Taller Práctico: Modularización de un Pipeline con Datos Reales (Olist)

Objetivo: Transformar el pipeline Medallion a una serie de notebooks modulares, utilizando datos reales del dataset Olist y un framework de validación de calidad explícito, listos para ser orquestados como un job de producción en Databricks.

Estructura de Notebooks Propuesta

El proceso se dividirá en los siguientes notebooks, reflejando las mejores prácticas de un proyecto de ingeniería de datos:

1. 00_Setup_Environment: Prepara el catálogo y los volúmenes, y guía al participante para cargar los datos de Olist.
2. 01_Bronze_Ingestion: Carga los datos CSV desde el volumen a las tablas de la capa Bronce.
3. 02_DataQuality_Validation: Ejecuta los guardianes de calidad sobre los datos de Bronce usando la función de validación.
4. 03_Silver_Transformation: Transforma los datos validados de Bronce a un modelo dimensional en la capa Plata.
5. 04_Gold_Aggregation: Crea las tablas de negocio agregadas en la capa Oro.

Contenido de cada Notebook

Notebook 0: 00_Setup_Environment

Propósito: Este notebook se ejecuta una sola vez para preparar toda la infraestructura. Explica por qué se separa el trabajo en un catálogo dedicado.

```
# Celda 1: Definir la configuración del catálogo y esquemas
# Usar variables permite cambiar fácilmente los nombres en un solo lugar.
catalog_name = "sesion_5"
bronze_schema = "bronze"
silver_schema = "silver"
gold_schema = "gold"
audit_schema = "auditoria"

# Celda 2: Crear la infraestructura en Unity Catalog
# Un catálogo aísla el proyecto de otros en el workspace.
spark.sql(f"CREATE CATALOG IF NOT EXISTS {catalog_name}")
spark.sql(f"USE CATALOG {catalog_name}")
# Los esquemas organizan las tablas por capa (bronze, silver, gold).
spark.sql(f"CREATE SCHEMA IF NOT EXISTS {bronze_schema}")
```

```
spark.sql(f"CREATE SCHEMA IF NOT EXISTS {silver_schema}")
spark.sql(f"CREATE SCHEMA IF NOT EXISTS {gold_schema}")
spark.sql(f"CREATE SCHEMA IF NOT EXISTS {audit_schema}")

# Celda 3: Crear el volumen para los archivos crudos
# Los volúmenes dan acceso a nivel de sistema de archivos dentro de Unity Catalog.
bronze_volume_path = f"/Volumes/{catalog_name}/{bronze_schema}/raw_files"
spark.sql(f"CREATE VOLUME IF NOT EXISTS {catalog_name}.{bronze_schema}.raw_files")

print("Catálogo, esquemas y volumen creados exitosamente.")
print(f"Por favor, cargue los archivos CSV del dataset Olist en el siguiente volumen:
{bronze_volume_path}")
```

Instrucción para el Participante:

1. Descargar el dataset "Brazilian E-Commerce Public Dataset by Olist" desde Kaggle.
2. Descomprimir el archivo ZIP.
3. En la interfaz de Databricks, navegar a Catalog, hasta sesion_5 -> bronze -> raw_files.
4. Subir todos los archivos .csv del dataset a este volumen.

Notebook 1: 01_Bronze_Ingestion

Propósito: Carga los archivos CSV desde el volumen a las tablas Delta de la capa Bronce. Esta es la primera tarea del job recurrente.

```
# Celda 1: Definir configuración
catalog_name = "sesion_5"
bronze_schema = "bronze"
bronze_volume_path = f"/Volumes/{catalog_name}/{bronze_schema}/raw_files"

# Celda 2: Función de ayuda para cargar y escribir tablas
def ingest_csv_to_bronze(file_name, table_name):
    """
    Lee un archivo CSV desde el volumen de Bronce y lo guarda como una tabla Delta.
    Se usa inferSchema=true para este ejercicio, pero en producción es mejor definir el esquema
explícitamente.
    """
    df = spark.read.option("header", "true").option("inferSchema",
"true").csv(f"{bronze_volume_path}/{file_name}")

df.write.format("delta").mode("overwrite").saveAsTable(f"{catalog_name}.{bronze_schema}.{tabl
e_name}")
    print(f"Tabla '{table_name}' creada/actualizada en la capa Bronce.")
```

```
# Celda 3: Ejecutar la ingesta para los archivos principales
ingest_csv_to_bronze("olist_customers_dataset.csv", "customers_bronze")
ingest_csv_to_bronze("olist_order_items_dataset.csv", "order_items_bronze")
ingest_csv_to_bronze("olist_order_payments_dataset.csv", "order_payments_bronze")
ingest_csv_to_bronze("olist_orders_dataset.csv", "orders_bronze")
ingest_csv_to_bronze("olist_products_dataset.csv", "products_bronze")
```

