Universidad Icesi

Bases de Datos Relacionales, Diseño & TypeORM en NestJS

Desarrollo de Entornos Digitales Web — Semana 3

Facultad de Ingeniería — Universidad Icesi

Universidad ICESI

Persistencia ACID

Consultas SQL

Relaciones Entidad

Persistencia de Datos en Aplicaciones Web

Una base de datos relacional es el núcleo donde las aplicaciones guardan información crítica de manera estructurada, concurrente y duradera (ACID).

- **Memoria vs Disco:** Evita la pérdida de datos cuando el servidor Node/NestJS se reinicia.
- **Estructura Tabular:** Organiza la información en entidades claramente definidas con tipos estrictos.
- **Relaciones Eficientes:** Permite consultar información cruzada mediante llaves sin duplicar datos.

Tablas: Filas, Columnas y Entidades

Conceptos Clave

En una BD relacional, cada **tabla** representa una entidad del mundo real (ej. Usuarios, Pedidos, Productos).

- **Columnas (Campos):** Definen los atributos y el tipo de dato permitido.
- **Filas (Registros):** Cada elemento individual guardado en la tabla.

EJEMPLO: TABLA `USUARIO` Y `PEDIDO`

ID (PK)	Nombre	Email
1	Ana Gómez	ana@correo.com
2	Luis Silva	luis@correo.com

ID (PK)	Producto	Total	Usuario_ID (FK)
100	Teclado	$45.00	1
101	Monitor	$150.00	2

Universidad ICESI

Primary Key (PK)

Foreign Key (FK)

Conectando Tablas: Llaves PK y FK

Las llaves garantizan que las tablas se conecten de forma unívoca y sin confusiones ni duplicados.

Primary Key (PK) — Llave Principal

Identificador único e irrepetible de cada fila en su propia tabla (ej. `id: 1`).

Foreign Key (FK) — Llave Foránea

Columna que almacena la PK de otra tabla para enlazar ambos registros (ej. `usuario_id` en Pedido).

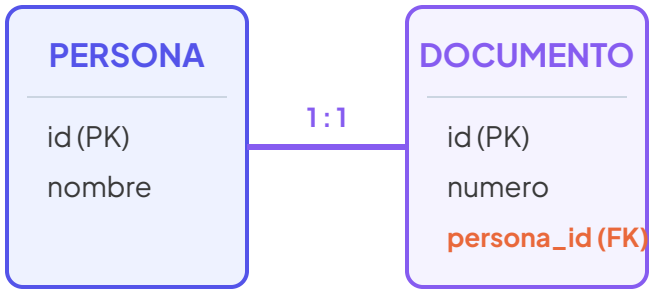
Relación 1:1 (Uno a Uno)

Concepto 1:1

Ocurre cuando un registro de la **Tabla A** se relaciona exactamente con un único registro de la **Tabla B**, y viceversa.

Ejemplo Real:

Una *Persona* tiene un solo *Documento de Identidad*, y ese documento pertenece a una sola persona.



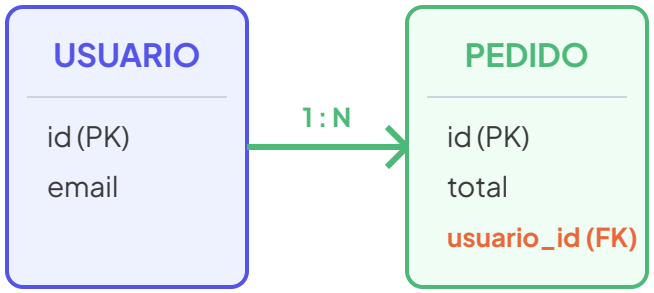
Relación 1:N (Uno a Muchos)

La Relación Más Común

Un registro de la **Tabla A** puede asociarse con varios registros de la **Tabla B**, pero cada registro de B pertenece a uno solo de A.

Ejemplo Real:

Un *Usuario* puede realizar cero o muchos *Pedidos*, pero cada pedido pertenece a un único usuario.

