**I.U. PASCUAL BRAVO**
**ET 0155 – Fundamentos de Big Data**
**Profesor: MSc Ing. Jaime E Soto U**

# INFORME DE RESULTADOS

## Unidad 3 - Tarea 7

**Equipo/Grupo** : 2
**Estudiantes** :

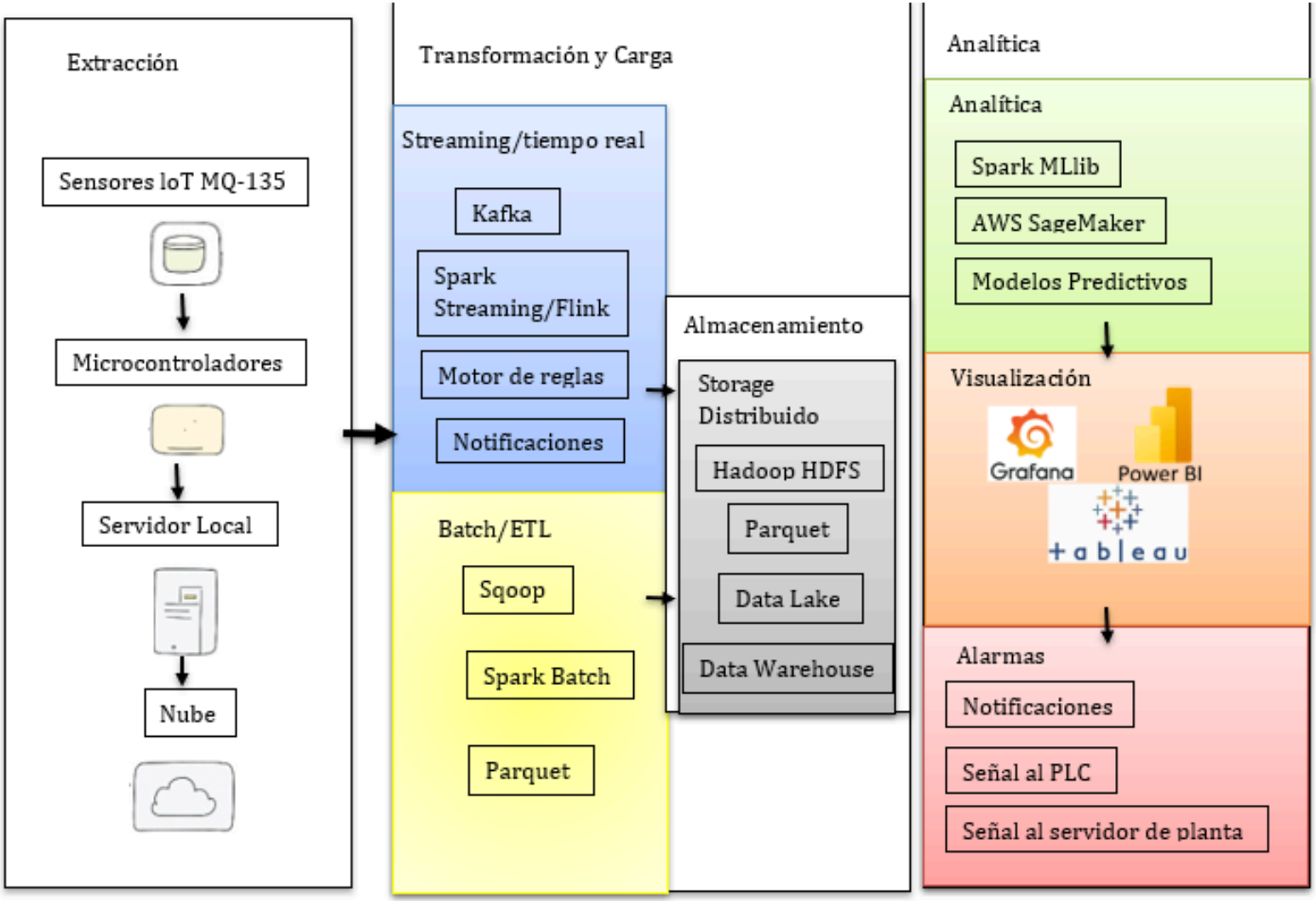
- **Harlan Santiago Enciso Riaño**
- **Miguel Angeles Rojas Pabon**
- **Maria Camila Rodriguez Ortiz**