🏆 Desafío: Completar el notebook añadiendo el código en la Celda 3 para ingestar los archivos olist_sellers_dataset.csv y product_category_name_translation.csv.

Notebook 2: 02_DataQuality_Validation

Propósito: Ejecuta los guardianes de calidad. Este es un paso crítico que asegura la fiabilidad del pipeline. Si alguna validación falla, el job se detendrá con un error claro.

```
# Celda 1: Función de ayuda para validaciones de calidad
def ejecutar_validacion_calidad(query, descripcion_validacion):
  """
  Ejecuta una consulta SQL que se espera que devuelva 0.
  Si devuelve un valor mayor a 0, lanza una excepción para detener el pipeline.
  """
  print(f"Ejecutando validación: '{descripcion_validacion}'...")
  df_resultado = spark.sql(query)
  conteo_filas_malas = df_resultado.first()[0]

  if conteo_filas_malas > 0:
    raise Exception(f"VALIDACIÓN FALLIDA: {descripcion_validacion}. Se encontraron
{conteo_filas_malas} filas que violan la regla.")
  else:
    print(f"VALIDACIÓN EXITOSA: {descripcion_validacion}. No se encontraron problemas. ✔️")

# Celda 2: Crear vistas temporales para facilitar las consultas
spark.sql("CREATE OR REPLACE TEMP VIEW customers_bronze_vw AS SELECT * FROM
sesion_5.bronze.customers_bronze")
spark.sql("CREATE OR REPLACE TEMP VIEW order_items_bronze_vw AS SELECT * FROM
sesion_5.bronze.order_items_bronze")
spark.sql("CREATE OR REPLACE TEMP VIEW orders_bronze_vw AS SELECT * FROM
sesion_5.bronze.orders_bronze")

# Celda 3: Ejecutar las reglas de negocio
# REGLA 1: La llave primaria de un cliente NUNCA debe ser nula.
query_pk_nula_cliente = "SELECT COUNT(*) FROM customers_bronze_vw WHERE customer_id
IS NULL"
```

```
ejecutar_validacion_calidad(query_pk_nula_cliente, "La columna 'customer_id' en la tabla de clientes no debe contener nulos.")

# REGLA 2: El precio de un item NUNCA debe ser negativo.
query_precio_negativo = "SELECT COUNT(*) FROM order_items_bronze_vw WHERE price < 0"
ejecutar_validacion_calidad(query_precio_negativo, "La columna 'price' en la tabla de items no debe contener valores negativos.")

# REGLA 3: No debe haber pedidos sin al menos un item.
query_pedidos_sin_items = """
  SELECT COUNT(o.order_id)
  FROM orders_bronze_vw o
  LEFT JOIN order_items_bronze_vw i ON o.order_id = i.order_id
  WHERE i.order_id IS NULL
"""
ejecutar_validacion_calidad(query_pedidos_sin_items, "Existen pedidos que no tienen ningún item asociado.")
```

🤔 Pregunta para el Participante: La REGLA 3 fallará (hay pedidos sin items en el dataset). ¿Qué implicaciones tiene esto para el JOIN en la capa Plata? ¿Por qué un INNER JOIN es una buena estrategia para manejar implícitamente este problema de calidad?

