RETO TECNICO INGENIERIA DE DATOS PRAGMA

Autor: *Germán Homero Morán Figueroa*

Teniendo en cuenta la descripción del reto técnico se realizan algunas pruebas para verificar la funcionalidad del script

1. Datos

Se descarga los archivos fuente y se almacenan dentro de la carpeta datos

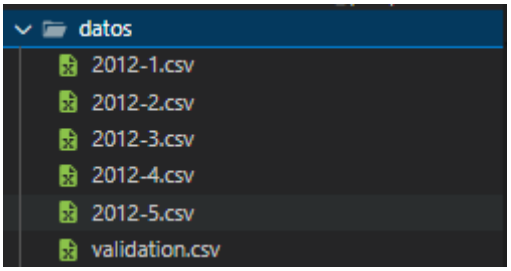

Figura 1.Archivos fuente

2. Funciones principales

El script final.py contiene toda la lógica que permite realizar la ingesta de los archivos (.csv) en pequeños micro-batch. A continuación, se describen los principales argumentos y parámetros para correr el script.

- **python final.py --chunksize 100:** Permite realizar la ingesta de los archivos .csv excluyendo el archivo de validación.csv. El parámetro chunksize, permite definir el tamaño del lote (batchsize) a ingestar, en este caso de prueba es 100 pero puede cambiar.
- **python final.py --chunksize 100 –validation:** El argumento **–validation** como su nombre lo indica permite realizar la ingesta específica del archivo de validación
- **python final.py --truncate-tables:** Permite eliminar todos los registros almacenados en las diferentes tablas.

Una vez ejecutado el primer comando ***python final.py --chunksize 100*** se observa que los datos cargan de manera exitosa dentro de la base de datos. En este caso se observa la carga incremental para cada archivo, teniendo en cuenta el tamaño del lote (chunksize). Es importante resaltar que hay algunos registros en los cuales el campo **price** es nulo, por tal motivo estos registros se omiten dentro del proceso de ingesta. Adicionalmente a medida que se termina de ingestar el lote (batch) se actualizan las estadísticas asociadas a ese procesamiento (Numero de filas (**cnt**), suma campo Price (**ssum**) , valor mínimo del campo Price(**smin**) , valor máximo del campo Price(**smax**) . Estas estadísticas son incrementales se van actualizando a medida que se ingesta y procesas cada batch. Al final de la ingesta de cada archivo se muestran las estadísticas en consola.

