

PROYECTO BASE DE DATOS

Base de datos antiguos alumnos



PRESENTADO POR:

Sara Gil (Data Science)
Mary Marín (Data Science)
Álvaro Martínez (Data Science)
Carlos Gómez (Fullstack)
Cristina González (Fullstack)

Tareas realizadas

Tecnologías utilizadas



pgAdmin



SQL

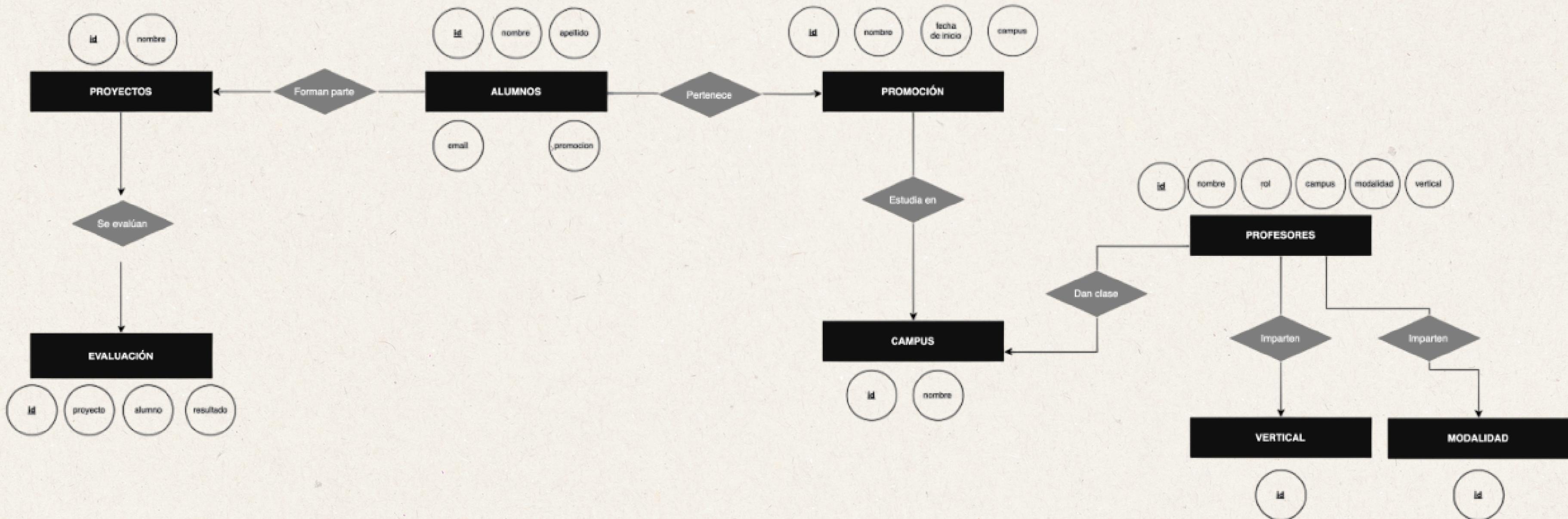


Render

01	Modelo Entidad-Relación
02	Modelo Lógico
03	Filtrado y normalización
04	Creación de tablas
05	Queries
06	Funcionalidades futuras