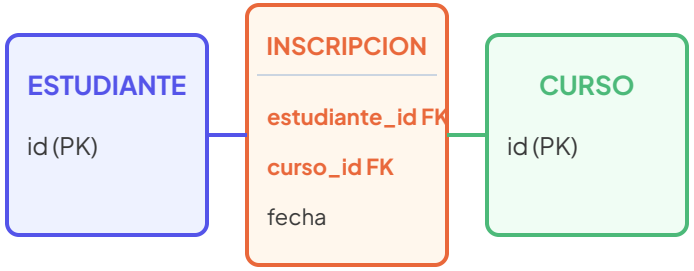


Relación N:M y Tabla Intermedia

Muchos a Muchos (N:M)

Ocurre cuando múltiples filas de A se conectan con múltiples filas de B.

Regla de Oro:
En SQL relacional **NUNCA** se conectan dos tablas N:M directamente. Se crea obligatoriamente una **Tabla Puente (Intermedia)**.



Universidad ICESI

Tipos de Datos en Columnas SQL

INT / BIGINT

Números enteros sin decimales. Utilizados para identificadores únicos (IDs), cantidades o edades.

VARCHAR / TEXT

Cadenas de texto. `VARCHAR(N)` limita la longitud (nombres/emails) y `TEXT` almacena textos extensos.

DATE / TIMESTAMP

Fechas e instantes de tiempo. Indispensable para auditar cuándo se creó o modificó un registro (`created_at`).

BOOLEAN / DECIMAL

`BOOLEAN` para estados (true/false) y `DECIMAL(10,2)` para precios exactos sin errores de redondeo.

Motores RDBMS de la Industria

POSTGRESQL

El motor del curso. Código abierto, robusto, excelente soporte JSON y estándar en startups y Enterprise.

MYSQL / MARIADB

Históricamente el más popular en la web tradicional (WordPress, LAMP stack). Muy veloz para lecturas.

SQLITE

Basado en un único archivo plano. Ideal para desarrollo local rápido, prototipos y aplicaciones móviles.

CRITERIOS DE ELECCIÓN

Integridad referencial estricta, alta concurrencia de escrituras y ecosistema de herramientas de desarrollo.

Universidad ICESI

Reglas para Diseñar una Buena BD

Principios de diseño relacional y normalización

Universidad ICESI

Redundancia

Celdas Nulas

Datos Mezclados

Diagnóstico: Síntomas de una BD Enferma

Analicemos una tabla mal diseñada (`CONTACTOS`) y sus consecuencias en desarrollo:

id	nombre_completo	telefono1	telefono2	ciudad_pais
1	Ana Gómez	300123	NULL	Medellín, Colombia
2	Juan Pérez	310456	320789	Medellín, Colombia

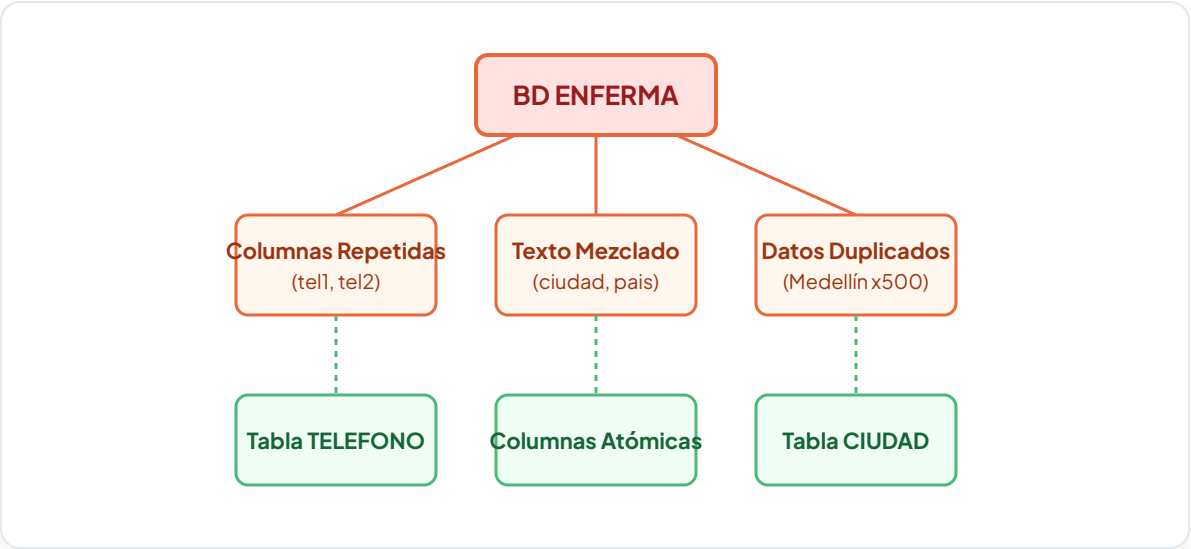
- **Redundancia:** "Medellín, Colombia" se repite en cientos de filas.
- **Celdas nulas vacías:** Columnas `telefono2` sin usar malgastan espacio.
- **Datos mezclados:** Imposible filtrar solo por país sin usar SQL complejo.