## Objetivo General

**Diseñar una Arquitectura de Sistema de Información para el procesamiento de Big Data**

**I.U. PASCUAL BRAVO**
**ET 0155 – Fundamentos de Big Data**
**Profesor: MSc Ing. Jaime E Soto U**

**1.- Diseño Arquitectura del Sistema de Información** (para procesamiento Big Data)



## 2.- Cálculos de procesamiento y almacenamiento en la tabla lecturas

### 2.1.-Cálculos del procesamiento de lecturas

| Tipo de Lectura | Cálculos | Total lecturas |
|---|---|---|
| Lecturas diarias de un sensor | 86.400 seg/dia ÷ 5 seg/lectura | 17.280 lecturas/día |
| Lecturas de un mes de una línea de producción. Nota: un mes = 30 días. | 17.280 lect/dia × 20 sensores/ línea × 30 días | 10.368.000 lecturas/mes |
| Lecturas de un año de todas las líneas de producción de todas las fábricas. Nota: un año = 365 días. | 17.280 × 240 sensores × 365 | 1.513.728.000 lecturas/año |

***Nota: "Cálculos" debe mostrar el procedimiento para calcular el total de lecturas***

### 2.2.- Almacenamiento en tabla "lecturas"

| Período de almacenamiento | Lecturas | Tamaño Tupla | Total Bytes |
|---|---|---|---|
| Un (1) minuto | 12 lect/min × 240 = 2.880 | 64 B | **184.320 B** (≈ 180 KB) |
| Un (1) hora | 720 lect/hora × 240 = 172.800 | 64 B | **11.059.200 B** (≈ 11.06 MB) |
| Un (1) día | 17.280 × 240 = 4.147.200 | 64 B | **265.420.800 B** (≈ 265.42 MB) |
| Un (1) mes | 4.147.200 × 30 = 124.416.000 | 64 B | **7.962.624.000 B** (≈ 7.96 GB) |
| Un (1) año | **4.147.200 × 365 = 1.513.728.000** | 64 B | **96.878.592.000 B** (≈ 96.88 GB) |

***Nota: "Tamaño Tupla" es el tamaño en BYTES del registro de la tabla "lectura" de la base de datos "monitoreo-produccion" de PostgreSQL***

### 2.3.- Almacenamiento en tabla "lecturas"

| Lote de "lecturas" | Lecturas | Tamaño Tupla | Total Bytes |
|---|---|---|---|
| Lote 1 | **3.000.000** | 64 B | **192,000,000 B** (≈ 192 MB) |
| Lote 2 | **20.000.000** | 64 B | **1,280,000,000 B** (≈ 1.28 GB) |

***Nota: "Tamaño Tupla" es el tamaño en BYTES del registro de la tabla "lectura" de la base de datos "monitoreo-produccion" de PostgreSQL***

***2.4.- ¿Cada cuánto tiempo se debe limpiar la hoja de cálculo "lector-sensor" y la hoja de cálculo "lector-fabrica" antes de que se llegue al límite del máximo de registros permitidos por hoja de cálculo con formato "xlxs? Explique brevemente como realizó ellos cálculos.***

## Hoja "lector-sensor" (una hoja por sensor)

- Lecturas/día por sensor = **17,280**
- Días para llegar al límite:
  1.048.576 ÷ 17.280 ≈ 60.67 dias

**Debe limpiarse cada 60 días aprox** (≈ 2 meses)

## Hoja "lector-fabrica" (todas las lecturas de todos los sensores)

- Lecturas por día de todos los sensores = **4.147.200**

- Horas para llegar al límite:
  1.048.576 ÷ 4.147.200 ≈ 0.253 días ≈ 6.07 horas

**Debe limpiarse cada 6 horas** para no superar el límite.

## Cálculo

El límite de filas de Excel es **1.048.576**
Se divide el máximo permitido entre la cantidad de lecturas generadas en el período:

$$\text{Tiempo} = \frac{1.048.576}{Lecturas\ por\ periodo}$$

Esto permite saber cuántos días u horas pasarán antes de que la hoja alcance su capacidad máxima.