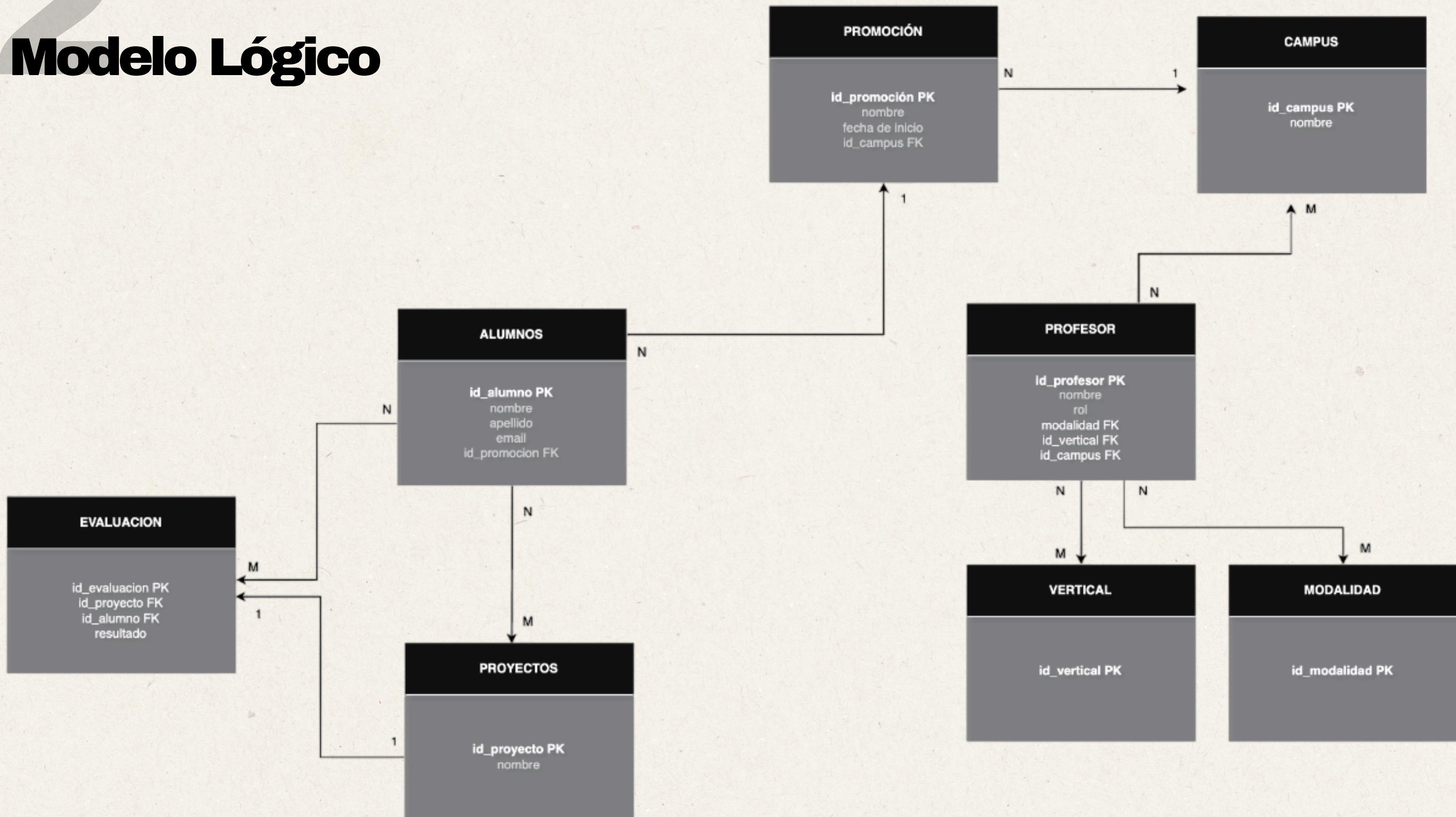
01

Modelo Entidad Relación



02

Modelo Lógico



03

Filtrado de tablas y normalización

Recogimos los cvs y realizamos un primer filtrado de datos y después, realizamos la importación a pgAdmin



Atendimos a los criterios de normalización:

- ✓ **1FN:** Todos los atributos son atómicos.
- ✓ **2FN:** No hay dependencias parciales porque las tablas con PK compuestas (como Evaluación) dependen completamente de su PK.
- ✓ **3FN:** No hay dependencias transitivas; los campos dependen directamente de su PK.

04 Creación de tablas y contenido

Una vez hecho esto, creamos nuestras tablas directamente en pgAdmin

Tabla campus

```
CREATE TABLE campus (
    id_campus SERIAL PRIMARY KEY,
    nombre VARCHAR(50) UNIQUE NOT NULL
);
```

```
INSERT INTO campus (nombre) VALUES
('Madrid'),
('Valencia')
ON CONFLICT (nombre) DO NOTHING;
```

Tabla vertical

```
CREATE TABLE vertical (
    id_vertical SERIAL PRIMARY KEY,
    nombre VARCHAR(10) UNIQUE NOT NULL
);
```

```
INSERT INTO vertical (nombre) VALUES
('DS'),
('FS')
ON CONFLICT (nombre) DO NOTHING;
```

04 Creación de tablas y contenido

Una vez hecho esto, creamos nuestras tablas directamente en pgAdmin

Tabla promoción

```
CREATE TABLE promocion (
    id_promocion SERIAL PRIMARY KEY,
    nombre VARCHAR(20) UNIQUE NOT NULL,
    fecha_inicio DATE,
    id_campus INT NOT NULL,
    FOREIGN KEY (id_campus) REFERENCES campus(id_campus)
);

INSERT INTO promocion (nombre, fecha_inicio, id_campus) VALUES
('Septiembre', '2023-09-18', 1),
('Febrero', '2024-02-12', 1),
('Septiembre', '2023-09-18', 2),
('Febrero', '2024-02-12', 2)
ON CONFLICT (nombre) DO NOTHING;
```