Diagnóstico y Remedios en BD

Síntomas vs Soluciones

Un buen diseño de base de datos aplica reglas de normalización para curar estos problemas:

- ✓ **Datos repetidos:** Se mueven a una tabla independiente.
- ✓ **Textos compuestos:** Se dividen en columnas atómicas.
- ✓ **Columnas repetidas:** Se convierten en filas relacionales.



Universidad ICESI

Cero Redundancia

Tabla Dedicada

Regla 1: "Un Dato, Una Sola Vez"

Cada pieza de información debe residir en exactamente **un único lugar** de la base de datos.

Antes vs Después:

Antes: El texto "Medellín" repetido en 500 filas de contactos.

Después: Tabla `CIUDAD` independiente. El contacto solo guarda `ciudad_id: 1`.

Universidad ICESI

Valores Atómicos

Fácil Filtrado SQL

Regla 2: "Cada Columna Guarda UNA Cosa"

Una columna debe contener un valor atómico e indivisible. Nunca mezcles conceptos en la misma celda.

Refactorización:

Incorrecto: `nombre_ciudad_pais` = "Medellín, Colombia"

Correcto: Columna `nombre_ciudad` y columna `nombre_pais` por separado.

Regla 3: No Más Columnas del Mismo Tipo

⚠️ Anti-Patrón: Columnas Rígidas

Definir múltiples columnas para el mismo concepto:

Entidad: PERSONA
- id: Entero (PK)
- nombre: Texto
- telefono1 : Texto
- telefono2 : Texto (Nulo vació)
- telefono3 : Texto (Nulo vació)

❌ Limita a máximo 3 teléfonos y desperdicia espacio.

✅ Diseño Relacional: 1 a Muchos (1:N)

Mover los números a una tabla relacional independiente:

Entidad: PERSONA
- id: Entero (PK)
- nombre: Texto

↓ (1 : N)

Entidad: TELEFONO
- id: Entero (PK)
- persona_id: Clave Foránea (FK)
- numero: Texto

✓ Permite N teléfonos dinámicos por persona sin tocar el esquema.

Universidad ICESI

Cohesión con PK

Aislamiento de Entidades

Regla 4: Dependencia Directa de la PK

Toda columna de una tabla debe describir directamente al sujeto principal identificado por la Primary Key, y nada más.

Ejemplo de Error:

En la tabla `LIBRO`, guardar `fecha_nacimiento_autor` es un error. Esa fecha describe al *Autor*, no al libro. Debe estar en la tabla `AUTOR`.

Proceso de Refactorización (Pasos 1–4)

PASO 1: TABLA PLANA

Escribir toda la información necesaria en una primera versión de borrador sin juzgar.

PASO 2: DETECTAR REPETIDOS

Identificar valores duplicados fila a fila y moverlos a tablas independientes.

PASO 3: DIVIDIR MEZCLAS

Separar celdas compuestas en columnas atómicas (ej. ciudad y país).