### 3.- Cálculo del costo de almacenamiento de en AWS de 20 millones de registros.

#### 3.1.- Cálculo de almacenamiento

| Tipo Almacenamiento | Total Bytes | Costo x byte | Costo Total |
|---|---|---|---|
| Almacenamiento en Bloque Elástico (EBS) | 1.280.000.000 B | 0.08 USD / GB-mes → 0.08 / 1.000.000.000 = **0.00000000008 USD/B** | 0.19 USD / mes |
| Amazon Simple Storage Service (Amazon S3) | 1.280.000.000 B | 0.023 USD / GB-mes → 0.023 / 1.000.000.000 = **0.00000000023 USD/B** | 0.02944 USD / mes |
| Amazon Aurora | 1.280.000.000 B | 0.10 USD / GB-mes → 0.10 / 1.000.000.000 = **0.00000000010 USD/B** | 0.128 USD / mes |

#### 3.2.- ¿Por qué la diferencia de costos en los diferentes tipos de almacenamiento?

**EBS (bloque)** es almacenamiento de bloques conectado a una instancia EC2. Su precio incluye rendimiento consistente y baja latencia; gp3 ofrece rendimiento incluido y cobro por GB y por IOPS/throughput extra si se provisionan. Eso lo hace más caro que S3 por GB porque está pensado para discos de sistema/BD con I/O altos. Amazon Web Services, Inc.+1

**S3 (objeto)** está optimizado para almacenamiento de objetos poco costoso, alta durabilidad y acceso vía HTTP; S3 cobra menos por GB pero puede tener cargos por solicitudes y transferencia. Es ideal para archivos, backups, data lake (Parquet). Amazon Web Services, Inc.

**Aurora** es un servicio de base de datos gestionado: su precio incluye almacenamiento replicado, durabilidad y operaciones de I/O administradas; por eso el almacenamiento por GB suele ser mayor que S3 y además hay cargos por I/O y por instancias del clúster que en práctica elevan el costo total. Amazon Web Services, Inc.

### 3.3.- Cotización AWS para el tipo de almacenamiento en EBS

**Estimate summary**

| Upfront cost | Monthly cost | Total 12 months cost |
|---|---|---|
| 0.00 USD | 0.19 USD | 2.28 USD<br>Includes upfront cost |

**Detailed Estimate**

| Name | Group | Region | Upfront cost | Monthly cost |
|---|---|---|---|---|
| Amazon Elastic Block Store (EBS) | - | US East (Ohio) | 0.00 USD | 0.19 USD |
| **Status** | - | | | |
| **Description:** | - | | | |
| **Config summary** | Number of volumes (1), Average duration of volume (730 hours per month), Storage amount per volume (1.28 GB), Snapshot Frequency (Monthly), Amount changed per snapshot (1 GB), Provisioning IOPS per volume (gp3) (3000), General Purpose SSD (gp3) - Throughput (125 MBps) | | | |

