Diseño de una base de datos

Proyecto final

Integrantes

Baños Monjaraz José Guadalupe

Blanco Pulido Gabriel Alonzo

Cruces Díaz Monserrat Margarita

Rojas Uribe Mario

Grupo 05

Profesor: Dr. Oscar Arana Hernández

25 de mayo del 2025

Índice

Introducción

En este proyecto se desarrolló un sistema de bases de datos relacional enfocado en la gestión de recursos humanos dentro de la UNAM, aplicando todos los conocimientos adquiridos a lo largo del curso. A lo largo del trabajo se llevaron a cabo distintas actividades que abarcan desde el análisis de requerimientos hasta la implementación práctica en PostgreSQL.

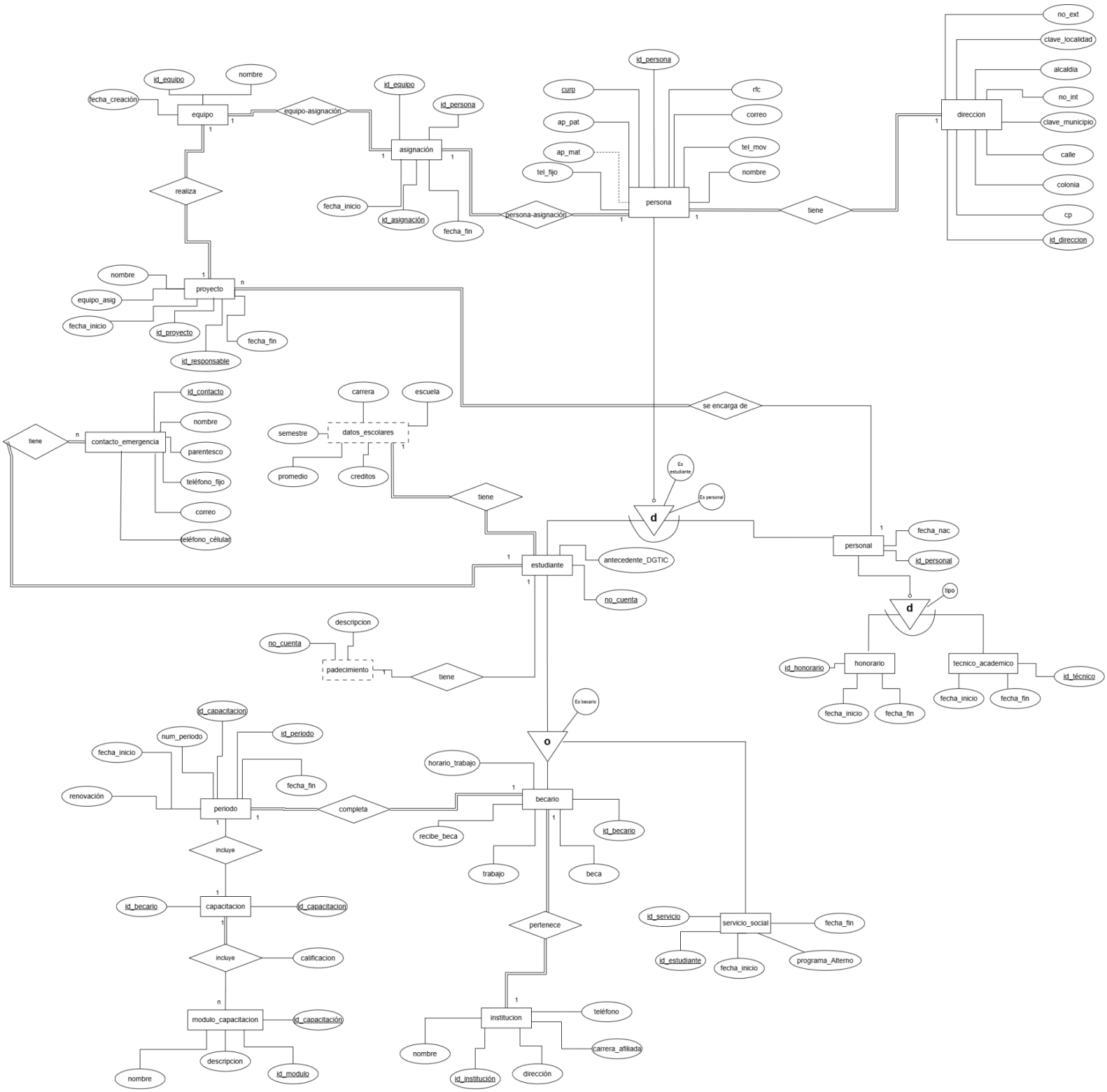
El proceso comenzó con el diseño de un modelo Entidad-Relación que representa de forma clara y precisa las relaciones entre becarios, estudiantes de servicio social, personal de honorarios y técnicos académicos. Posteriormente, este modelo conceptual fue normalizado hasta la Tercera Forma Normal (3FN), con el fin de asegurar integridad y consistencia en los datos.

Como parte del diseño lógico, se implementaron *triggers* para validar reglas de negocio específicas, como evitar que una misma persona tenga más de un rol institucional al mismo tiempo. También se crearon vistas estratégicas para facilitar consultas frecuentes. Finalmente, se generaron 500 registros de prueba, lo cual permitió evaluar el rendimiento del sistema, hacer ajustes y asegurar su correcto funcionamiento.

En conjunto, este proyecto no solo demuestra la comprensión teórica de los temas vistos en clase, sino también la capacidad de aplicarlos en la construcción de una solución real y funcional. Cada una de las actividades fue realizada con el objetivo de obtener un sistema robusto, bien estructurado y adaptado a las necesidades del entorno educativo.