Tabla modalidad

```
CREATE TABLE modalidad (
    id_modalidad SERIAL PRIMARY KEY,
    nombre VARCHAR(20) UNIQUE NOT NULL
);

INSERT INTO modalidad (nombre) VALUES
('Presencial'),
('Online')
ON CONFLICT (nombre) DO NOTHING;
```

Creación de tablas y contenido

Una vez hecho esto, creamos nuestras tablas directamente en pgAdmin

Tabla alumnos

```
CREATE TABLE alumnos (
    id_alumno SERIAL PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    email VARCHAR(150) NOT NULL,
    id_promocion INT NOT NULL,
    FOREIGN KEY (id_promocion) REFERENCES promocion(id_promocion)
);

INSERT INTO alumnos (nombre, email, id_promocion)
SELECT
    a."Nombre",
    a."Email",
    p.id_promocion
FROM alumnos_final a
JOIN promocion p ON a."Promoción" = p.nombre;
```

Tabla profesor

```
CREATE TABLE profesor (
    id_profesor SERIAL PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    email VARCHAR(150),
    rol VARCHAR(10) NOT NULL,
    id_modalidad INT,
    id_vertical INT,
    id_campus INT,
    FOREIGN KEY (id_modalidad) REFERENCES modalidad(id_modalidad),
    FOREIGN KEY (id_vertical) REFERENCES vertical(id_vertical),
    FOREIGN KEY (id_campus) REFERENCES campus(id_campus)
);

INSERT INTO profesor (nombre, email, rol, id_modalidad, id_vertical, id_campus)
SELECT
    c."Nombre",
    '' as email, -- No hay email en el CSV
    c."Rol",
    m.id_modalidad,
    v.id_vertical,
    cam.id_campus
FROM claustro_final c
JOIN modalidad m ON c."Modalidad" = m.nombre
JOIN vertical v ON c."Vertical" = v.nombre
JOIN campus cam ON c."Campus" = cam.nombre;
```

04 Creación de tablas y contenido

Una vez hecho esto, creamos nuestras tablas directamente en pgAdmin

Tabla proyecto

```
CREATE TABLE proyecto (
    id_proyecto SERIAL PRIMARY KEY,
    nombre VARCHAR(50) UNIQUE NOT NULL
);

INSERT INTO proyecto (nombre) VALUES
('Proyecto_HLF'),
('Proyecto_EDA'),
('Proyecto_BBDD'),
('Proyecto_ML'),
('Proyecto_Deployment'),
('Proyecto_WebDev'),
('Proyecto_FrontEnd'),
('Proyecto_Backend'),
('Proyecto_React'),
('Proyecto_FullStack')
ON CONFLICT (nombre) DO NOTHING;
```

