¡Claro que sí! Aquí tienes dos versiones del script: una en **Python** (ideal si tu backend de IA está en Flask/FastAPI/Django) y otra en **Node.js** (ideal si usas Express/NestJS).

Ambos scripts hacen lo siguiente:

1. **Simulan** los datos de entrada (el contexto industrial).
2. **Registran** la generación de la imagen (Insert en `ai_generations`).
3. **Simulan** que el usuario vota negativo 10 segundos después (Insert en `ai_feedback`).

Opción A: Script en Python (Recomendado para Backend de IA)

Este script utiliza la librería estándar `psycopg2`.

Prerrequisitos: `pip install psycopg2-binary`

Python
```
import psycopg2
import json
from datetime import datetime

# CONFIGURACIÓN DE CONEXIÓN
DB_CONFIG = {
    "dbname": "pyrolysis_db",
    "user": "postgres",
    "password": "tu_password_seguro",
    "host": "localhost",
    "port": "5432"
}

def simulate_generation_cycle():
    conn = None
    try:
        conn = psycopg2.connect(**DB_CONFIG)
        cur = conn.cursor()

        print("🔌 Conectado a la Base de Datos.")

        # ------------------------------------------------------
        # PASO 0: Crear un Proyecto Ficticio (o usar uno existente)
        # ------------------------------------------------------
        project_sql = """
            INSERT INTO projects (name, owner_id)
```

```
    VALUES (%s, %s) RETURNING id;
"""
# Nota: owner_id es NULL aquí para el ejemplo, pon un UUID real de usuario si tienes
cur.execute(project_sql, ("Proyecto Biomasa Algas 2024", None))
project_id = cur.fetchone()[0]
print(f"✅ Proyecto creado: {project_id}")

# ------------------------------------------------------
# PASO 1: Registrar la GENERACIÓN (Lo que hizo la IA)
# ------------------------------------------------------

# Datos "Duros" (Snapshot)
telemetry_data = {
    "demand_kw": 385,
    "cost_eur": 184800,
    "efficiency_rate": 0.06,
    "machine_status": "OPTIMAL"
}

# Configuración del Motor
engine_config = {
    "audience": "INVESTOR",
    "style_preset": "CYBERPUNK_DARK",
    "model_version": "dall-e-3"
}

gen_sql = """
    INSERT INTO ai_generations
    (project_id, telemetry_snapshot, engine_config, full_prompt_text, output_url)
    VALUES (%s, %s, %s, %s, %s)
    RETURNING id;
"""

cur.execute(gen_sql, (
    project_id,
    json.dumps(telemetry_data), # PostgreSQL JSONB ama json.dumps
    json.dumps(engine_config),
    "Subject: massive dark-matter turbine... Lighting: soft bioluminescent glow...",
    "https://s3.amazonaws.com/bucket/img_gen_001.png"
))

generation_id = cur.fetchone()[0]
print(f"✅ Generación registrada ID: {generation_id}")
```

```
        # ... Pasan 10 segundos y el usuario ve la imagen ...

        # ---------------------------------------------------
        # PASO 2: Registrar el FEEDBACK (La queja del usuario)
        # ---------------------------------------------------

        feedback_sql = """
            INSERT INTO ai_feedback
            (generation_id, sentiment, tags, user_comment)
            VALUES (%s, %s, %s, %s);
        """

        user_tags = ["TOO_DARK", "ABSTRACT"] # El usuario seleccionó esto en la UI
        user_comment = "No se ve bien la turbina, aclarar la imagen."

        cur.execute(feedback_sql, (
            generation_id,
            "NEGATIVE",
            user_tags, # Psycopg2 maneja listas de Python a Arrays de Postgres automáticamente
            user_comment
        ))

        print("✅ Feedback negativo registrado. El sistema aprenderá de esto.")

        # Confirmar cambios
        conn.commit()

    except (Exception, psycopg2.DatabaseError) as error:
        print(f"❌ Error en la transacción: {error}")
        if conn:
            conn.rollback()
    finally:
        if conn:
            cur.close()
            conn.close()
            print("🔌 Conexión cerrada.")

if __name__ == "__main__":
    simulate_generation_cycle()
```

Opción B: Script en Node.js (Recomendado para API/Web)

Este script utiliza `pg` (node-postgres), el estándar de la industria.