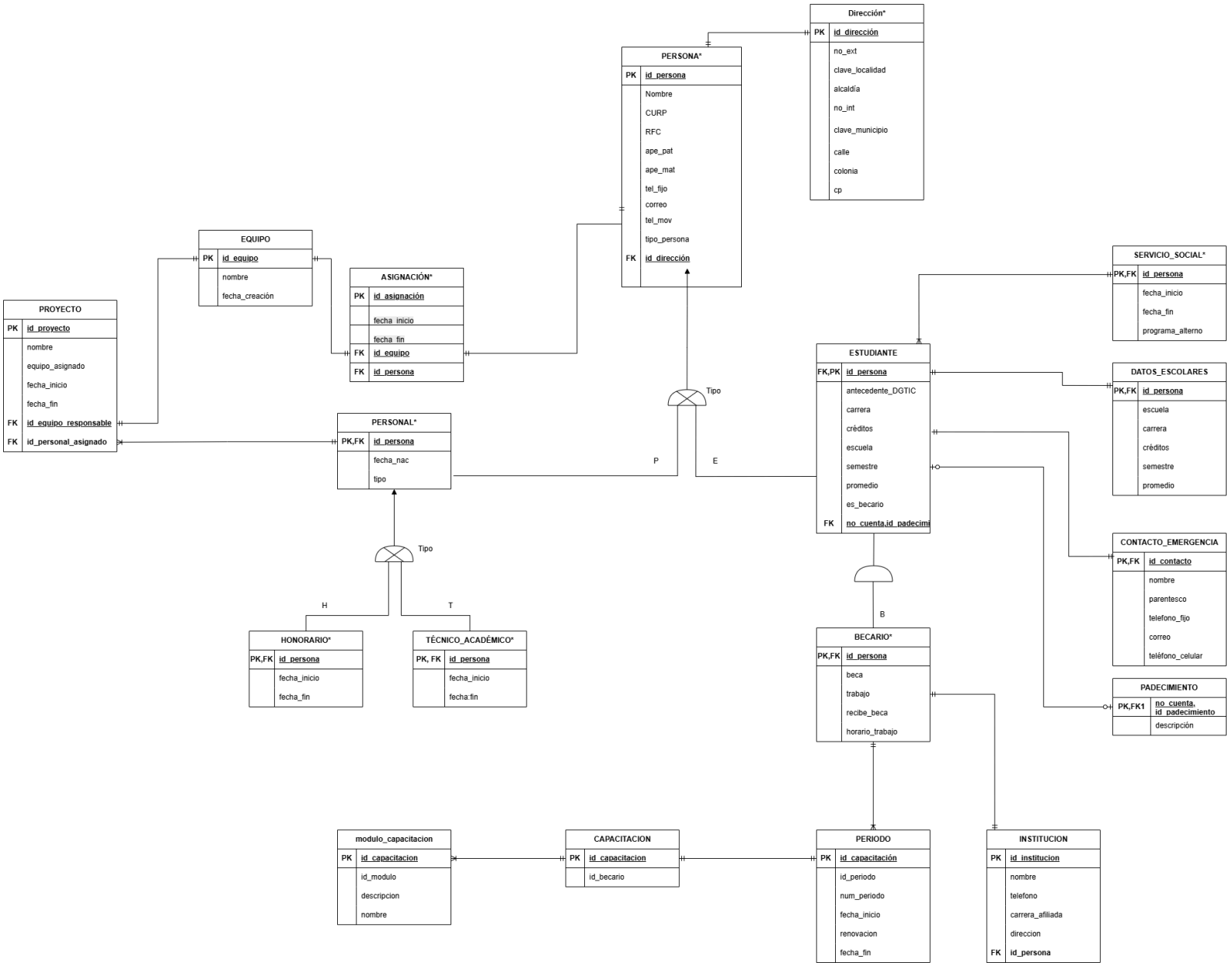
Modelo Entidad - Relación

En el modelo Entidad-Relación desarrollado, se señaló cuidadosamente la cardinalidad de cada relación para reflejar con precisión las restricciones del sistema. Además, se optó por un diseño con especialización jerárquica de tres niveles, lo cual permitió organizar de manera más clara y lógica los distintos tipos de actores institucionales, facilitando tanto la normalización del modelo como su futura implementación.


id_equipo
nombre
fecha_creación
equipo
equipo-asignación
id_equipo
id_persona
asignación
fecha_inicio
id_asignación
fecha_fin
persona-asignación
id_persona
curp
rfc
correo
ap_pat
ap_mat
tel_mov
tel_fijo
nombre
persona
tiene
direccion
no_ext
clave_localidad
alcaldia
no_int
clave_municipio
calle
colonia
cp
id_direccion
realiza
nombre
equipo_asig
fecha_inicio
proyecto
id_proyecto
fecha_fin
id_responsable
se encarga de
id_contacto
nombre
parentesco
teléfono_fijo
correo
teléfono_célular
tiene
contacto_emergencia
carrera
escuela
semestre
datos_escolares
promedio
creditos
tiene
Es estudiante
Es personal
d
estudiante
antecedente_DGTIC
no_cuenta
personal
fecha_nac
id_personal
tipo
d
id_honorario
honorario
fecha_inicio
fecha_fin
tecnico_academico
id_técnico
fecha_inicio
fecha_fin
descripcion
no_cuenta
padecimiento
tiene
Es becario
o
becario
horario_trabajo
recibe_beca
id_becario
trabajo
beca
id_capacitacion
num_periodo
id_periodo
fecha_inicio
fecha_fin
renovación
periodo
completa
incluye
id_becario
capacitacion
id_capacitacion
incluye
calificacion
modulo_capacitacion
id_capacitación
nombre
descripcion
id_modulo
pertenece
id_servicio
servicio_social
fecha_fin
id_estudiante
fecha_inicio
programa_Alterno
institucion
teléfono
nombre
carrera_afiliada
id_institución
dirección

Modelo Relacional

El modelo relacional fue construido a partir del modelo Entidad-Relación, respetando las estructuras y restricciones previamente definidas. Se utilizó la notación de pie de cuervo (Crow 's Foot) para representar las relaciones entre las tablas, lo que permite visualizar de forma clara la cardinalidad y las dependencias entre entidades. Este modelo sirvió como base para aplicar las reglas de normalización y garantizar una estructura lógica coherente y eficiente para la posterior implementación en el sistema gestor de bases de datos.