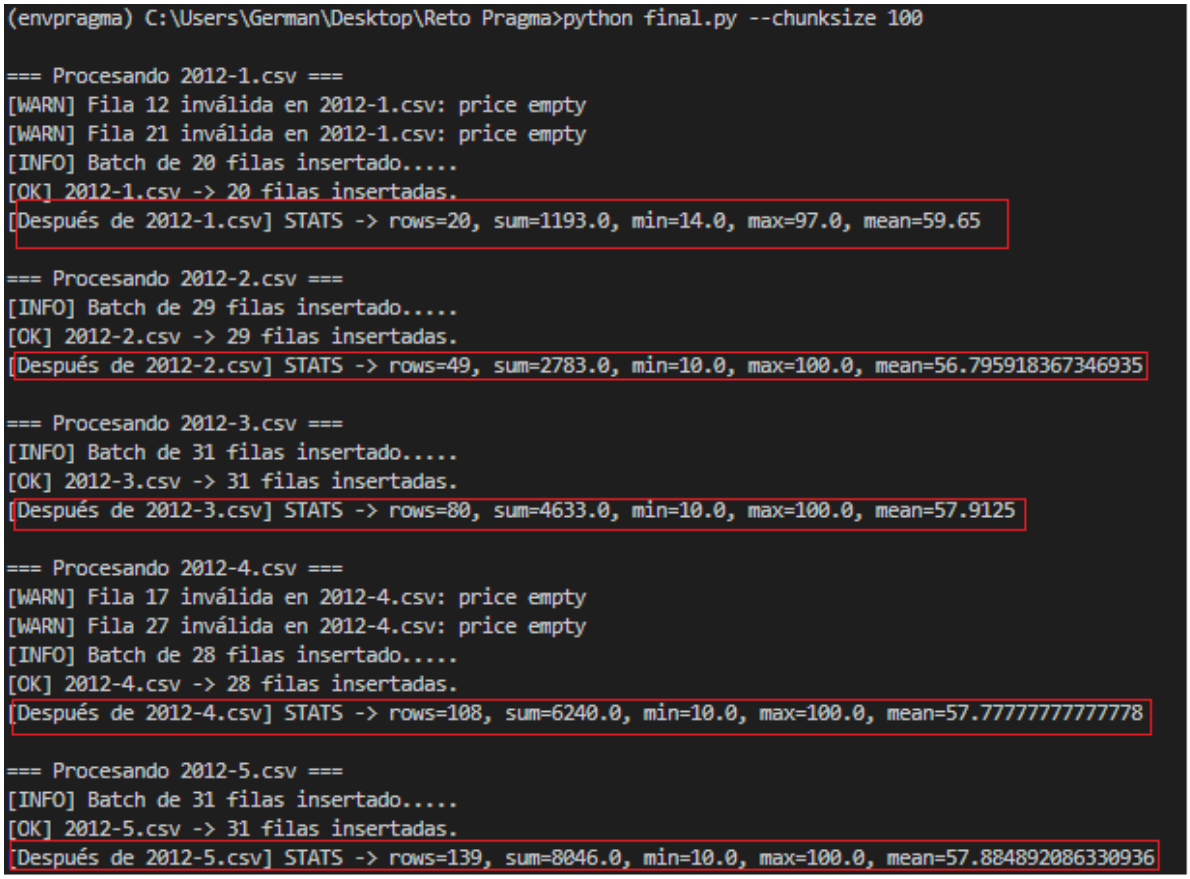

Figura 2.Estadisticas asociadas al procesamiento de cada uno de los archivos

3. Configuración de Base de datos

Para el desarrollo de este reto se definieron 3 tablas que se describen a continuación:

- **transactions:** Esta tabla guarda todos los registros(filas) asociados a cada uno de los archivos de ingesta (archivos.csv). El esquema de esta tabla tiene las mismas columnas de los archivos originales (***timestamp, price ,user_id***) , adicionalmente se agregaron 2 nuevas columnas denominadas (***Id***) que actúa como llave primaria , permitiendo identificar cada registro como único y la columna (***source_file***), que permite verificar el origen del registro cargado. En el ejemplo de la Figura 3 se observa los datos cargados para los 5 archivos CSV.