Prerrequisitos: `npm install pg`

JavaScript

```
const { Pool } = require('pg');

// CONFIGURACIÓN DE CONEXIÓN
const pool = new Pool({
  user: 'postgres',
  host: 'localhost',
  database: 'pyrolysis_db',
  password: 'tu_password_seguro',
  port: 5432,
});

async function simulateCycle() {
  const client = await pool.connect();

  try {
    await client.query('BEGIN'); // Iniciamos transacción

    console.log("🔌 Conectado a Postgres.");

    // ------------------------------------------------------
    // PASO 0: Crear Proyecto
    // ------------------------------------------------------
    const projectRes = await client.query(
      'INSERT INTO projects (name) VALUES ($1) RETURNING id',
      ['Planta Piloto Barcelona']
    );
    const projectId = projectRes.rows[0].id;
    console.log(`✅ Proyecto creado: ${projectId}`);

    // ------------------------------------------------------
    // PASO 1: Insertar Generación (JSONB es nativo en JS)
    // ------------------------------------------------------
    const telemetrySnapshot = {
      demand_kw: 385,
      cost_eur: 184800,
      timestamp: new Date().toISOString()
    };

    const engineConfig = {
```

```
    style: 'CYBERPUNK_DARK',
    audience: 'ENGINEER'
  };

  const genQuery = `
    INSERT INTO ai_generations
    (project_id, telemetry_snapshot, engine_config, full_prompt_text, output_url)
    VALUES ($1, $2, $3, $4, $5)
    RETURNING id
  `;

  const genRes = await client.query(genQuery, [
    projectId,
    telemetrySnapshot, // node-postgres convierte objetos JS a JSONB automáticamente
    engineConfig,
    "Subject: holographic blueprint of a turbine, cyan lines...",
    "https://cdn.pyrolysis-hub.com/assets/img_blueprint_v1.png"
  ]);

  const generationId = genRes.rows[0].id;
  console.log(`✅ Generación guardada ID: ${generationId}`);

  // ---------------------------------------------------------
  // PASO 2: Insertar Feedback
  // ---------------------------------------------------------
  const feedbackQuery = `
    INSERT INTO ai_feedback
    (generation_id, sentiment, tags, manual_instruction)
    VALUES ($1, $2, $3, $4)
  `;

  // Simulamos un feedback NEUTRO con petición manual
  await client.query(feedbackQuery, [
    generationId,
    'NEUTRAL',
    ['TEXT_GLITCH'],
    'Por favor quita las letras chinas que aparecieron en la esquina.'
  ]);

  console.log("✅ Feedback registrado.");

  await client.query('COMMIT'); // Todo salió bien
} catch (e) {
  await client.query('ROLLBACK'); // Algo falló, deshacer todo
```

```
    console.error('❌ Error:', e);
  } finally {
    client.release();
    pool.end();
  }
}

simulateCycle();
```

Puntos Clave de estos Scripts:

1. **Manejo de JSONB:** Fíjate cómo en Node.js pasamos el objeto JavaScript directo (`telemetrySnapshot`) y PostgreSQL lo guarda como JSON binario. Esto te permite guardar *cualquier* dato extraño que venga de tus sensores sin cambiar la tabla.
2. **Relaciones (Foreign Keys):** Usamos `RETURNING id` inmediatamente después de insertar. Esto es crucial. No hacemos otra consulta para buscar el ID; la base de datos nos lo devuelve en el mismo acto de creación, garantizando que el vínculo entre la **Generación** y el **Feedback** sea 100% preciso.
3. **Arrays de Texto:** En PostgreSQL, `tags TEXT[]` es un array real. En Python pasas `['tag1', 'tag2']` y en DB se guarda `{tag1, tag2}`. Esto hace que buscar "todas las quejas con tag TOO_DARK" sea rapidísimo.