Implementación de la base de datos

A continuación se presenta la implementación de las tablas en PostgreSQL.

```sql
-- ====================
-- TABLAS PRINCIPALES
-- ====================

CREATE TABLE Direccion (
    id_direccion VARCHAR(5) PRIMARY KEY NOT NULL,
    no_ext VARCHAR(10) NOT NULL,
    clave_localidad VARCHAR(20) NOT NULL,
    alcaldia VARCHAR(50) NOT NULL,
    no_int VARCHAR(10),
    clave_municipio VARCHAR(20) NOT NULL,
    calle VARCHAR(100) NOT NULL,
    colonia VARCHAR(100) NOT NULL,
    cp VARCHAR(10) NOT NULL
);

CREATE TABLE Persona (
    id_persona VARCHAR(5) PRIMARY KEY NOT NULL,
    nombre VARCHAR(100) NOT NULL,
    curp VARCHAR(18) NOT NULL,
    rfc VARCHAR(13) NOT NULL,
    ape_pat VARCHAR(50) NOT NULL,
    ape_mat VARCHAR(50) NOT NULL,
    tel_fijo VARCHAR(15) NOT NULL,
    correo VARCHAR(100) NOT NULL,
    tel_mov VARCHAR(15) NOT NULL,
    tipo_persona CHAR(1) NOT NULL CHECK (tipo_persona IN ('P', 'E')), -- 'P': personal, 'E': estudiante
    id_direccion VARCHAR(5) NOT NULL REFERENCES Direccion(id_direccion)
);
```

```sql
-- ====================
-- PADECIMIENTO (movida antes de Estudiante porque es referenciada)
-- ====================

CREATE TABLE Padecimiento (
    no_cuenta VARCHAR(9) NOT NULL,
    id_padecimiento VARCHAR(5) NOT NULL,
    descripcion TEXT NOT NULL,
    PRIMARY KEY(no_cuenta, id_padecimiento)
);

-- ====================
-- ESTUDIANTE Y BECARIO
-- ====================

CREATE TABLE Estudiante (
    id_persona VARCHAR(5) PRIMARY KEY REFERENCES Persona(id_persona),
    antecedente_DGTIC VARCHAR(100) NOT NULL,
    carrera VARCHAR(100) NOT NULL,
    cvu INT NOT NULL,
    escuela VARCHAR(100) NOT NULL,
    semestre INT NOT NULL,
    promedio NUMERIC(3,2) NOT NULL,
    es_becario BOOLEAN NOT NULL,
    no_cuenta VARCHAR(9) NOT NULL,
    id_padecimiento VARCHAR(5) NOT NULL,
    FOREIGN KEY (no_cuenta, id_padecimiento) REFERENCES Padecimiento(no_cuenta, id_padecimiento)
);
```

```sql
CREATE TABLE Becario (
    id_persona VARCHAR(5) NOT NULL PRIMARY KEY REFERENCES Estudiante(id_persona),
    beca VARCHAR(100) NOT NULL,
    trabajo VARCHAR(100) NOT NULL,
    recibe_beca BOOLEAN NOT NULL,
    horario_trabajo VARCHAR(100) NOT NULL
);
```

```sql
-- ====================
-- DATOS ESCOLARES, CONTACTO Y PADECIMIENTO
-- ====================

CREATE TABLE Datos_Escolares (
    id_persona VARCHAR(5) NOT NULL PRIMARY KEY REFERENCES Estudiante(id_persona),
    escuela VARCHAR(100) NOT NULL,
    carrera VARCHAR(100) NOT NULL,
    creditos INT,
    semestre INT NOT NULL,
    promedio NUMERIC(3,2) NOT NULL
);

CREATE TABLE Contacto_Emergencia (
    id_contacto VARCHAR(5) NOT NULL,
    id_persona VARCHAR(5) NOT NULL REFERENCES Persona(id_persona),
    nombre VARCHAR(100) NOT NULL,
    parentesco VARCHAR(50) NOT NULL,
    telefono_fijo VARCHAR(15) NOT NULL,
    correo VARCHAR(100) NOT NULL,
    telefono_celular VARCHAR(15) NOT NULL,
    PRIMARY KEY(id_contacto, id_persona)
);
```

```sql
-- ====================
-- PERSONAL Y ESPECIALIZACIONES
-- ====================

CREATE TABLE Personal (
    id_persona VARCHAR(5) NOT NULL PRIMARY KEY REFERENCES Persona(id_persona),
    fecha_nac DATE NOT NULL,
    tipo CHAR(1) NOT NULL CHECK (tipo IN ('H', 'T')) -- 'H' Honorario, 'T' Técnico Académico
);

CREATE TABLE Honorario (
    id_persona VARCHAR(5) NOT NULL PRIMARY KEY REFERENCES Personal(id_persona),
    fecha_inicio DATE NOT NULL,
    fecha_fin DATE NOT NULL
);

CREATE TABLE Tecnico_Academico (
    id_persona VARCHAR(5) NOT NULL PRIMARY KEY REFERENCES Personal(id_persona),
    fecha_inicio DATE NOT NULL,
    fecha_fin DATE NOT NULL
);
```

```sql
-- ====================
-- SERVICIO SOCIAL
-- ====================

CREATE TABLE Servicio_Social (
    id_persona VARCHAR(5) NOT NULL PRIMARY KEY REFERENCES Persona(id_persona),
    fecha_inicio DATE NOT NULL,
    fecha_fin DATE NOT NULL,
    programa_alterno VARCHAR(100) NOT NULL
);

-- ====================
-- EQUIPO Y PROYECTO
-- ====================

CREATE TABLE Equipo (
    id_equipo VARCHAR(5) NOT NULL PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    fecha_creacion DATE NOT NULL
);

CREATE TABLE Proyecto (
    id_proyecto VARCHAR(5) NOT NULL PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    fecha_inicio DATE NOT NULL,
    fecha_fin DATE NOT NULL,
    id_equipo_responsable VARCHAR(5) NOT NULL REFERENCES Equipo(id_equipo),
    id_personal_asignado VARCHAR(5) NOT NULL REFERENCES Personal(id_persona)
);
```

```sql
-- ====================
-- ASIGNACION DE PERSONAS A EQUIPOS
-- ====================

CREATE TABLE Asignacion (
    id_asignacion VARCHAR(5) NOT NULL PRIMARY KEY,
    fecha_inicio DATE,
    fecha_fin DATE,
    id_equipo VARCHAR(5) NOT NULL REFERENCES Equipo(id_equipo),
    id_persona VARCHAR(5) NOT NULL REFERENCES Persona(id_persona)
);

-- ====================
-- INSTITUCION
-- ====================

CREATE TABLE Institucion (
    id_institucion SERIAL NOT NULL PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    telefono VARCHAR(15) NOT NULL,
    carrera_afiliada VARCHAR(100) NOT NULL,
    direccion TEXT NOT NULL,
    id_persona VARCHAR(5) NOT NULL REFERENCES Persona(id_persona)
);
```

```sql
-- ====================
-- CAPACITACION
-- ====================

CREATE TABLE Modulo_Capacitacion (
    id_capacitacion SERIAL NOT NULL PRIMARY KEY,
    id_modulo VARCHAR(50) NOT NULL,
    descripcion TEXT NOT NULL,
    nombre VARCHAR(100) NOT NULL
);

CREATE TABLE Periodo (
    id_capacitacion INT NOT NULL REFERENCES Modulo_Capacitacion(id_capacitacion),
    id_periodo SERIAL NOT NULL,
    num_periodo INT NOT NULL,
    fecha_inicio DATE NOT NULL,
    fecha_fin DATE NOT NULL,
    PRIMARY KEY(id_capacitacion, id_periodo)
);

CREATE TABLE Capacitacion (
    id_capacitacion INT NOT NULL REFERENCES Modulo_Capacitacion(id_capacitacion),
    id_becario VARCHAR(5) NOT NULL REFERENCES Becario(id_persona),
    PRIMARY KEY(id_capacitacion, id_becario)
);
```