Tabla evaluación

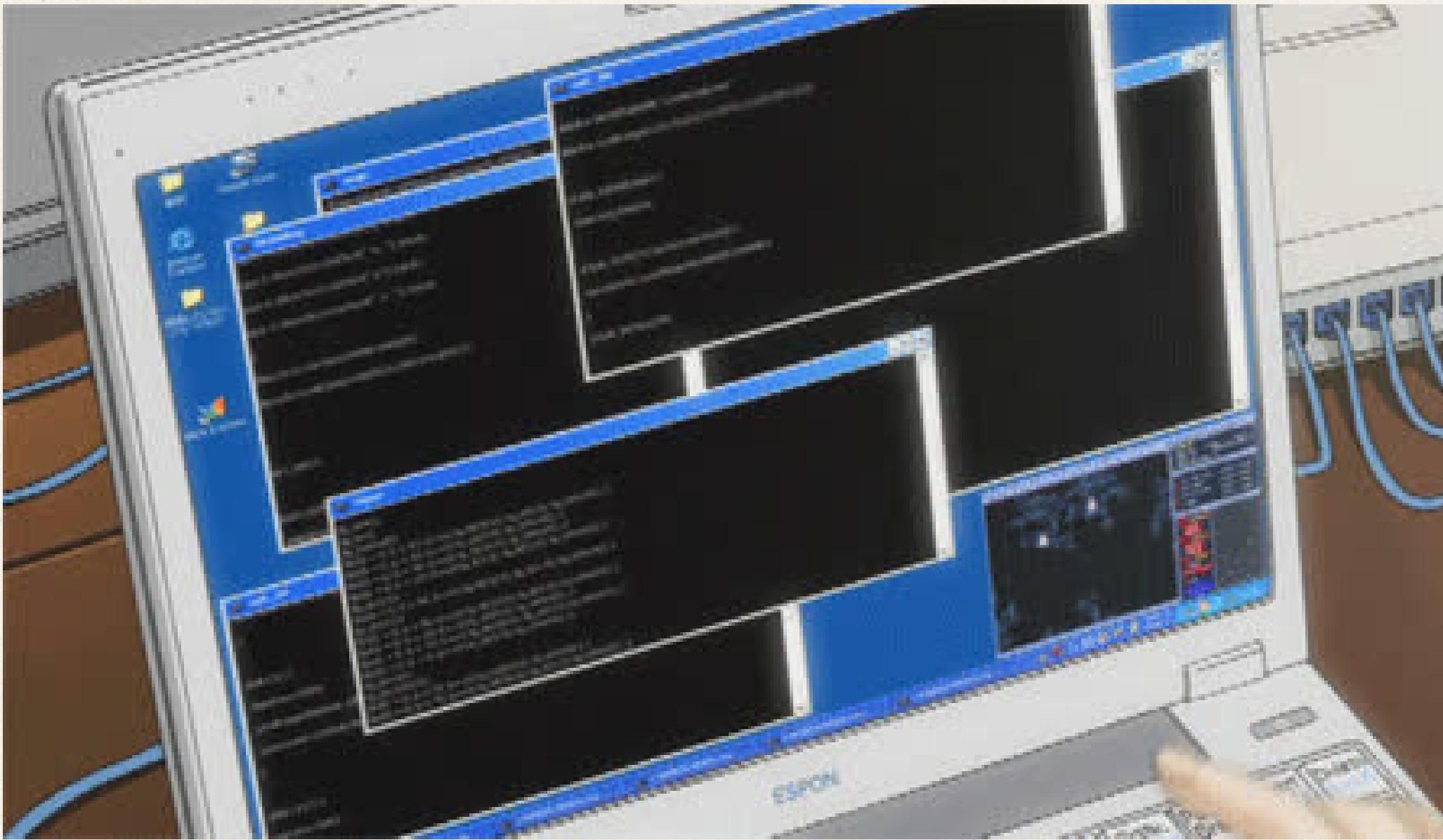
```
CREATE TABLE evaluacion (
    id_evaluacion SERIAL PRIMARY KEY,
    id_proyecto INT NOT NULL,
    id_alumno INT NOT NULL,
    resultado VARCHAR(10) CHECK (resultado IN ('APTO', 'NO APTO')),
    FOREIGN KEY (id_proyecto) REFERENCES proyecto(id_proyecto),
    FOREIGN KEY (id_alumno) REFERENCES alumnos(id_alumno)
);

INSERT INTO evaluacion (id_proyecto, id_alumno, resultado)
SELECT p.id_proyecto, a.id_alumno,
       CASE WHEN af."PROYECTO_NOMBRE" = 'Apto' THEN 'APTO' ELSE 'NO APTO' END
FROM alumnos a, alumnos_final af, proyecto p
WHERE a.nombre = af."Nombre" AND p.nombre = 'PROYECTO_NOMBRE'
AND af."PROYECTO_NOMBRE" IS NOT NULL;
```

05

Queries

Y ahora... ¡¡MANOSALTURÓN!!