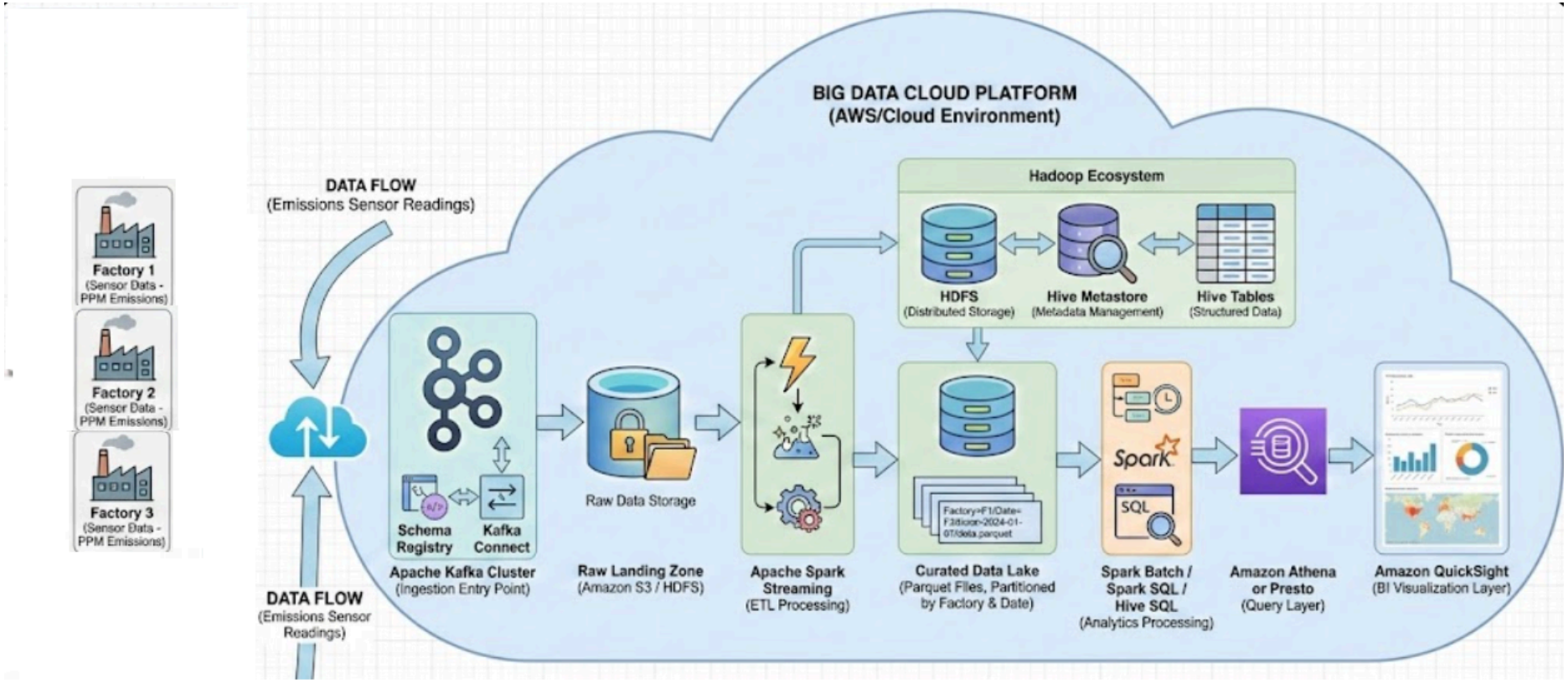
## 4.- Cálculo del costo por 1 año de uso de los servicios AWS

### 4.1.- Cálculo del costo de los servicios

| Nombre del Servicio | Costo individual | Costo anual |
|---|---|---|
| Amazon EBS | 0.19 USD/mes | -+2.28 USD |
| Amazon S3 Standard | 0.029 USD/mes | 0.35 USD |
| Amazon Aurora | 0.128 USD/mes | 1.54 USD |

**I. U. PASCUAL BRAVO**
**ET 0155 – Fundamentos de Big Data**
**Profesor: MSc Ing. Jaime E Soto U**

## 5.- Diseño de Arquitectura de Sistema de Información Big Data con Hadoop




+-+

### 6.- Algoritmo de "poblamiento" de la tabla "lecturas" y la hoja de cálculo "lecturas-sensor"

Para el poblamiento de los datos se desarrolló un script en Python llamado poblar_lecturas.py.

El objetivo del algoritmo es simular lecturas de benceno de un sensor (A1S01) y, con esos datos:

- Generar la hoja de cálculo del microcontrolador lecturas-sensor-A1S01.csv, con la estructura definida en el punto 4 del informe (id_lectura, fecha, hora, ppm, id_sensor, id_micro, línea, fábrica, latitud, longitud, estado_ppm y filtro_cercano).
- Insertar las mismas lecturas en la tabla lecturas de la base de datos monitoreo-produccion, respetando la estructura diseñada en la Tarea 6 (fecha_hora, id_fabrica, id_linea, id_sensor, id_filtro, id_producto, ppm, temperatura, humedad, latitud, longitud, id_clasificacion_ppm, alarma_activa y origen).

El script genera lecturas aleatorias para un intervalo de tiempo partiendo de la fecha 2025-11-22 a partir de las 08:00:00, con un registro cada 10 segundos. Para cada lectura se simulan:

- concentración de benceno en ppm,
- temperatura
- humedad
- geolocalización (latitud y longitud).

Con el valor de ppm se llama a una función de clasificación de peligrosidad, que asigna un id_clasificacion_ppm y un estado textual (Normal, Riesgo o Crítico) y determina si la alarma está activa (alarma_activa).