Implementación de requerimientos

```sql
-- Modificación a la tabla Servicio_Social para registrar solo una vez por estudiante
ALTER TABLE Servicio_Social ADD COLUMN realizado BOOLEAN NOT NULL DEFAULT FALSE;

-- Modificación a la tabla Becario para llevar control de periodos
ALTER TABLE Becario ADD COLUMN fecha_fin_beca DATE;

CREATE OR REPLACE FUNCTION validar_servicio_social_unico()
RETURNS TRIGGER AS $$
DECLARE
    servicio_previo BOOLEAN;
BEGIN
    -- Verificar si el estudiante ya realizó servicio social
    SELECT realizado INTO servicio_previo
    FROM Servicio_Social
    WHERE id_persona = NEW.id_persona;

    IF servicio_previo THEN
        RAISE EXCEPTION 'Un estudiante solo puede realizar servicio social una vez';
    END IF;

    -- Marcar como realizado al insertar
    NEW.realizado := TRUE;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_servicio_social_unico
BEFORE INSERT ON Servicio_Social
FOR EACH ROW
EXECUTE FUNCTION validar_servicio_social_unico();
```

```sql
CREATE OR REPLACE FUNCTION validar_solapamiento_beca_servicio()
RETURNS TRIGGER AS $$
BEGIN
    -- No se necesita validación especial ya que el solapamiento está permitido
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_solapamiento_beca_servicio
BEFORE INSERT OR UPDATE ON Servicio_Social
FOR EACH ROW
EXECUTE FUNCTION validar_solapamiento_beca_servicio();

CREATE OR REPLACE FUNCTION validar_honorario_sin_beca()
RETURNS TRIGGER AS $$
DECLARE
    beca_activa BOOLEAN;
BEGIN
    -- Verificar si tiene beca activa
    SELECT TRUE INTO beca_activa
    FROM Becario
    WHERE id_persona = NEW.id_persona
    AND (fecha_fin_beca IS NULL OR fecha_fin_beca > CURRENT_DATE);

    IF beca_activa THEN
        RAISE EXCEPTION 'No se puede contratar como honorario mientras tenga una beca activa';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```sql
CREATE TRIGGER trg_honorario_sin_beca
BEFORE INSERT OR UPDATE ON Honorario
FOR EACH ROW
EXECUTE FUNCTION validar_honorario_sin_beca();
```

Configuración de roles y permisos.

```sql
--==================================
--Creación de roles
--==================================

-- Rol administrador con todos los privilegios
CREATE ROLE admin LOGIN PASSWORD 'admin_pass';
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO admin;
GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA public TO admin;

-- Rol operador: puede insertar y actualizar datos pero no borrar
CREATE ROLE operador LOGIN PASSWORD 'operador_pass';

-- Rol reportes: solo puede leer datos (SELECT)
CREATE ROLE reportes LOGIN PASSWORD 'reportes_pass';

--==================================
--Asignación de privilegios
--==================================

--Operador
GRANT SELECT, INSERT, UPDATE ON ALL TABLES IN SCHEMA public TO operador;
GRANT USAGE, SELECT ON ALL SEQUENCES IN SCHEMA public TO operador;

--Reportes
GRANT SELECT ON ALL TABLES IN SCHEMA public TO reportes;
```

```sql
--==================================
--Configuración
--==================================

-- Default para tablas
ALTER DEFAULT PRIVILEGES IN SCHEMA public
GRANT SELECT, INSERT, UPDATE ON TABLES TO operador;

ALTER DEFAULT PRIVILEGES IN SCHEMA public
GRANT SELECT ON TABLES TO reportes;

ALTER DEFAULT PRIVILEGES IN SCHEMA public
GRANT ALL PRIVILEGES ON TABLES TO admin;

-- Default para secuencias (ej: SERIAL IDs)
ALTER DEFAULT PRIVILEGES IN SCHEMA public
GRANT USAGE, SELECT ON SEQUENCES TO operador;

ALTER DEFAULT PRIVILEGES IN SCHEMA public
GRANT USAGE, SELECT ON SEQUENCES TO reportes;

ALTER DEFAULT PRIVILEGES IN SCHEMA public
GRANT ALL PRIVILEGES ON SEQUENCES TO admin;

```

Dependencias Funcionales

Para la tabla persona **id_persona** es la clave primaria y determina de forma única todos los demás atributos.

id_persona → nombre, CURP, RFC, app_pat, app_mat, tel_fijo, tel_mov, correo, tipo_persona, id_direccion

Para la tabla **ESTUDIANTE** se hereda desde **PERSONA** por lo tanto id_persona también es PK y FK.

Para la tabla becario como id_persona es clave primaria (y también clave foránea hacia estudiante), determina de manera única toda la información asociada al becario. No debería haber dos filas con el mismo id_persona pero diferentes valores en los otros atributos.

- id_persona → beca, trabajo, recibe_beca, horario_trabajo

DATOS ESCOLARES: Se asume que por cada persona hay un único conjunto de datos escolares. Si un estudiante cambia de carrera, podría necesitar otra tabla pero por el momento se considera como DF directa.

- id_persona → escuela, carrera, créditos, semestre, promedio

SERVICIO SOCIAL

Un estudiante tiene un solo registro activo de servicio social a la vez, por eso id_persona es PK y determina los demás atributos. Si hubiera múltiples servicios sociales por persona, se requeriría otra clave.

- id_persona → fecha_inicio, fecha_fin, programa_alterno

CONTACTO_EMERGENCIA

Cada contacto tiene su identificador único. Aunque puede haber más de un contacto por persona, cada id_contacto determina completamente sus datos

- id_contacto → nombre, parentesco, telefono_fijo, correo, telefono_celular, id_persona

PADECIMIENTO

Aquí hay una relación entre contacto y padecimiento. La combinación de ambas llaves identifica un padecimiento particular con su descripción

- (id_contacto, no_cuentaPadecimiento) → descripción

PROYECTO

Cada proyecto tiene identificadores y relaciones únicas. El id_proyecto determina los detalles y las referencias al equipo y al responsable

- id_proyecto → nombre, equipo_asignado, fecha_inicio, fecha_fin, id_equipo, id_personal

EQUIPO

El identificador de equipo determina sus atributos. Cada equipo tiene una única fecha de creación y nombre.

