013    |
|C007678    |
|C009707    |
|C010142    |
|C011458    |
|C012341    |
|C012535    |
|C013159    |
|C017496    |
|C017757    |
|C017910    |
|C018622    |
|C020861    |
|C021165    |
|C022046    |
+-----------+
only showing top 20 rows
```

PHASE 10

```
out_root = "/tmp/gold_out"   # <-- change to your ADLS/S3/DBFS/local path
```

```
#T1
from pyspark.sql import functions as F


if 'segmented_customers_df' in globals():
    customer_master = segmented_customers_df
elif 'segmented_with_code' in globals():
    customer_master = segmented_with_code
else:

    customer_master = (
        customer_profile_df
        .withColumn(
            "customer_segment",
            F.when((F.col("total_spend") >= 200_000) & (F.col("order_count") >= 5), "VIP")
             .when(F.col("total_spend") >= 100_000, "Premium")
             .otherwise("Regular")
        )
    )

(customer_master
 .write.mode("overwrite")
 .partitionBy("customer_segment")
 .parquet(f"{out_root}/customer_master_parquet"))

print("✓ Wrote: customer_master_parquet (partitioned by customer_segment)")
```

```
✓ Wrote: customer_master_parquet (partitioned by customer_segment)
```

```
#T2

from pyspark.sql import functions as F

orders = optimized_clean_orders_df if 'optimized_clean_orders_df' in globals() else clean_orders_df

monthly_revenue_city = (
    orders.withColumn("month", F.date_trunc("month", "order_date"))
          .groupBy("city", "month")
          .agg(F.sum("amount").alias("monthly_revenue"))
)

monthly_orders_category = (
    orders.withColumn("month", F.date_trunc("month", "order_date"))
          .groupBy("category", "month")
          .agg(F.count("*").alias("monthly_order_count"))
)

# Write as ORC
monthly_base = f"{out_root}/monthly_analytics_orc"
monthly_revenue_city.write.mode("overwrite").orc(f"{monthly_base}/monthly_revenue_city")
monthly_orders_category.write.mode("overwrite").orc(f"{monthly_base}/monthly_orders_by_category")

print("✓ Wrote: monthly_revenue_city (ORC)")
```

```
print(" Wrote: monthly_revenue_city (ORC)")
print(" Wrote: monthly_orders_by_category (ORC)")
```

```
 Wrote: monthly_revenue_city (ORC)
 Wrote: monthly_orders_by_category (ORC)
```

```
#T3

cm = spark.read.parquet(f"{out_root}/customer_master_parquet")
print("Customer master — rows:", cm.count())
print("Customer master — partitions:", cm.rdd.getNumPartitions())

mrc = spark.read.orc(f"{out_root}/monthly_analytics_orc/monthly_revenue_city")
moc = spark.read.orc(f"{out_root}/monthly_analytics_orc/monthly_orders_by_category")
print("Monthly revenue (rows):", mrc.count())
print("Monthly orders by category (rows):", moc.count())
```

```
Customer master — rows: 47500
Customer master — partitions: 19
Monthly revenue (rows): 14
Monthly orders by category (rows): 8
```

PHASE 11

T 1: WHY DANGEROUS ? DataFrame.show() returns None, it doesn't return a DataFrame. Assigning its result to df discards the DataFrame and replaces it with None.

```
agg_df = clean_orders_df.groupBy("customer_id").agg(F.sum("amount").alias("sum_amount"))

agg_df.show(10, truncate=False)


top_customers = agg_df.orderBy(F.col("sum_amount").desc()).limit(10)
top_customers.show(truncate=False)
```

```
+-----------+----------+
|customer_id|sum_amount|
+-----------+----------+
|C000142    |313288    |
|C000299    |228261    |
|C000433    |285507    |
|C001115    |163614    |
|C001875    |213381    |
|C002484    |118291    |
|C002512    |336838    |
|C002837    |225248    |
|C003194    |227384    |
|C003484    |300712    |
+-----------+----------+
only showing top 10 rows
+-----------+----------+
|customer_id|sum_amount|
+-----------+----------+
|C043076    |493949    |
|C034689    |486879    |
|C039985    |484057    |
|C026691    |477147    |
|C038979    |477138    |
|C020762    |474717    |
|C044654    |471304    |
|C014292    |468617    |
|C019565    |467523    |
|C045487    |467050    |
+-----------+----------+
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```