Finalmente:

- se escribe un archivo CSV con el formato de la hoja de cálculo del sensor;
- y se realiza un INSERT en lote a la tabla lecturas mediante la librería psycopg2, utilizando los campos de la estructura real de la tabla.

## Script 1. Algoritmo de poblamiento poblar_lecturas.py

```
import csv

import random

from datetime import datetime, timedelta


import psycopg2
from psycopg2 import Error

DB_HOST = "localhost"
DB_PORT = "5432"
DB_NAME = "monitoreo-produccion"
```

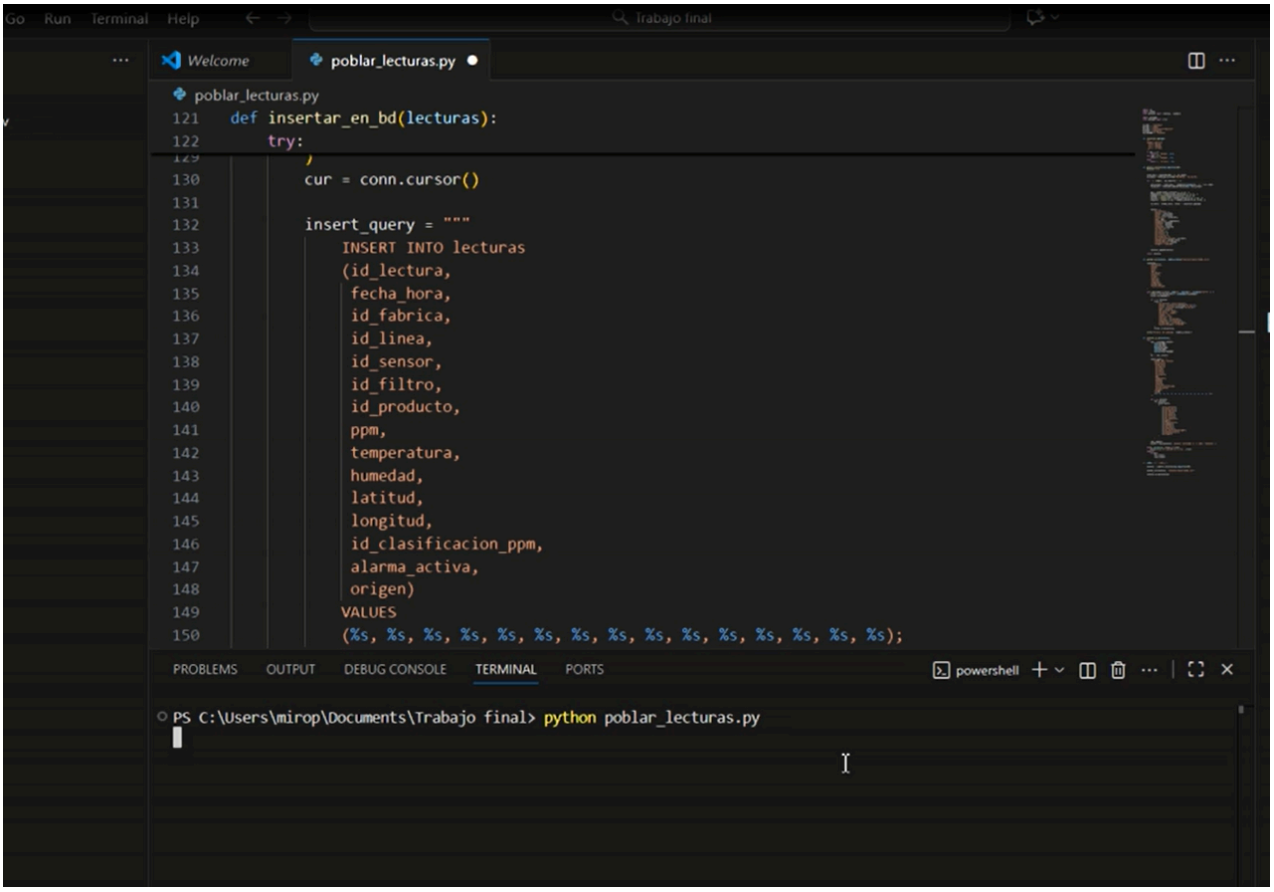
```
DB_USER = "postgres"
DB_PASSWORD = "postgres"   # ajustar según instalación


def clasificar_ppm(ppm):
    if ppm < 1.0:
        return 1, "Normal", False
    elif ppm < 5.0:
        return 2, "Riesgo", True
    else:
        return 3, "Crítico", True


def generar_lecturas(num_registros=100):
    lecturas = []
    fecha_base = datetime(2025, 11, 22).date()
    hora_base = datetime.strptime("08:00:00", "%H:%M:%S")

    for i in range(1, num_registros + 1):
        hora_actual = (hora_base + timedelta(seconds=10 * (i - 1))).time()
        fecha_hora = datetime.combine(fecha_base, hora_actual)

        ppm = round(random.uniform(0.1, 15.0), 1)
        temperatura = round(random.uniform(20.0, 35.0), 1)
        humedad = round(random.uniform(30.0, 80.0), 1)
        latitud = round(6.250 + random.uniform(-0.01, 0.01), 6)
        longitud = round(-75.565 + random.uniform(-0.01, 0.01), 6)

        id_clasif, estado_texto, alarma = clasificar_ppm(ppm)

        lectura = {
            "id_lectura": i,
            "fecha": fecha_base,
            "hora": hora_actual,
            "fecha_hora": fecha_hora,
            "ppm": ppm,
            "temperatura": temperatura,
            "humedad": humedad,
            "latitud": latitud,
            "longitud": longitud,
            "id_fabrica": 1,
```