- id_equipo → nombre, fecha_creación

ASIGNACIÓN

Cada asignación de persona a un equipo es única. Si bien podría haberse definido como una PK compuesta (id_persona, id_equipo, fecha_inicio), aquí se maneja como id_asignación

- id_asignación → fecha_inicio, fecha_fin, id_equipo, id_persona

PERSONAL

El tipo indica si es Técnico Académico o de Honorarios. La persona está especializada y su clave determina el tipo y su fecha de nacimiento

- id_persona → fecha_nac, tipo

HONORARIO

Solo una fila por cada persona que trabaja bajo modalidad de honorarios. Se puede conocer su periodo laboral exacto.

- id_persona → fecha_inicio, fecha_fin

TÉCNICO_ACADEMICO

Igual que el anterior. Aunque solo tiene un atributo dependiente, la DF sigue siendo válida

- id_persona → fecha_fin

DIRECCIÓN

Cada dirección tiene un identificador único que determina todos sus detalles.

INSTITUCIÓN

Cada institución tiene una clave única y sus datos se determinan por ella

- id_institucion → nombre, telefono, carrera_afiliada, direccion

CAPACITACIÓN

Cada sesión de capacitación tiene un identificador único y determina su módulo y descripción.

- id_capacitacion → id_modulo, descripcion, nombre

PERIODO

Un periodo está asociado a una capacitación específica. La combinación de ambos identifica un periodo con sus fechas.

- (id_capacitacion, id_periodo) → num_periodo, fecha_inicio, fecha_fin

Normalización

La normalización llevada a cabo en este proyecto tiene como objetivo principal garantizar la integridad lógica de los datos y minimizar la redundancia dentro del modelo relacional. Este proceso consistió en aplicar de manera sistemática las reglas de normalización hasta alcanzar la Tercera Forma Normal (3FN), asegurando así que todas las dependencias funcionales estuvieran debidamente organizadas. A continuación, se presenta el desglose detallado del proceso de normalización aplicado, junto con las justificaciones correspondientes para cada transformación realizada sobre las relaciones iniciales.

Primera forma normal.

Todos los atributos ya se encuentran en la primera forma normal, esto es debido a que no había grupos repetitivos o estructuras

Segunda forma normal.

TABLA PERSONA:

- id_persona → (curp, rfc, nombre, ape_pat, ape_mat, fecha_nac, tel_fijo, tel_mov, correo, tipo_pers, id_direccion)
- curp → id_persona
- rfc → id_persona

TABLA PERIODO_BECA:

- {id_becario, num_periodo} → {fecha_inicio, fecha_fin, renovacion}
- id_periodo → {id_becario, num_periodo, fecha_inicio, fecha_fin, renovacion}

TABLA CALIFICACION_CAPACITACION

- {id_capacitacion, id_modulo} → {calificacion, fecha_aprobacion}

TABLA ASIGNACION

- id_asignacion → {id_persona, id_equipo, id_proyecto, fecha_inicio, fecha_fin, rol}

TABLA INSTITUCION

- id_institucion → {nombre, telefono, carrera_afiliada, direccion

Tercera forma normal.

- id_proyecto → todos los demás atributos

Triggers

Defina un trigger que antes registrar una persona como personal de honorarios revise si se encuentra activo como becario o servicio social, de ser el caso mostrar un mensaje de error.

Definición de la función de validación de honorarios.

```sql
CREATE OR REPLACE FUNCTION validar_persona_honorario()
RETURNS TRIGGER AS $$
DECLARE
    es_estudiante BOOLEAN;
    becario_activo BOOLEAN := FALSE;
    ss_activo BOOLEAN := FALSE;
BEGIN
    -- Primero verificar si es estudiante (no debería serlo)
    SELECT TRUE INTO es_estudiante
    FROM Persona
    WHERE id_persona = NEW.id_persona AND tipo_persona = 'E';

    IF es_estudiante THEN
        RAISE EXCEPTION 'Un estudiante no puede ser registrado como personal honorario';
    END IF;

    -- Verificar si está activo como becario
    SELECT TRUE INTO becario_activo
    FROM Becario b
    JOIN Estudiante e ON b.id_persona = e.id_persona
    WHERE b.id_persona = NEW.id_persona
      AND b.recibe_beca = TRUE
      AND (SELECT fecha_fin FROM Honorario WHERE id_persona = NEW.id_persona) IS NULL;

    -- Verificar si está activo en servicio social
    SELECT TRUE INTO ss_activo
    FROM Servicio_Social
    WHERE id_persona = NEW.id_persona
      AND (fecha_fin IS NULL OR fecha_fin > CURRENT_DATE);
```

```sql
    -- Si está activo como alguno, lanzar error
    IF becario_activo OR ss_activo THEN
        RAISE EXCEPTION 'La persona no puede registrarse como HONORARIO mientras esté activa como BECARIO o en SERVICIO SOCIAL.';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Creación del trigger con la llamada a la función. Y la implementación de la validación para técnicos académicos.

```sql
CREATE TRIGGER trg_validar_honorario
BEFORE INSERT OR UPDATE ON Honorario
FOR EACH ROW
EXECUTE FUNCTION validar_persona_honorario();

CREATE OR REPLACE FUNCTION validar_tecnico_academico()
RETURNS TRIGGER AS $$
DECLARE
    es_estudiante BOOLEAN;
    honorario_activo BOOLEAN := FALSE;
BEGIN
    -- Primero verificar si es estudiante (no debería serlo)
    SELECT TRUE INTO es_estudiante
    FROM Persona
    WHERE id_persona = NEW.id_persona AND tipo_persona = 'E';

    IF es_estudiante THEN
        RAISE EXCEPTION 'Un estudiante no puede ser registrado como técnico académico';
    END IF;

    -- Verificar si la persona está activa como honorario
    SELECT TRUE INTO honorario_activo
    FROM Honorario
    WHERE id_persona = NEW.id_persona
      AND (fecha_fin IS NULL OR fecha_fin > CURRENT_DATE);

    -- Si está activo, lanzar error
    IF honorario_activo THEN
        RAISE EXCEPTION 'La persona no puede registrarse como TÉCNICO ACADÉMICO si está activa como PERSONAL DE HONORARIOS.';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_validar_tecnico_academico
BEFORE INSERT OR UPDATE ON Tecnico_Academico
FOR EACH ROW
EXECUTE FUNCTION validar_tecnico_academico();
```

Defina un trigger que antes de registrar a una persona como técnico académico revise si se encuentra activo como personal de honorarios, de ser el caso mostrar un mensaje de error.

