

GUÍA DE IMPLEMENTACIÓN: PROYECTO SISTEMA DE GESTIÓN UNIVERSITARIA (POSTGRESQL)

Objetivo: Desplegar la base de datos del proyecto en PostgreSQL, configurando un entorno seguro con el usuario del estudiante (en adelante <tu_usuario>), creando la estructura completa (tablas, claves, restricciones) y realizando una carga masiva de datos que cumple con los requisitos del Caso de Estudio 4.

FASE 1: Configuración del Entorno (Usuario y Base de Datos)

Herramienta: pgAdmin 4 (o cualquier cliente SQL como DBeaver).

Usuario requerido: Superusuario (postgres) para la configuración inicial.

Paso 1.1: Ejecución del Script de Setup

Debido a restricciones de PostgreSQL (no se puede crear una DB dentro de una transacción), este script se diseñó para ejecutarse **por bloques**.

Instrucciones:

1. Conectar a `postgres` en pgAdmin.
2. Abrir Query Tool.
3. Reemplazar <tu_usuario> por tu nombre de usuario real (ej: fmerino) y <nombre_db> por el nombre de tu base (ej: fmerino_GestionUniversitaria) en el siguiente script.
4. Copiar el código y ejecutarlo **BLOQUE POR BLOQUE** (seleccionar con mouse y dar F5).
/*
=====
=====
 SCRIPT DE CONFIGURACIÓN DEL ENTORNO POSTGRESQL
 Variable a reemplazar: <tu_usuario> (ej: jperez)
 Variable a reemplazar: <nombre_db> (ej: jperez_GestionUniversitaria)
===== * /
-- BLOQUE 1: CREACIÓN DEL USUARIO (ROL)
DO \$\$
BEGIN
 IF NOT EXISTS (SELECT FROM pg_catalog.pg_roles WHERE rolname =
 '<tu_usuario>') THEN
 CREATE ROLE "<tu_usuario>" WITH LOGIN PASSWORD '123' CREATEDB;

```

        RAISE NOTICE 'Usuario <tu_usuario> creado exitosamente.';
    ELSE
        RAISE NOTICE 'El usuario <tu_usuario> ya existía.';
    END IF;
END
$$;

-- BLOQUE 2: CREACIÓN DE LA BASE DE DATOS
-- (Ejecutar SOLO esta línea seleccionada)
CREATE DATABASE "<nombre_db>"
    WITH OWNER = "<tu_usuario>"
    ENCODING = 'UTF8';

-- BLOQUE 3: ASIGNACIÓN DE PRIVILEGIOS TOTALES
GRANT CONNECT ON DATABASE "<nombre_db>" TO "<tu_usuario>";
ALTER DATABASE "<nombre_db>" OWNER TO "<tu_usuario>";
-- Nota: Las siguientes líneas otorgan permisos sobre el esquema public
de la base actual.
-- Asegúrate de que el dueño tenga control total.
GRANT ALL PRIVILEGES ON SCHEMA public TO "<tu_usuario>";
GRANT ALL PRIVILEGES ON ALL FUNCTIONS IN SCHEMA public TO "<tu_usuario>";



```

Paso 1.2: Conexión como <tu_usuario>

1. En pgAdmin, clic derecho en **Servers > Register > Server**.
2. **Name:** Proyecto_<tu_usuario>.
3. **Connection:**
 - Host: localhost
 - User: <tu_usuario>
 - Password: 123
 - Maintenance DB: <nombre_db> (ej: fmerino_GestionUniversitaria)
4. Guardar y Conectar.

FASE 2: Creación de la Estructura (DDL)

Usuario: <tu_usuario> (Conectado al servidor nuevo).

Fuente: Script generado por PowerDesigner (Script_Proyecto_Postgres.sql).

Instrucciones:

1. Abrir el archivo .sql generado por PowerDesigner.
2. **Limpieza:** Borrar las líneas iniciales que contienen `DROP TABLE` o `DROP INDEX` (para evitar errores en una base vacía).
3. Ejecutar el script completo.
4. **Verificación:** Revisar en el panel izquierdo (Schemas > public > Tables) que aparezcan las ~28 tablas (ESTUDIANTE, CARRERA, HISTORIAL_ACADEMICO, etc.).

FASE 3: Carga Masiva de Datos y Correcciones

Este script realiza tres acciones críticas:

1. **Refactorización:** Modifica la tabla PRESTAMO para agregar una PK subrogada (ID_PRESTAMO), corrigiendo un error lógico del modelo original.
2. **Limpieza:** Borra datos previos para evitar duplicados.
3. **Carga Masiva:** Inserta volúmenes de datos exigidos (500 estudiantes, 100 docentes, etc.) usando bucles PL/pgSQL.