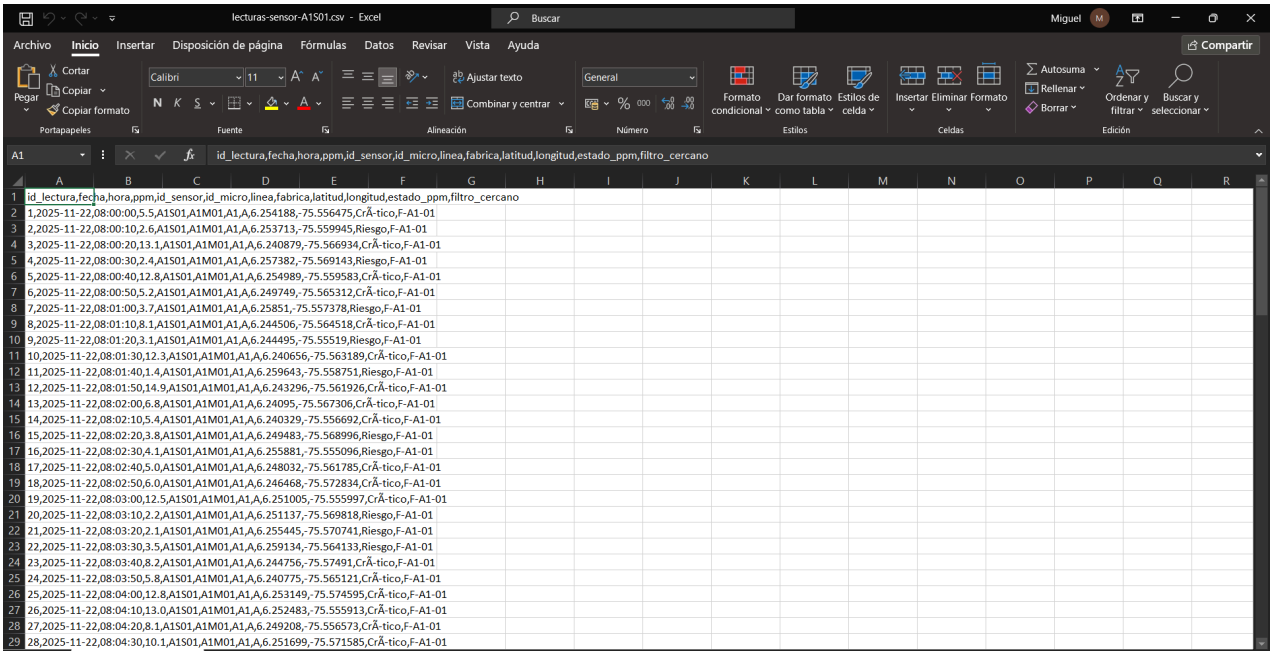
```
            "id_linea": 1,
            "id_sensor": 1,
            "id_filtro": 1,
            "id_producto": 1,
            "id_clasificacion_ppm": id_clasif,
            "estado_ppm": estado_texto,
            "alarma_activa": alarma,
            "origen": "simulado"
        }
        lecturas.append(lectura)
    return lecturas

def guardar_csv(lecturas, nombre_archivo="lecturas-sensor-A1S01.csv"):
    fieldnames = [
        "id_lectura","fecha","hora","ppm","id_sensor","id_micro",
        "linea","fabrica","latitud","longitud","estado_ppm","filtro_cercano"
    ]
    with open(nombre_archivo, mode="w", newline="", encoding="utf-8") as f:
        writer = csv.DictWriter(f, fieldnames=fieldnames)
        writer.writeheader()
        for l in lecturas:
            fila = {
                "id_lectura": l["id_lectura"],
                "fecha": l["fecha"].strftime("%Y-%m-%d"),
                "hora": l["hora"].strftime("%H:%M:%S"),
                "ppm": l["ppm"],
                "id_sensor": "A1S01",
                "id_micro": "A1M01",
                "linea": "A1",
                "fabrica": "A",
                "latitud": l["latitud"],
                "longitud": l["longitud"],
                "estado_ppm": l["estado_ppm"],
                "filtro_cercano": "F-A1-01"
            }
            writer.writerow(fila)
    print(f"Archivo CSV generado: {nombre_archivo}")

def insertar_en_bd(lecturas):
```

```
    try:
        conn = psycopg2.connect(
            host=DB_HOST, port=DB_PORT, dbname=DB_NAME,
            user=DB_USER, password=DB_PASSWORD
        )
        cur = conn.cursor()
        insert_query = """
            INSERT INTO lecturas
            (id_lectura, fecha_hora, id_fabrica, id_linea, id_sensor,
             id_filtro, id_producto, ppm, temperatura, humedad,
             latitud, longitud, id_clasificacion_ppm, alarma_activa, origen)
            VALUES
            (%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s);
        """
        for l in lecturas:
            cur.execute(insert_query, (
                l["id_lectura"], l["fecha_hora"], l["id_fabrica"],
                l["id_linea"], l["id_sensor"], l["id_filtro"],
                l["id_producto"], l["ppm"], l["temperatura"],
                l["humedad"], l["latitud"], l["longitud"],
                l["id_clasificacion_ppm"], l["alarma_activa"], l["origen"]
            ))
        conn.commit()
        print(f"{len(lecturas)} lecturas insertadas en la tabla 'lecturas'.")
    except (Exception, Error) as error:
        print("Error al insertar en la BD:", error)
    finally:
        if conn:
            cur.close()
            conn.close()

if __name__ == "__main__":
    lecturas = generar_lecturas(num_registros=100)
    guardar_csv(lecturas, "lecturas-sensor-A1S01.csv")
    insertar_en_bd(lecturas)
```