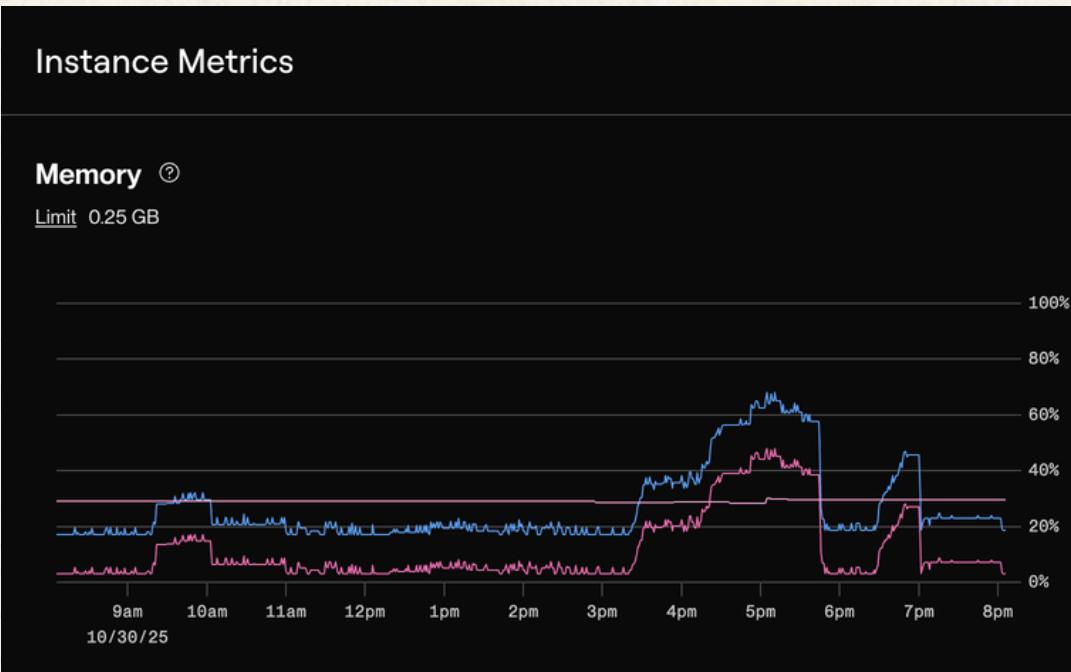
06

Funcionalidades futuras

- Reforzar las estructuras de las tablas para mejorar escalabilidad
- Pruebas automatizadas de integridad: validaciones para asegurar que las restricciones de integridad siguen funcionando después de cambios.
- Dashboards con herramientas como Metabase, Power BI, o Grafana conectadas a PostgreSQL.

07

Consultas en milisegundos, bajo consumo de recursos



Top Queries

QUERY 5 CALLS ↓ TOTAL ROWS TOTAL TIME TIME PER CALL

/*pga4dash*/ SELECT \$1 AS chart_name, pg_catalog.row_to_json...	5576	27880	8.1s	1.45ms
/*pga4dash*/ SELECT \$1 AS chart_name, pg_catalog.row_to_json...	3489	17445	4.76s	1.36ms
SELECT rel.oid, rel.relname AS name, rel.reltablename AS sp...	54	54	313.56ms	5.81ms
T * M T , , , e M t N d E) D) e D E e) 2 0 8 6 y 1	44	7128	306.97ms	6.98ms
, e 5 7 9 e , , E N E) 5 D 9 l (T E 1 3 5 6 , E 1 3 5 6 ...	2	11	806.39ms	403.2ms

1 – 5 of 50 items

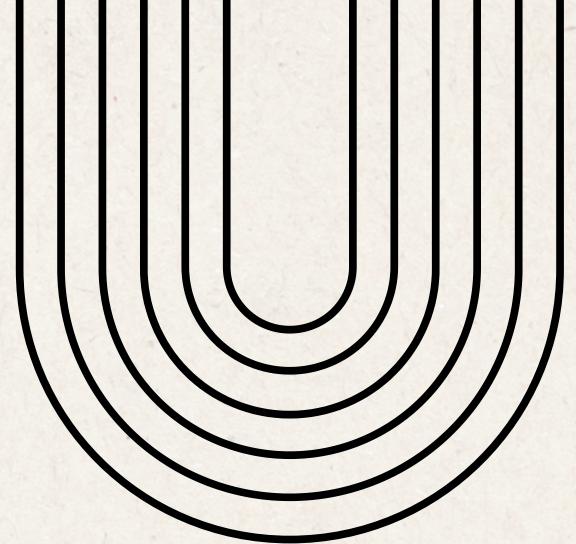
Table Sizes

DATABASE 5 SCHEMA TABLE SIZE ↓

database_project_098h	public	basura_1	16 KB
database_project_098h	public	basura_2	16 KB
database_project_098h	public	campus	8 KB
database_project_098h	public	promocion	8 KB
database_project_098h	public	vertical	8 KB

1 – 5 of 10 items

- Nuestro servidor PostgreSQL en Render mantiene el uso de memoria estable y eficiente (máx. 60 %), lo que demuestra un rendimiento consistente.
- El tamaño total de las tablas es mínimo (<100 KB), demostrando un modelo de datos limpio y sin redundancia, lo que contribuye a la rapidez de las consultas.
- Las consultas más ejecutadas tienen tiempos de respuesta inferiores a 2 ms, lo que demuestra una optimización excelente del motor y los índices.



GRACIAS