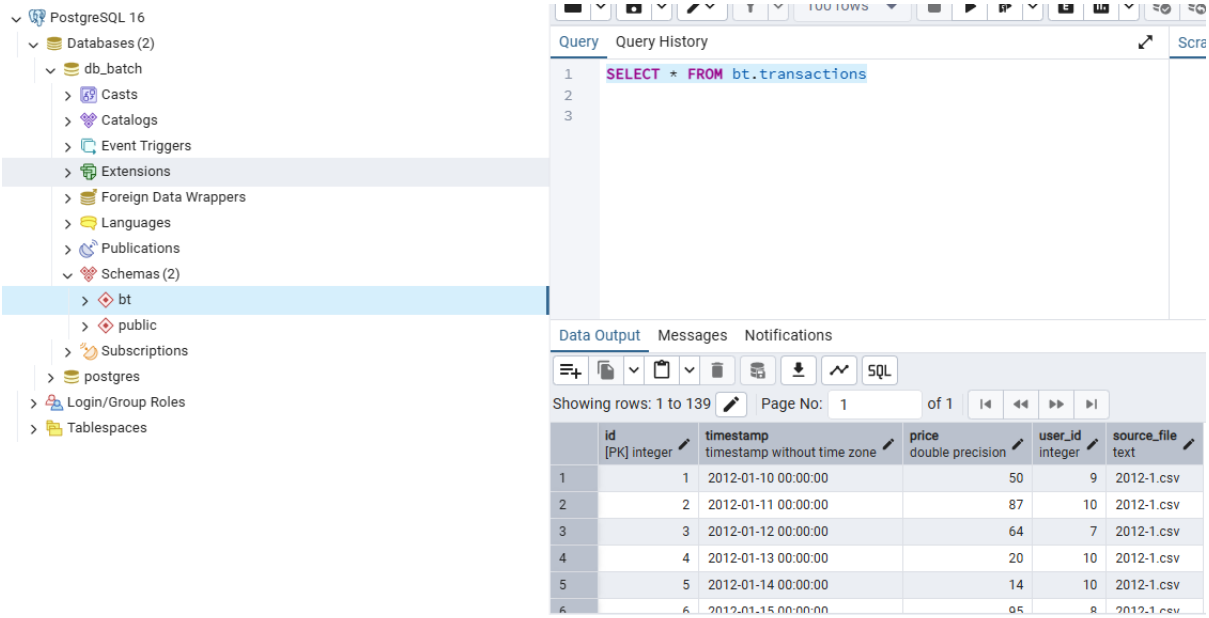

Figura 3. Esquema tabla transactions

- **ingestion_log:** Esta tabla permite almacenar un histórico de los archivos procesados como también del número de registros cargados por cada archivo, tal como se observa en la Figura 4 .

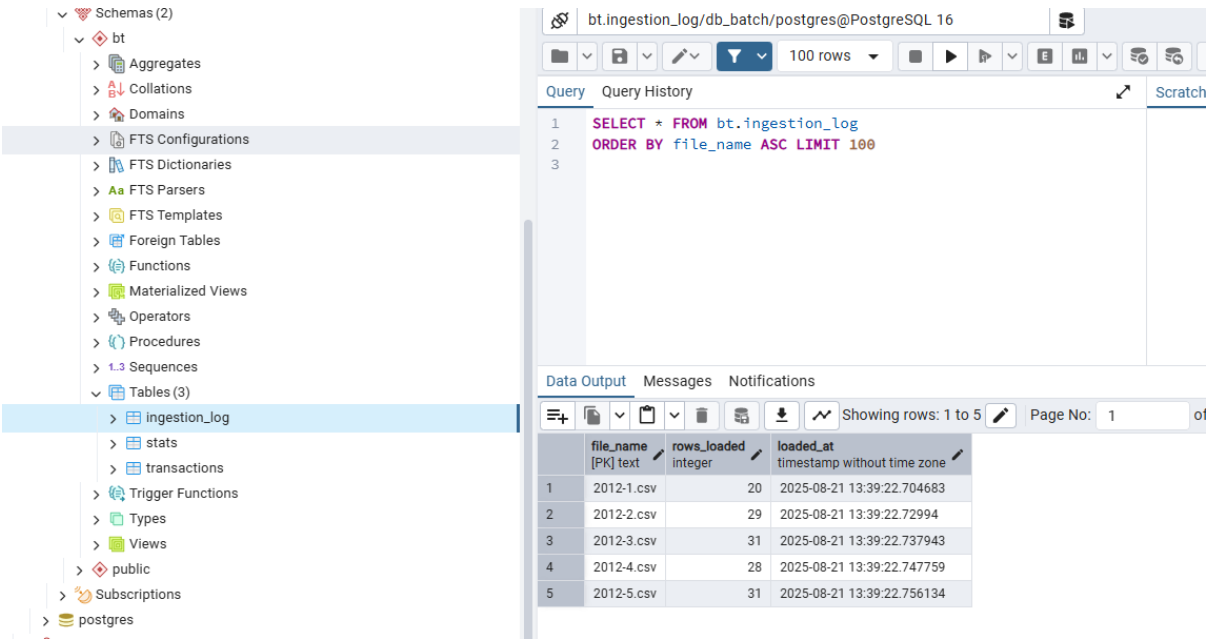

Figura 4 Esquema de la tabla ingestions_log

- **stats:** Esta tabla guarda las estadísticas obtenidas por cada batch procesado, es una tabla temporal que se va actualizando cada vez que se ingesta un nuevo batch. Esta tabla solo tiene un único registro asociado tal como se observa en la Figura 5.