PASO 4: ELIMINAR GRUPOS

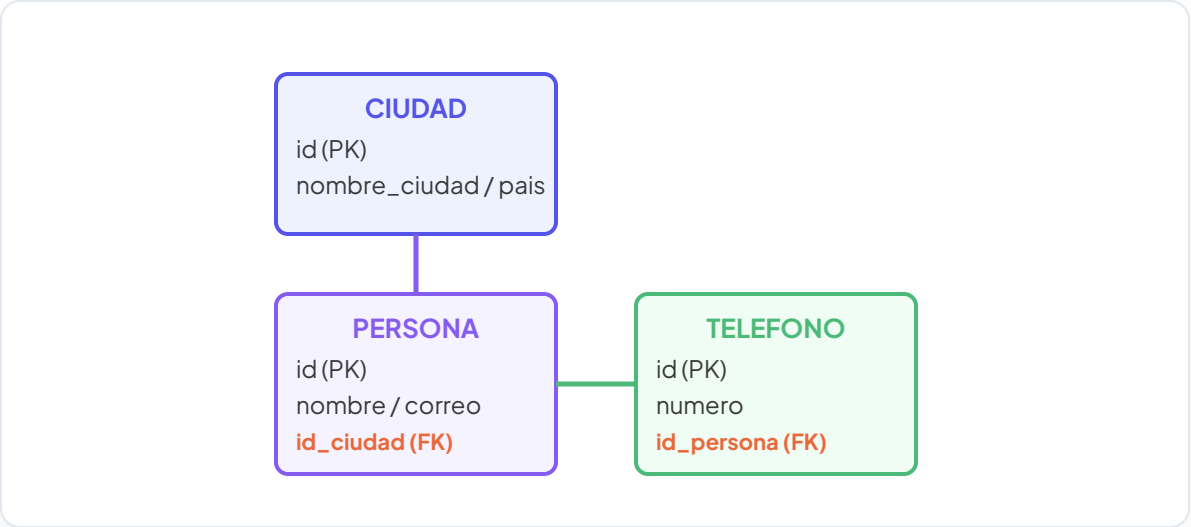
Convertir columnas numeradas (`tel1` , `tel2`) en filas de una tabla relacional.

Resultado Final: Esquema Normalizado

De 1 a 3 Tablas Limpias

Pasamos de una tabla caótica con 8 columnas a tres tablas con responsabilidades bien separadas y cero datos duplicados.

- ✓ `CIUDAD` (Catálogo de ciudades y países).
- ✓ `PERSONA` (Datos personales + `id_ciudad`).
- ✓ `TELEFONO` (N números asociados a persona).



Anti-Patrones Comunes a Evitar

ARRAYS EN TEXTO

Guardar `"1,4,7"` en una columna de texto en vez de usar una tabla intermedia N:M.

NOMBRES AMBIGUOS

Nombrar columnas como `data` o `value` en lugar de identificadores semánticos claros.

FALTA DE FKS

Confiar solo en la aplicación para mantener la consistencia sin declarar restricciones SQL.

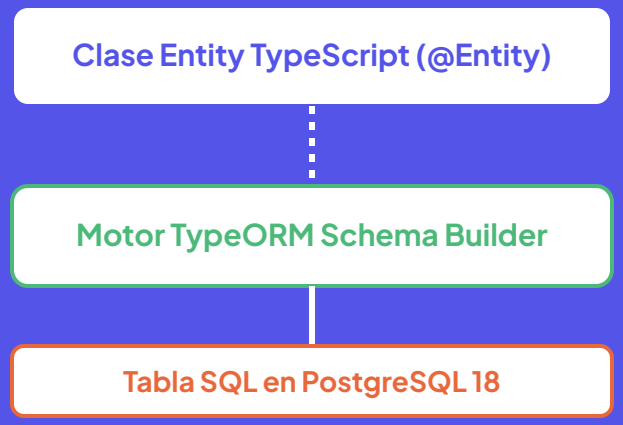
CARENCIA DE PKS

Crear tablas sin clave primaria autoincremental o UUID para identificar registros.

Universidad ICESI

Implementación en NestJS con TypeORM

Traduciendo diseños relacionales a código TypeScript con PostgreSQL 18



Object-Relational Mapping (ORM)

TypeORM es la librería que mapea nuestras clases de TypeScript a tablas relacionales de PostgreSQL automáticamente.

- ✓ Sin SQL crudo manual (`CREATE TABLE...`).
- ✓ TypeScript estricto con autocompletado e Inferencia de tipos.

Mapeo de Conceptos ER a TypeORM

En el Diagrama ER

- Entidad (Tabla)
- Llave Primaria autoincremental
- Atributo / Columna
- Relación 1:N (Uno a Muchos)
- Relación N:1 (Muchos a Uno)
- Nombre de la Foreign Key

En TypeORM (TypeScript)

- `@Entity('nombre_tabla')`
- `@PrimaryGeneratedColumn()`
- `@Column({ options })`
- `@OneToMany(() => Target, ...)`
- `@ManyToOne(() => Target, ...)`
- `@JoinColumn({ name: 'fk_id' })`

Universidad ICESI

Docker Postgres 18

ConfigModule (.env)

TypeOrmModule

Arquitectura de Conexión NestJS

La integración une el framework NestJS con PostgreSQL usando variables centralizadas en `.env`.

- ✓ `ConfigModule` lee variables de entorno de forma global.
- ✓ `TypeOrmModule.forRootAsync` inyecta dinámicamente credenciales al arrancar.