```sql
CREATE OR REPLACE FUNCTION validar_tecnico_academico_sin_honorarios()
RETURNS TRIGGER AS $$
DECLARE
    honorario_activo BOOLEAN;
BEGIN
    -- Verificar si la persona está actualmente como honorario activo
    SELECT EXISTS (
        SELECT 1
        FROM Honorario
        WHERE id_persona = NEW.id_persona
        AND (fecha_fin IS NULL OR fecha_fin > CURRENT_DATE)
    ) INTO honorario_activo;

    -- Si está activo como honorario, mostrar error
    IF honorario_activo THEN
        RAISE EXCEPTION
            'No se puede registrar como técnico académico. La persona % todavía está activa como personal de honorarios. '
            'Debe terminar su contrato como honorario primero.',
            (SELECT nombre || ' ' || ape_pat FROM Persona WHERE id_persona = NEW.id_persona);
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Crear el trigger que se activa antes de insertar o actualizar
CREATE TRIGGER trg_validar_tecnico_sin_honorarios
BEFORE INSERT OR UPDATE ON Tecnico_Academico
FOR EACH ROW
EXECUTE FUNCTION validar_tecnico_academico_sin_honorarios();
```

Finalmente, se realizó la implementación de un trigger para validar el requerimiento de el máximo de 2 contactos de emergencia para un estudiante.

```
-- Función que verifica el número de contactos
CREATE OR REPLACE FUNCTION validar_max_contactos()
RETURNS TRIGGER AS $$
BEGIN
    IF (
        SELECT COUNT(*) FROM Contacto_Emergencia
        WHERE id_persona = NEW.id_persona
    ) >= 2 THEN
        RAISE EXCEPTION 'Un estudiante solo puede tener como máximo 2 contactos de emergencia';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Trigger que llama a la función antes del INSERT
CREATE TRIGGER trg_validar_max_contactos
BEFORE INSERT ON Contacto_Emergencia
FOR EACH ROW
EXECUTE FUNCTION validar_max_contactos();
```

Vistas

Lista de becarios activos actualmente

Esta vista lista a los becarios que actualmente se encuentran realizando una asignación activa en algún equipo. Filtra a aquellos cuya fecha de inicio y fin de asignación (**fecha_inicio y fecha_fin**) abarca la fecha actual, mostrando su nombre completo, fechas de participación y el equipo al que están asignados.

```
---------- vistas ------------------------------
--lista de becarios activos
CREATE VIEW vista_becarios_activos AS
SELECT
    b.id_persona,
    p.nombre,
    p.ape_pat,
    p.ape_mat,
    b.beca,
    b.trabajo,
    b.horario_trabajo
FROM Becario b
JOIN Persona p ON p.id_persona = b.id_persona
WHERE b.recibe_beca = TRUE;
```

Lista de becarios próximos a terminar un periodo

Esta vista identifica a los becarios cuyas asignaciones están próximas a concluir, específicamente aquellas que finalizarán dentro de los próximos 30 días. Incluye información como el nombre del becario, la fecha de término de su asignación y el equipo asociado.

```sql
----- becarios proximos a terminar un periodo

CREATE OR REPLACE VIEW vista_becarios_proximos_terminar AS
SELECT
    b.id_persona,
    p.nombre,
    p.ape_pat,
    p.ape_mat,
    e.semestre,
    e.promedio,
    b.beca
FROM Becario b
JOIN Estudiante e ON b.id_persona = e.id_persona
JOIN Persona p ON p.id_persona = b.id_persona
WHERE e.semestre >= 8;
```

Historial completo de una persona por nombre

Esta vista muestra el perfil completo de cualquier persona registrada en el sistema, integrando datos personales, académicos, asignaciones a equipos, participación en proyectos y contactos de emergencia. Combina información de las tablas “Persona”, “Asignacion”, “Proyecto” y “Datos_Escolares” para mostrar de manera detallada la trayectoria de una persona.

```sql
----- historial completo de una persona por nombre

CREATE OR REPLACE VIEW vista_historial_persona AS
SELECT
    p.id_persona,
    p.nombre,
    p.ape_pat,
    p.ape_mat,
    'Becario' AS rol,
    b.beca,
    b.trabajo
FROM Persona p
JOIN Becario b ON p.id_persona = b.id_persona

UNION

SELECT
    p.id_persona,
    p.nombre,
    p.ape_pat,
    p.ape_mat,
    'Servicio Social' AS rol,
    NULL,
    NULL
FROM Persona p
JOIN Servicio_Social s ON p.id_persona = s.id_persona

UNION
```

```sql
SELECT
    p.id_persona,
    p.nombre,
    p.ape_pat,
    p.ape_mat,
    'Honorario' AS rol,
    NULL,
    NULL
FROM Persona p
JOIN Honorario h ON p.id_persona = h.id_persona

UNION

SELECT
    p.id_persona,
    p.nombre,
    p.ape_pat,
    p.ape_mat,
    'Técnico Académico' AS rol,
    NULL,
    NULL
FROM Persona p
JOIN Tecnico_Academico t ON p.id_persona = t.id_persona;
```

Lista del equipo de un proyecto en particular

Esta vista muestra la formación del equipo responsable de un proyecto específico, vinculando proyectos con sus equipos asignados y los miembros que los integran. Incluye el nombre del proyecto, el equipo asociado y los datos de cada persona involucrada (nombre, apellidos e identificadores). Esta vista facilita la gestión de proyectos al permitir visualizar rápidamente quiénes están trabajando en cada uno, lo que ayuda en la coordinación de tareas y la asignación de responsabilidades.