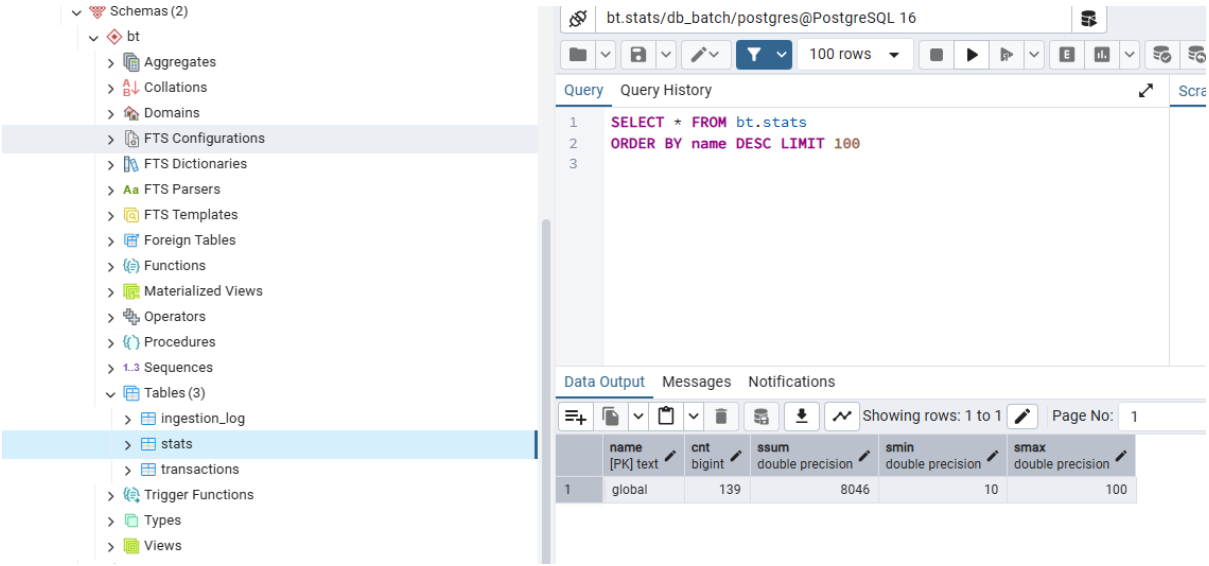

Figura 5. Esquema de la tabla stats

Para el despliegue e instalación del la base de datos Postgres (Server y PG Admin) se pude realizar localmente o mediante el uso de contendores Docker.

En el script **final.py** mediante la librería `psycopg2` se establece la conexión con la base de datos, teniendo en cuenta los parámetros iniciales. A continuación, en la Figura 6 se muestra la configuración de la conexión. Si se realiza el despliegue mediante contendores se debe cambiar el parámetro host por el nombre del contenedor.

```
# Configuracion Inicial de los datos
DB_CONF = {
    "host": os.getenv("DB_HOST", "localhost"),
    "port": int(os.getenv("DB_PORT", 5432)),
    "dbname": os.getenv("DB_NAME", "db_batch"),
    "user": os.getenv("DB_USER", "postgres"),
    "password": os.getenv("DB_PASSWORD", "user"),
}
```

Figura 6. Configuración inicial de la base de datos postgres

Posteriormente una vez establecida la conexión se realizan la creación de las tablas y del esquema, tal como se observa en la Figura 7 . En este caso la función ***init_db*** permite la creación del esquema y creación de las respectivas tablas. Dentro de esta función se ejecutan diferentes consultas SQL a través de la conexión al motor de la base de datos. Por ejemplo, la instrucción `cur.execute("CREATE SCHEMA IF NOT EXISTS BT;")` permite crear el esquema en caso de que aún no exista.

```
conn = psycopg2.connect(**DB_CONF)
init_db(conn)
```

```python
def init_db(conn):
    with conn.cursor() as cur:
        #Crear el esquema primero
        cur.execute("CREATE SCHEMA IF NOT EXISTS BT;")
        for sql in DDL.values():
            cur.execute(sql)
        cur.execute(INIT_STATS_ROW_SQL)
    conn.commit()
```