Universidad ICESI

npm install

@nestjs/typeorm

pg driver

Instalación de Dependencias

Instalamos el módulo de TypeORM para NestJS, el núcleo de TypeORM, el gestor de configuración y el driver oficial de PostgreSQL:

```
npm install --save @nestjs/typeorm typeorm @nestjs/config pg
```

* `pg` es el cliente nativo que permite a Node.js comunicarse con el protocolo de PostgreSQL.

Configuración de Entorno (.env)

Centralizando Credenciales

Nunca escribas contraseñas directamente en el código TypeScript. Usa el archivo `.env` en la raíz del proyecto.

Este archivo servirá tanto para NestJS como para Docker Compose.

ARCHIVO `.ENV`

```
# Configuración de Base de Datos
DB_HOST=localhost
DB_PORT=5437
DB_USERNAME=postgres
DB_PASSWORD=postgres
DB_DATABASE=nest_db
```

Contenedor PostgreSQL 18

Docker Compose

Levantamos la base de datos en un contenedor aislado con PostgreSQL v18 parametrizado desde `.env`.

```
# Iniciar contenedor DB
docker compose up -d
```

ARCHIVO `DOCKER-COMPOSE.YML`

```
services:
 db:
  image: postgres:18
  restart: always
  environment:
   POSTGRES_USER: ${DB_USERNAME}
   POSTGRES_PASSWORD: ${DB_PASSWORD}
   POSTGRES_DB: ${DB_DATABASE}
  ports:
   - '${DB_PORT}:5432'
  volumes:
   - postgres-data:/var/lib/postgresql/data
```

TypeOrmModule.forRootAsync

Configuración en NestJS

Usamos `forRootAsync` para esperar a que `ConfigModule` lea las variables del `.env`.

> **Importante:**
>
> `synchronize: true` crea las tablas automáticamente en desarrollo. ¡Desactivar en producción!

```
@Module({
 imports: [
  ConfigModule.forRoot({ isGlobal: true }),
  TypeOrmModule.forRootAsync({
   imports: [ConfigModule],
   inject: [ConfigService],
   useFactory: (config: ConfigService) => ({
    type: 'postgres',
    host: config.get<string>('DB_HOST'),
    port: config.get<number>('DB_PORT'),
    username: config.get<string>('DB_USERNAME'),
    password: config.get<string>('DB_PASSWORD'),
    database: config.get<string>('DB_DATABASE'),
    entities: [__dirname + '/**/*.entity{.ts,.js}'],
    synchronize: true,
   }),
  }),
 ],
})
export class AppModule {}
```

Definición de Entidad y Llave Primaria

@Entity('nombre')

Fija el nombre físico exacto de la tabla en PostgreSQL. Si no se especifica, usa el nombre de la clase en minúsculas.

```
@Entity('roles')
export class Role { ... }
```

@PrimaryGeneratedColumn()

Genera la clave primaria de la tabla.

- ✓ `'increment'`: Entero autoincremental (default).
- ✓ `'uuid'`: Identificador único universal de 128 bits.

Opciones Avanzadas de @Column()

TYPE & LENGTH

Especifica el tipo SQL (`varchar` , `int` , `text`) y longitud máxima: `@Column({ type: 'varchar', length: 100 })`

UNIQUE: TRUE

Aplica una restricción de unicidad en la base de datos. Ideal para correos o usernames: `@Column({ unique: true })`

NULLABLE: TRUE

Permite que la columna acepte valores nulos (`NULL`). Por defecto todas las columnas son obligatorias (`false`).