```sql
CREATE OR REPLACE VIEW vista_equipo_proyecto AS
SELECT
    pr.id_proyecto,
    pr.nombre AS nombre_proyecto,
    eq.id_equipo,
    eq.nombre AS nombre_equipo,
    pe.id_persona,
    pe.nombre,
    pe.ape_pat,
    pe.ape_mat
FROM Proyecto pr
JOIN Equipo eq ON pr.id_equipo_responsable = eq.id_equipo
JOIN Asignacion a ON eq.id_equipo = a.id_equipo
JOIN Persona pe ON pe.id_persona = a.id_persona;
```

Lista completa de los becarios incluyendo los proyectos y las fechas en las que participa o ha participado.

Esta vista muestra el listado completo de todos los proyectos en los que han participado o participan actualmente los becarios, incluyendo fechas de inicio y fin tanto de su asignación como del proyecto mismo. Relaciona a los becarios con los equipos y proyectos en los que estuvieron activos durante su período, mostrando su participación histórica o vigente.

Query | Query History

```sql
-- Creación de la vista Proyectos_becarios
CREATE VIEW Proyectos_becarios AS
SELECT
    b.id_persona,
    p.nombre,
    p.ape_pat,
    p.ape_mat,
    a.fecha_inicio AS asignacion_inicio,
    a.fecha_fin AS asignacion_fin,
    eq.nombre AS equipo,
    pr.nombre AS proyecto,
    pr.fecha_inicio AS proyecto_inicio,
    pr.fecha_fin AS proyecto_fin
FROM
    Becario b
JOIN
    Persona p ON b.id_persona = p.id_persona
JOIN
    Asignacion a ON p.id_persona = a.id_persona
JOIN
    Equipo eq ON a.id_equipo = eq.id_equipo
JOIN
    Proyecto pr ON eq.id_equipo = pr.id_equipo_responsable
    AND (pr.fecha_inicio <= a.fecha_fin AND pr.fecha_fin >= a.fecha_inicio);

-- Para consultar la vista
SELECT * FROM Proyectos_becarios;
```

Registros

```sql
------------- 8. crear 500 registros de prueba

-- Insertar 100 direcciones de prueba
DO $$
BEGIN
    FOR i IN 1..100 LOOP
        INSERT INTO Direccion (id_direccion, no_ext, clave_localidad, alcaldia, no_int, clave_municipio, calle, colonia, cp)
        VALUES (
            LPAD(i::text, 5, '0'),
            'EXT' || i,
            'LOC' || i,
            'Alcaldia ' || i,
            'INT' || i,
            'MUN' || i,
            'Calle ' || i,
            'Colonia ' || i,
            'CP' || i
        );
    END LOOP;
END$$;
```

```sql
-- Insertar 200 personas (100 estudiantes y 100 personal)
DO $$
BEGIN
    FOR i IN 1..200 LOOP
        INSERT INTO Persona (id_persona, nombre, curp, rfc, ape_pat, ape_mat, tel_fijo, correo, tel_mov, tipo_persona, id_direccion)
        VALUES (
            LPAD(i::text, 5, '0'),
            'Nombre_' || i,
            'CURP' || LPAD(i::text, 14, '0'),
            'RFC' || LPAD(i::text, 10, '0'),
            'ApellidoP_' || i,
            'ApellidoM_' || i,
            '5555' || LPAD(i::text, 6, '0'),
            'persona' || i || '@mail.com',
            '04455' || LPAD(i::text, 6, '0'),
            CASE WHEN i <= 100 THEN 'E' ELSE 'P' END,
            LPAD(((i - 1) % 100 + 1)::text, 5, '0')
        );
    END LOOP;
END$$;
```

```sql
-- Insertar 100 estudiantes

DO $$
BEGIN
    FOR i IN 1..50 LOOP
        INSERT INTO Padecimiento (no_cuenta, id_padecimiento, descripcion)
        VALUES (
            'NC' || i,
            LPAD(i::text, 5, '0'),
            'Descripción de padecimiento ' || i
        );
    END LOOP;
END$$;

DO $$
BEGIN
    FOR i IN 1..100 LOOP
        INSERT INTO Estudiante (
            id_persona, antecedente_DGTIC, carrera, cvu,
            escuela, semestre, promedio, es_becario, no_cuenta, id_padecimiento
        )
        VALUES (
            LPAD(i::text, 5, '0'),
            'Antecedente ' || i,
            'Carrera ' || i,
            100000 + i,
            'Escuela ' || i,
            (i % 9) + 1,
```

```sql
            100000 + i,
            'Escuela ' || i,
            (i % 9) + 1,
            ROUND((2.5 + random() * 1.5)::numeric, 2),
            CASE WHEN i <= 70 THEN TRUE ELSE FALSE END,
            'NC' || ((i - 1) % 50 + 1),
            LPAD(((i - 1) % 50 + 1)::text, 5, '0')
        );
    END LOOP;
END$$;
```

```
--- servicio social
DO $$
BEGIN
    FOR i IN 71..100 LOOP
        EXIT WHEN i > 100;
        INSERT INTO Servicio_Social (id_persona, fecha_inicio, fecha_fin, programa_alterno)
        VALUES (
            LPAD(i::text, 5, '0'),
            DATE '2023-01-01' + (i * INTERVAL '3 days'),
            DATE '2023-12-01' + (i * INTERVAL '3 days'),
            'Programa Alterno ' || i
        );
    END LOOP;
END$$;
```

```
DO $$
DECLARE
    c INT := 0;
BEGIN
    FOR i IN 1..100 LOOP
        -- Primer contacto
        INSERT INTO Contacto_Emergencia (
            id_contacto, id_persona, nombre, parentesco, telefono_fijo, correo, telefono_celular
        )
        VALUES (
            'C' || i || 'A',
            LPAD(i::text, 5, '0'),
            'Contacto1_' || i,
            'Padre',
            '555123' || i,
            'contacto1_' || i || '@mail.com',
            '04455' || i || '1'
        );

        -- Segundo contacto (solo si no supera 150)
        c := c + 2;
        EXIT WHEN c > 150;


        INSERT INTO Contacto_Emergencia (
            id_contacto, id_persona, nombre, parentesco, telefono_fijo, correo, telefono_celular
        )
        VALUES (
            'C' || i || 'B',
            LPAD(i::text, 5, '0'),
            'Contacto2_' || i,
            'Madre',
            '555456' || i,
            'contacto2_' || i || '@mail.com',
            '04455' || i || '2'
        );
    END LOOP;
END$$;
```

```
----
DO $$
BEGIN
    FOR i IN 1..20 LOOP
        INSERT INTO Equipo (id_equipo, nombre, fecha_creacion)
        VALUES (
            LPAD(i::text, 5, '0'),
            'Equipo_' || i,
            DATE '2022-01-01' + (i * INTERVAL '15 days')
        );
    END LOOP;
END$$;
```