Instrucciones: Copiar y ejecutar este bloque completo en el Query Tool de <tu_usuario>.

```
/*
=====
= CARGA MASIVA DE DATOS PARA POSTGRESQL (PL/pgSQL)
= Incluye corrección de estructura en tabla PRESTAMO.
=====

-- 1. LIMPIEZA Y REFACTORIZACIÓN
DO $$ BEGIN
    RAISE NOTICE 'Limiando y refactorizando...';

    -- Modificar PRESTAMO para permitir múltiples préstamos por día
    ALTER TABLE PRESTAMO DROP CONSTRAINT IF EXISTS PK_PRESTAMO;
    ALTER TABLE PRESTAMO ADD COLUMN IF NOT EXISTS ID_PRESTAMO SERIAL;
    ALTER TABLE PRESTAMO ADD CONSTRAINT PK_PRESTAMO PRIMARY KEY
    (ID_PRESTAMO);

    -- Limpiar tablas en orden de dependencia
    TRUNCATE TABLE HISTORIAL_ACADEMICO, CALIFICACION, ASIGNACION_BECA,
    PRESTAMO,
    TRIBUNAL_GRADO, PROCESO_TITULACION, AUTORIA, PUBLICACION,
    COLABORAN_EN,
    PROYECTO_INVESTIGACION, HORARIO_CLASE, MATRICULA, OFERTA_ASIGNATURA,
    DETALLE_MALLA, MALLA_CURRICULAR, CORREQUISITO, TIENE_PRERREQUISITO,
    TITULO_ACADEMICO, ESTUDIANTE, ASIGNATURA, PERIODO_ACADEMICO, CARRERA,
    DEPARTAMENTO, LIBRO, AULA, TIPO_BECA, DOCENTE, FACULTAD CASCADE;
END $$;

-- 2. BLOQUE DE INSERCIÓN DE DATOS
DO $$
DECLARE
    i INT; j INT; matr_id INT; hist_id INT;
BEGIN
    RAISE NOTICE 'Iniciando Carga Masiva...';

    -- [AQUÍ VA EL CÓDIGO DE LOS BUCLES FOR QUE GENERAMOS PREVIAMENTE]
    -- ... (Insertar Facultades, Docentes, Estudiantes, etc.) ...
    -- ... (Ver Script_Datos_Postgres.sql completo en el chat anterior)
    ...

    RAISE NOTICE 'Carga Masiva Completada.';
```

```
END $$;
```

FASE 4: Sincronización de Secuencias

Debido a que los datos se insertaron con IDs explícitos (1, 2, 3...) pero PostgreSQL usa secuencias automáticas para los nuevos registros, es obligatorio sincronizar los contadores.

Instrucciones: Ejecutar este script final para evitar errores de "Duplicate Key" en el futuro.

```
/* SINCRONIZACIÓN DE SECUENCIAS (CORREGIDO) */
DO $$
DECLARE
    rec RECORD;
    secuencia_nombre TEXT;
BEGIN
    RAISE NOTICE 'Ajustando secuencias...';

    -- Iteramos sobre las tablas para ajustar sus secuencias
    FOR rec IN SELECT * FROM (VALUES
        ('SEQ_LIBRO', 'libro', 'id_libro'),
        ('SEQ_PROYECTO', 'proyecto_investigacion', 'id_proyecto'),
        ('SEQ_PUBLICACION', 'publicacion', 'id_publicacion'),
        ('SEQ_TIPO_BECA', 'tipo_beca', 'id_beca'),
        ('SEQ_PROCESO_TIT', 'proceso_titulacion', 'id_proceso'),
        ('SEQ_HISTORIAL', 'historial_academico', 'id_historial'),
        ('SEQ_ESTUDIANTE', 'estudiante', 'id_estudiante'),
        ('SEQ_DOCENTE', 'docente', 'id_docente'),
        ('SEQ_MATRICULA', 'matricula', 'id_matricula'),
        ('SEQ_OFERTA', 'oferta_asignatura', 'id_oferta'),
        ('SEQ_AULA', 'aula', 'id_aula')
    ) AS t(nombre_seq, nombre_tabla, nombre_columna)
    LOOP
        secuencia_nombre := lower(rec.nombre_seq);

        -- Crear secuencia si no existe
        IF NOT EXISTS (SELECT FROM pg_class WHERE relname =
secuencia_nombre) THEN
            EXECUTE format('CREATE SEQUENCE %I START WITH 1 INCREMENT BY
1', secuencia_nombre);
        END IF;

        -- Ajustar valor al MAX ID
        EXECUTE format('SELECT setval('''%I''', COALESCE((SELECT MAX(%I)
FROM %I), 1))',
                        secuencia_nombre, rec.nombre_columna,
rec.nombre_tabla);

        -- Vincular como Default (Opcional)
        BEGIN
            EXECUTE format('ALTER TABLE %I ALTER COLUMN %I SET DEFAULT
nextval('''%I'''',
                    rec.nombre_tabla, rec.nombre_columna,
secuencia_nombre);
        EXCEPTION WHEN OTHERS THEN NULL; END;
    END LOOP;
```

```

-- Ajuste especial para PRESTAMO (SERIAL nativo)
IF EXISTS (SELECT FROM pg_class WHERE relname =
'prestamo_id_prestamo_seq') THEN
    PERFORM setval('prestamo_id_prestamo_seq', COALESCE((SELECT
MAX(id_prestamo) FROM prestamo), 1));
END IF;

RAISE NOTICE '>>> SECUENCIAS SINCRONIZADAS CORRECTAMENTE.';
END $$;

```

FASE 5: Validación Final

Ejecutar las siguientes consultas para confirmar el éxito del despliegue:

```

-- 1. Verificar Dueño (Debe ser <tu_usuario>)
SELECT pg_catalog.pg_get_userbyid(d.datdba) as dueño FROM pg_database d
WHERE datname = current_database();

-- 2. Conteo de Datos (Debe coincidir con requisitos)
SELECT 'Estudiantes (Req: 500)' as Item, count(*) as Cantidad FROM
estudiante
UNION ALL
SELECT 'Docentes (Req: 100)', count(*) FROM docente
UNION ALL
SELECT 'Historial (Req: 3000)', count(*) FROM historial_academico;

```

Resultado: Si el dueño es tu usuario y los conteos son 500, 100 y 3000 respectivamente, la implementación en PostgreSQL ha sido exitosa.