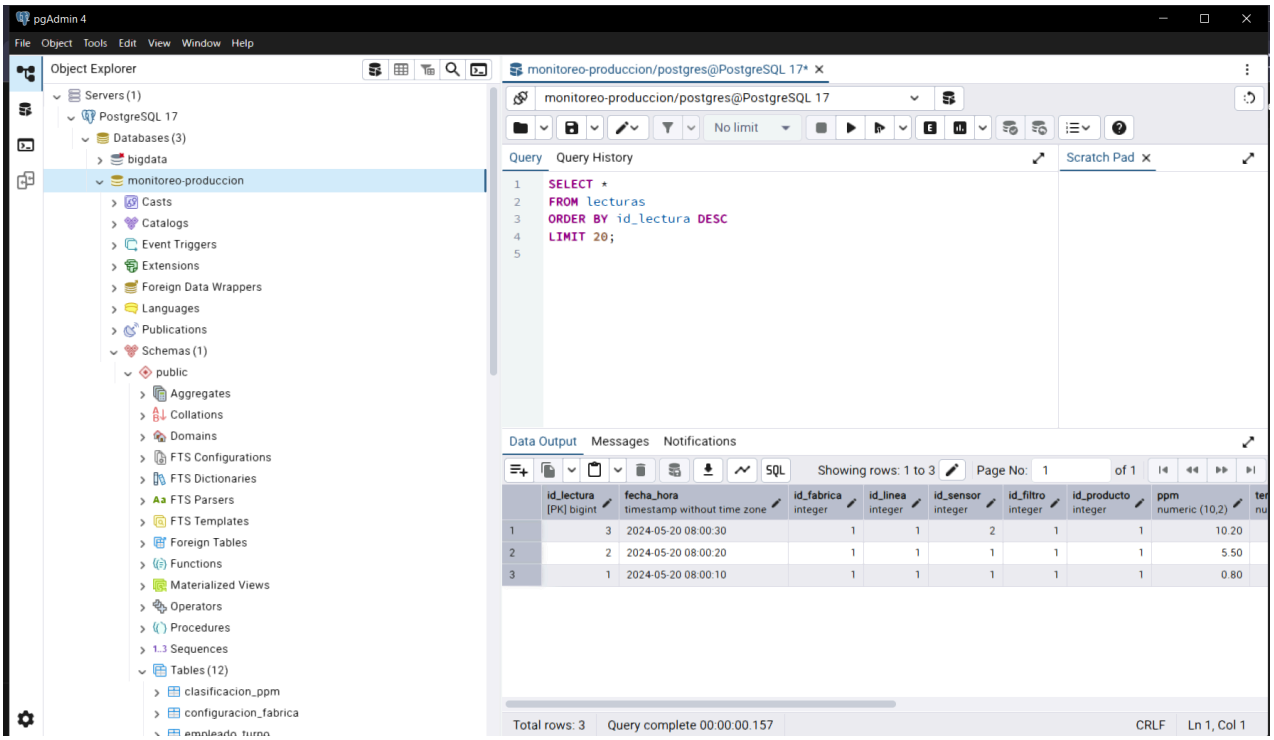
**I. U. PASCUAL BRAVO**
**ET 0155 – Fundamentos de Big Data**
**Profesor: MSc Ing. Jaime E Soto U**