Figura 7. Creación de la conexión y esquema dentro de postgres

Para poblar las tablas (transactions y stats) de acuerdo con cada batch procesado se hace uso de la función ***process_file*** que permite procesar cada archivo de acuerdo con el tamaño del lote (batch) definido. Internamente se hacen diferentes consultas a las tablas de ***ingetestion_log y stats*** para actualizar o consultar la respectiva información según corresponda.

```python
def process_file(conn, filepath, source_file_name, microbatch_size=100):

    # Crear cursor para enviar y recibir ordenes SQL
    cur = conn.cursor()

    # Verificar si el archivo CSV  ya fue procesado
    cur.execute(CHECK_FILE_SQL, (source_file_name,))
    if cur.fetchone():
        print(f"[SKIP] {source_file_name} ya procesado previamente.")
        cur.close()
        return 0


    inserted_total = 0
    batch, batch_prices = [], []

    with open(filepath, newline="", encoding="utf-8") as f:
        reader = csv.DictReader(f)
        # Lectura de cada fila para cada archivo
        for idx, row in enumerate(reader, start=1):
            try:
                # Conversion al tipo de dato requerido
                ts, price, user_id = parse_row(row)
            except Exception as e:
                print(f"[WARN] Fila {idx} inválida en {source_file_name}: {e}")
                continue

            # Se agrega c/d fila en formato de tupla (4) a la lista batch
            batch.append((ts, price, user_id, source_file_name))
            # Dentro de la lista batch_prices se agrega el precio de c/d registro para posteriores calculos.
            batch_prices.append(price)
```

Figura 8. Función principal que permite poblas las tablas transactions y stats de acuerdo con cada ejecución

Nota: Es importante resaltar que para la actualización de las estadísticas se manejan tuplas y listas que permite iteración tras iteración actualizarse, no obstante hasta que se cumpla el tamaño del lote definido se escribe directamente en la tabla stats de la base de datos para posteriores consultas.

4. Casos de Prueba

En esta sesión se describen algunos casos de prueba:

- *Caso 1: Tablas vacías*

cómo se observa la Figura 9, inicialmente las tablas se encuentran vacias

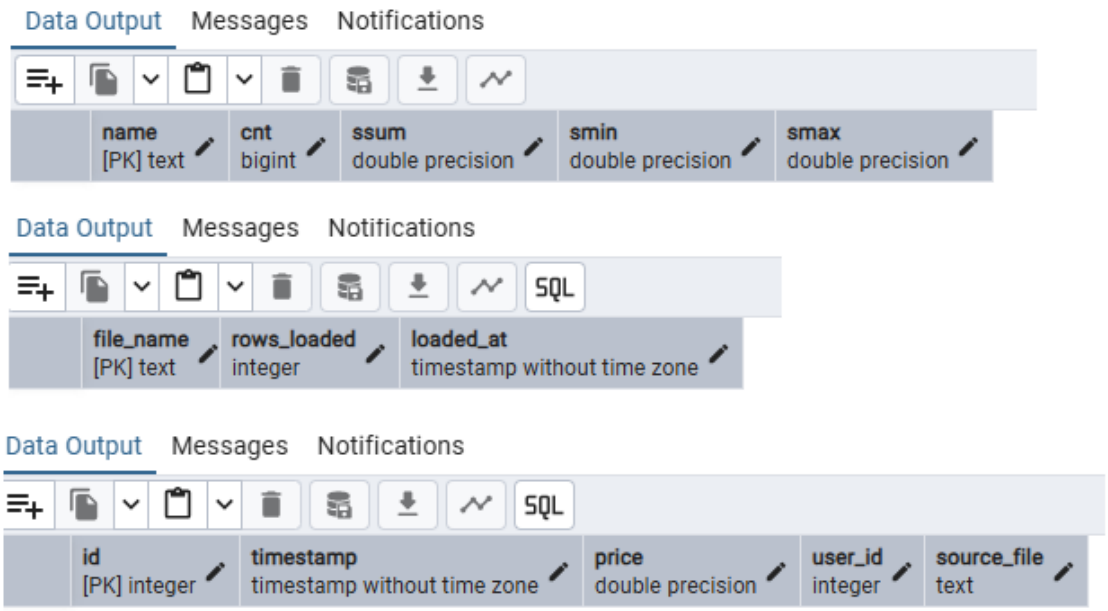

Figura 9. Tablas vacías reto técnico

- *Caso 2: Ingesta de datos (batch size = 8)*

Para este caso se realiza la ingesta de los archivos (.csv) tomando un tamaño del lote (batchsize de 8 registros). Al correr el script se observa, que los datos se insertan en bloques de 8 registros y al finalizar cada batch se actualizan las estadísticas en la tabla stats. Cuando finaliza el procesamiento de cada archivo csv, se muestras las estadísticas acumuladas que se han ido actualizado en cada iteración. La actualización de las estadísticas en la tabla ***stats*** depende del tamaño del batch.

```
=== Procesando 2012-2.csv ===
[INFO] Batch de 8 filas insertado.....
[INFO] Batch de 8 filas insertado.....
[INFO] Batch de 8 filas insertado.....
[INFO] Batch de 5 filas insertado.....
[OK] 2012-2.csv -> 29 filas insertadas.
[Después de 2012-2.csv] STATS -> rows=49, sum=2783.0, min=10.0, max=100.0, mean=56.795918367346935

=== Procesando 2012-3.csv ===
[INFO] Batch de 8 filas insertado.....
[INFO] Batch de 8 filas insertado.....
[INFO] Batch de 8 filas insertado.....
[INFO] Batch de 7 filas insertado.....
[OK] 2012-3.csv -> 31 filas insertadas.
[Después de 2012-3.csv] STATS -> rows=80, sum=4633.0, min=10.0, max=100.0, mean=57.9125

=== Procesando 2012-4.csv ===
[INFO] Batch de 8 filas insertado.....
[INFO] Batch de 8 filas insertado.....
[WARN] Fila 17 inválida en 2012-4.csv: price empty
[INFO] Batch de 8 filas insertado.....
[WARN] Fila 27 inválida en 2012-4.csv: price empty
[INFO] Batch de 4 filas insertado.....
[OK] 2012-4.csv -> 28 filas insertadas.
[Después de 2012-4.csv] STATS -> rows=108, sum=6240.0, min=10.0, max=100.0, mean=57.77777777777778

=== Procesando 2012-5.csv ===
[INFO] Batch de 8 filas insertado.....
[INFO] Batch de 8 filas insertado.....
[INFO] Batch de 8 filas insertado.....
[INFO] Batch de 7 filas insertado.....
[OK] 2012-5.csv -> 31 filas insertadas.
[Después de 2012-5.csv] STATS -> rows=139, sum=8046.0, min=10.0, max=100.0, mean=57.884892086330936

>>> Estadísticas acumuladas tras 2012-*.csv
 STATS -> rows=139, sum=8046.0, min=10.0, max=100.0, mean=57.884892086330936
```

Figura 10. Ingesta de datos y actualización de estadísticas

Finalmente, cuando se procesan todos los archivos .csv se obtiene las estadisticas (*count = 139 registros, suma price =8046, minimo price= 10, maximo_price = 100 y mean price = 57.88*), si se comparan estos valores que iteración tras iteración se actualizan con los valores finales obtenidos directamente de la tabla transactions se observa que los valores coinciden.

```
5   select * from bt.transactions
6
7
8   SELECT
9       COUNT(*) AS total_registros,
10      AVG(price) AS promedio_price,
11      MIN(price) AS minimo_price,
12      MAX(price) AS maximo_price
13  FROM bt.transactions;
```

Data Output | Messages | Notifications

Showing rows: 1 to 1 | Page No: 1 of 1

	total_registros bigint	promedio_price double precision	minimo_price double precision	maximo_price double precision
1	139	57.884892086330936	10	100