Notebook 3: 03_Silver_Transformation

Propósito: Crear la capa Plata, transformando los datos validados en un modelo dimensional limpio y optimizado.

```
-- Celda 1: Crear y usar el esquema Silver
USE CATALOG sesion_5;
CREATE SCHEMA IF NOT EXISTS silver;
USE silver;

-- Celda 2: Crear Dimensiones (dim_customers, dim_products)
-- Aquí se convierten los datos crudos en entidades de negocio claras.
CREATE OR REPLACE TABLE dim_customers AS
SELECT
  customer_id,
  customer_unique_id,
  customer_zip_code_prefix,
  customer_city,
  customer_state
FROM sesion_5.bronze.customers_bronze;
```

```
CREATE OR REPLACE TABLE dim_products AS
SELECT
  p.product_id,
  p.product_category_name,
  -- Se usa la tabla de traducción para enriquecer los datos
  t.product_category_name_english AS product_category_name_en,
  p.product_weight_g,
  p.product_length_cm,
  p.product_height_cm,
  p.product_width_cm
FROM sesion_5.bronze.products_bronze p
LEFT JOIN sesion_5.bronze.category_translation_bronze t ON p.product_category_name =
t.product_category_name;

-- Celda 3: Crear Tabla de Hechos (fact_orders)
-- Esta es la tabla central del modelo. Se particiona por año para optimizar consultas.
CREATE OR REPLACE TABLE fact_orders
PARTITIONED BY (order_purchase_year)
AS
SELECT
  o.order_id,
  o.customer_id,
  i.product_id,
  i.seller_id,
  CAST(o.order_purchase_timestamp AS DATE) AS order_purchase_date,
  YEAR(o.order_purchase_timestamp) AS order_purchase_year,
  o.order_status,
  i.price,
  i.freight_value,
  p.payment_type,
  p.payment_installments,
  p.payment_value
FROM sesion_5.bronze.orders_bronze o
JOIN sesion_5.bronze.order_items_bronze i ON o.order_id = i.order_id
JOIN sesion_5.bronze.order_payments_bronze p ON o.order_id = p.order_id;
```

🏆 Desafío: La tabla fact_orders tiene price y freight_value (valor del flete) separados. Se debe modificar la consulta de creación de fact_orders para añadir una nueva columna llamada total_value que sea la suma de ambas.

Notebook 4: 04_Gold_Aggregation

Propósito: Construir tablas agregadas, pre-calculadas, para que los análisis y dashboards sean extremadamente rápidos.

```
-- Celda 1: Crear y usar el esquema Gold
USE CATALOG sesion_5;
CREATE SCHEMA IF NOT EXISTS gold;
USE gold;

-- Celda 2: Crear tabla de ventas mensuales por categoría
-- Esta tabla responde directamente a una pregunta de negocio: ¿Cómo se comportan las ventas de cada categoría a lo largo del tiempo?
CREATE OR REPLACE TABLE monthly_sales_by_category_gold AS
SELECT
  YEAR(fo.order_purchase_date) AS anio,
  MONTH(fo.order_purchase_date) AS mes,
  dp.product_category_name_en AS categoria_producto,
  COUNT(DISTINCT fo.order_id) AS numero_pedidos,
  SUM(fo.payment_value) AS ingresos_totales
FROM silver.fact_orders fo
JOIN silver.dim_products dp ON fo.product_id = dp.product_id
WHERE fo.order_status = 'delivered' -- Solo se cuentan pedidos entregados
GROUP BY anio, mes, categoria_producto;
```

🏆 Desafío: Crear una nueva tabla Gold llamada customer_lifetime_value_gold que calcule, para cada customer_unique_id, el gasto total y el número total de pedidos que ha realizado.

Guía de Orquestación con Databricks Jobs

El flujo del job orquesta estos notebooks en una secuencia lógica con dependencias.

1. Tarea 1: 1_Ingesta_Bronce (Notebook: 01_Bronze_Ingestion)
2. Tarea 2: 2_Validacion_Calidad (Notebook: 02_DataQuality_Validation, depende de la Tarea 1)
3. Tarea 3: 3_Transformacion_Plata (Notebook: 03_Silver_Transformation, depende de la Tarea 2)
4. Tarea 4: 4_Agregacion_Oro (Notebook: 04_Gold_Aggregation, depende de la Tarea 3)

Visualización del Job en Databricks

Una vez configurado, el job se verá así en la interfaz de Workflows, mostrando claramente el flujo de dependencias:

Este diseño asegura que el pipeline se detenga si se detectan problemas de calidad y que cada etapa del proceso esté claramente definida y separada, listo para una ejecución de producción.