# Evidencias de ejecución del algoritmo



**I. U. PASCUAL BRAVO**
**ET 0155 – Fundamentos de Big Data**
**Profesor: MSc Ing. Jaime E Soto U**



**I. U. PASCUAL BRAVO**
**ET 0155 – Fundamentos de Big Data**
**Profesor: MSc Ing. Jaime E Soto U**

## 7.- Implementación de un tablero de control y monitoreo con la herramienta “Metabase”

## 8.- Conclusiones.

**Maria Camila Rodriguez Ortiz:**
Al desarrollar la arquitectura separada por capas comprendí de manera más profunda cómo funciona un sistema Big Data completo, desde la captura inicial de datos hasta su análisis y visualización final. Organizar el sistema en capas de Extracción, Transformación/Carga y Analítica me permitió entender que cada una cumple un rol específico, y que todas deben trabajar coordinadas para lograr un monitoreo confiable en tiempo real. También aprendí a calcular el volumen real de lecturas generadas por los sensores y cómo este crecimiento afecta directamente el almacenamiento, el rendimiento y las herramientas necesarias para procesarlo, como Hadoop o sistemas distribuidos.

**Harlan Enciso:**
Saber calcular los costos para diferentes servicios de la nube realmente es de gran valor ya que da un plus sobre el conocimiento de la nube, y es algo que se hace en todos los proyectos a gran escala ya que AWS es la nube más utilizada por ende puede ser un diferenciador a la hora buscar trabajo, por parte de la arquitectura realmente permite conocer las herramientas que se deben usar a la hora de ejecutar un proyecto de big data y es algo muy util porque se puede usar como un estándar, y permite conocer servicios que ofrecen escalabilidad en proyectos.

**Miguel Rojas Pabon:**
Este proyecto final me permitió comprender de manera práctica cómo se integra una arquitectura Big Data completa, desde la generación y poblamiento de datos hasta la visualización en tiempo real mediante un dashboard. Pude aplicar procesos ETL, trabajar con bases de datos, automatizar la creación de lecturas y construir indicadores útiles para la toma de decisiones. Además, el uso de herramientas como Python y Metabase me permitió entender cómo se conectan todos los componentes para monitorear un sistema crítico. En general, fue una experiencia retadora pero muy enriquecedora, que fortalece mis habilidades técnicas y mi capacidad para abordar problemas reales con soluciones basadas en datos.

## 9.- Video de sustentación:

*https://drive.google.com/file/d/1ljHXlNf3UaAnIf32JOa7weQXWzb37e3a/view?usp=sharing*