Figura 11.Verificación de las estadísticas directamente de la tabla transactions vs estadísticas incrementales

- *Caso 3: Ingesta de datos (batch size = 8) archivo validación.csv*

Para este caso se realiza la inserción del archivo validation.csv. Este archivo únicamente tiene 8 registros que coinciden con el tamaño del batch, por tal motivo se realiza la inserción en un único lote.

```
(envpragma) C:\Users\German\Desktop\Reto Pragma>python final.py --chunksize 8 --validation

=== Procesando SOLO validation.csv ===
[INFO] Batch de 8 filas insertado.....
[OK] validation.csv -> 8 filas insertadas.

>>> Estadísticas tras validation.csv
STATS -> rows=147, sum=8380.0, min=10.0, max=100.0, mean=57.006802721088434
```

Figura 12. Ingesta y procesamiento archivo validation.csv

Al final se muestran las estadísticas actualizadas, que coinciden con las estadísticas obtenidas directamente de la tabla transactions, una vez agregado los registros del archivo validatios.csv

```sql
SELECT
    COUNT(*) AS total_registros,
    AVG(price) AS promedio_price,
    MIN(price) AS minimo_price,
    MAX(price) AS maximo_price
FROM bt.transactions;
```

Data Output | Messages | Notifications

Showing rows: 1 to 1 | Page No: 1 of 1

	total_registros bigint	promedio_price double precision	minimo_price double precision	maximo_price double precision
1	147	57.006802721088434	10	100

Figura 13.Validación de la ingesta del archivo validation.csv vs estadísticas incrementales

- *Caso 4: Ingesta de datos archivo validación.csv (chunksize = 2)*

En este caso se ejecuta primero el pipeline de ingesta para el archivo *validacion.csv.* Se observa que los datos se cargan en batchsize = 2 (2 registros) y cuando termina la ingesta del archivo se muestran las estadísticas asociadas.

```
(envpragma) C:\Users\German\Desktop\Reto Pragma>python final.py --chunksize 2 --validation

=== Procesando SOLO validation.csv ===
[INFO] Batch de 2 filas insertado.....
[INFO] Batch de 2 filas insertado.....
[INFO] Batch de 2 filas insertado.....
[INFO] Batch de 2 filas insertado.....
[OK] validation.csv -> 8 filas insertadas.

>>> Estadísticas tras validation.csv
 STATS -> rows=8, sum=334.0, min=11.0, max=92.0, mean=41.75
```

Figura 14. Cargue único del archivo validations.csv dentro de la base de datos

Estos valores se comprueban directamente en las diferentes tablas, donde se observan datos del archivo validation.csv ingestado

```
SELECT * FROM bt.stats

SELECT * FROM bt.ingestion_log

select * from bt.transactions
```

	name [PK] text	cnt bigint	ssum double precision	smin double precision	smax double precision
1	global	8	334	11	92

	file_name [PK] text	rows_loaded integer	loaded_at timestamp without time zone
1	validation.csv	8	2025-08-21 14:52:20.297276

	id [PK] integer	timestamp timestamp without time zone	price double precision	user_id integer	source_file text
1	1	2012-06-01 00:00:00	26	10	validation.csv
2	2	2012-06-02 00:00:00	11	8	validation.csv
3	3	2012-06-03 00:00:00	13	9	validation.csv
4	4	2012-06-04 00:00:00	92	7	validation.csv
5	5	2012-06-05 00:00:00	31	10	validation.csv

Figura 15.Verificación de datos en las tablas stats, ingestion_log , transactions una vez ingestado únicamente el archivo validation.csv

5. Conclusiones

- Para la ejecución del ejercicio se puede instalar el servidor porgres y pg-admin localmente, sin embargo, también se puede realizar el ejercicio mediante contenedores Docker.
- En este ejercicio se cargan los datos secuencialmente dentro de la tabla transactions dependiendo del tamaño del **batch**, asi mismo para cada batch se calculan las respectivas estadísticas que se actualizan directamente en la tabla **stats**