DEFAULT: VALOR

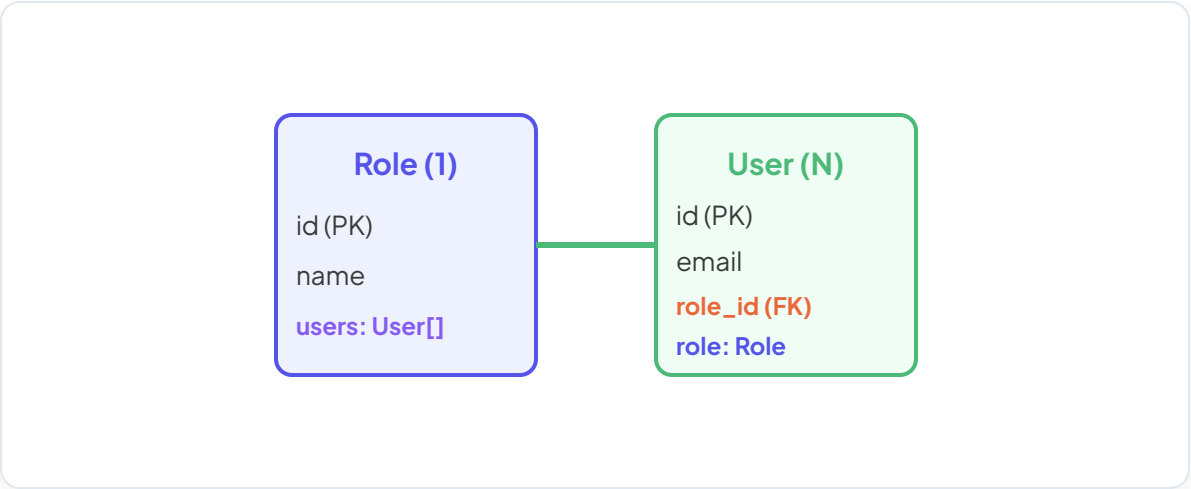
Establece un valor por defecto si no se proporciona uno al crear el registro: `@Column({ default: true })`

Relaciones 1:N y N:1 en Código

Caso: Rol y Usuario

Un **Rol** tiene muchos usuarios (`@OneToMany`). Un **Usuario** pertenece a un solo rol (`@ManyToOne`).

> **Regla TypeORM:**
> La entidad que lleva la clave foránea (`User`) usa `@ManyToOne` y `@JoinColumn` .



Entidad Role (role.entity.ts)

Entidad Principal (1)

La clase `Role` define la relación inversa `@OneToMany`. Retorna el arreglo de usuarios asociados.

```ts
import { Entity, PrimaryGeneratedColumn, Column, OneToMany } from 'typeorm';
import { User } from '../../users/entities/user.entity';

@Entity('roles')
export class Role {
 @PrimaryGeneratedColumn()
 id: number;

 @Column({ type: 'varchar', length: 50, unique: true })
 name: string;

 // Un rol se relaciona con muchos usuarios
 @OneToMany(() => User, (user) => user.role)
 users: User[];
}
```

Entidad User (user.entity.ts)

Entidad Propietaria de FK (N)

`User` lleva la decoración `@ManyToOne` y `@JoinColumn({ name: 'role_id' })` para crear la columna física FK.

```ts
import { Entity, PrimaryGeneratedColumn, Column, ManyToOne, JoinColumn } from 'typeorm';
import { Role } from '../../roles/entities/role.entity';

@Entity('users')
export class User {
 @PrimaryGeneratedColumn()
 id: number;

 @Column({ type: 'varchar', length: 120, unique: true })
 email: string;

 @Column({ type: 'varchar', length: 255 })
 passwordHash: string;

 // Muchos usuarios pertenecen a un rol
 @ManyToOne(() => Role, (role) => role.users, { nullable: false })
 @JoinColumn({ name: 'role_id' })
 role: Role;
}
```

Seeding de Datos Base (db/seed.sql)

Poblado de Datos Iniciales

Para registrar los roles iniciales (`ADMIN` , `USER`) y un usuario administrador por defecto creamos el script SQL en `db/seed.sql` .

ARCHIVO `DB/SEED.SQL`

```sql
-- Insertar roles base si no existen
INSERT INTO roles (name)
VALUES ('ADMIN'), ('USER'), ('GUEST')
ON CONFLICT (name) DO NOTHING;

-- Insertar usuario Administrador por defecto
INSERT INTO users (email, "passwordHash", role_id)
VALUES (
 'admin@docukelo.edu.co',
 'secret_hashed_password',
 (SELECT id FROM roles WHERE name = 'ADMIN')
)
ON CONFLICT (email) DO NOTHING;
```

Universidad ICESI

docker compose exec

psql client

seed.sql

Ejecución del Seed en Contenedor

Ejecutamos el script SQL directamente dentro del contenedor de PostgreSQL con una sola línea de comando:

```
docker compose exec -T db psql -U postgres -d nest_db < db/seed.sql
```

- ✓ `-T` : Desactiva asignación TTY para permitir redirección de entrada.
- ✓ `psql` : Abre el cliente nativo de PostgreSQL dentro del servicio `db` .