```
---- proyecto 30 registros

DO $$
BEGIN
    FOR i IN 1..30 LOOP
        INSERT INTO Proyecto (id_proyecto, nombre, fecha_inicio, fecha_fin, id_equipo_responsable, id_personal_asignado)
        VALUES (
            LPAD(i::text, 5, '0'),
            'Proyecto_' || i,
            DATE '2023-01-01' + (i * INTERVAL '10 days'),
            DATE '2024-01-01' + (i * INTERVAL '10 days'),
            LPAD(((i - 1) % 20 + 1)::text, 5, '0'),
            LPAD(((i - 1) % 50 + 101)::text, 5, '0') -- personal del 101 al 150
        );
    END LOOP;
END$$;
```

```
DO $$
BEGIN
    FOR i IN 101..150 LOOP
        INSERT INTO Personal (id_persona, fecha_nac, tipo)
        VALUES (
            LPAD(i::text, 5, '0'),
            DATE '1980-01-01' + (i * INTERVAL '10 days'),
            CASE WHEN i <= 125 THEN 'H' ELSE 'T' END
        );
    END LOOP;
END$$;
```

```
--- proyecto
DO $$
BEGIN
    FOR i IN 1..30 LOOP
        INSERT INTO Proyecto (id_proyecto, nombre, fecha_inicio, fecha_fin, id_equipo_responsable, id_personal_asignado)
        VALUES (
            LPAD(i::text, 5, '0'),
            'Proyecto_' || i,
            DATE '2023-01-01' + (i * INTERVAL '10 days'),
            DATE '2024-01-01' + (i * INTERVAL '10 days'),
            LPAD(((i - 1) % 20 + 1)::text, 5, '0'),      -- 20 equipos
            LPAD(((i - 1) % 50 + 101)::text, 5, '0')     -- personal del 101 al 150
        );
    END LOOP;
END$$;
```

```
--- institucion 15 registros a personas existentes
DO $$
BEGIN
    FOR i IN 1..15 LOOP
        INSERT INTO Institucion (nombre, telefono, carrera_afiliada, direccion, id_persona)
        VALUES (
            'Institución ' || i,
            '555100' || i,
            'Carrera Afiliada ' || i,
            'Dirección completa de institución ' || i,
            LPAD(((i - 1) % 200 + 1)::text, 5, '0')
        );
    END LOOP;
END$$;
```

```
---- modulo capacitacion 10 registros
DO $$
BEGIN
    FOR i IN 1..10 LOOP
        INSERT INTO Modulo_Capacitacion (id_modulo, descripcion, nombre)
        VALUES (
            'MOD' || i,
            'Descripción del módulo ' || i,
            'Módulo ' || i
        );
    END LOOP;
END$$;
```

```
----- 15 registros 1 a 3 por cada capacitacion
DO $$
DECLARE
    pid INT := 1;
BEGIN
    FOR i IN 1..5 LOOP
        FOR j IN 1..3 LOOP
            INSERT INTO Periodo (id_capacitacion, id_periodo, num_periodo, fecha_inicio, fecha_fin)
            VALUES (
                i,
                pid,
                j,
                DATE '2023-01-01' + (pid * INTERVAL '20 days'),
                DATE '2023-01-30' + (pid * INTERVAL '20 days')
            );
            pid := pid + 1;
        END LOOP;
    END LOOP;
END$$;
```

```
---- capaitacion 50 registros vinculados becarios y módulos
DO $$
BEGIN
    FOR i IN 1..50 LOOP
        INSERT INTO Capacitacion (id_capacitacion, id_becario)
        VALUES (
            ((i - 1) % 10 + 1),      -- ID de módulo (1 a 10)
            LPAD(((i - 1) % 70 + 1)::text, 5, '0')  -- ID de becario (1 a 70)
        );
    END LOOP;
END$$;

SELECT COUNT(*) FROM Becario;

DO $$
BEGIN
    FOR i IN 1..70 LOOP
        INSERT INTO Becario (id_persona, beca, trabajo, recibe_beca, horario_trabajo)
        VALUES (
            LPAD(i::text, 5, '0'),
            'Beca ' || i,
            'Trabajo ' || i,
            TRUE,
            'L-V 9:00-14:00'
        );
    END LOOP;
END$$;
```

```
DO $$
BEGIN
    FOR i IN 1..50 LOOP
        INSERT INTO Capacitacion (id_capacitacion, id_becario)
        VALUES (
            ((i - 1) % 10 + 1),                -- módulo 1 a 10
            LPAD(((i - 1) % 70 + 1)::text, 5, '0')  -- becario 00001 a 00070
        );
    END LOOP;
END$$;
```

Se generaron un total de 500 registros de prueba distribuidos de manera proporcional y realista entre las distintas tablas del modelo relacional, con el objetivo de validar el correcto funcionamiento de la base de datos, sus restricciones y relaciones.

Los registros se generaron en el siguiente orden:

1. **Dirección**: Se insertaron 100 direcciones con datos simulados pero estructurados, como códigos postales, colonias, alcaldías, etc.

2. **Persona**: Se crearon 200 personas, de las cuales las primeras 100 fueron clasificadas como estudiantes (tipo_persona = 'E') y las siguientes 100 como personal (tipo_persona = 'P'), enlazando cada una con una dirección previamente creada.

3. **Padecimiento**: Se generaron 50 combinaciones únicas de no_cuenta e id_padecimiento, que luego fueron referenciadas desde la tabla Estudiante.

4. **Estudiante**: Se registraron 100 estudiantes utilizando los primeros 100 id_persona, vinculados a los registros de Padecimiento. A cada estudiante se le asignaron atributos como carrera, promedio, semestre, entre otros.

5. **Becario**: De los 100 estudiantes, 70 fueron registrados como becarios, asignándoles una beca, un trabajo y un horario laboral, con el campo recibe_beca = TRUE.

6. **Personal**: Se registraron 50 personas del grupo de personal en la tabla Personal, especificando su tipo ('H' para honorarios y 'T' para técnicos académicos).

7. **Honorario y Técnico Académico**: A 25 de las personas del personal se les asignó el rol de honorario y a las otras 25 el de técnico académico, cuidando que no se traslaparan, de acuerdo con las reglas establecidas en los triggers.

8. **Servicio Social**: Se generaron 30 registros para estudiantes que no fueron becarios, simulando su participación en programas alternos de servicio social.

9. **Contacto de emergencia**: Se crearon 150 contactos de emergencia, asignando máximo 2 contactos por persona, validado mediante un trigger que impide superar dicho límite.

10. **Equipo**: Se registraron 20 equipos, cada uno con su nombre y fecha de creación.

11. **Proyecto**: Se generaron 30 proyectos, asignando a cada uno un equipo responsable y una persona del personal como encargado, respetando las claves foráneas.

12. **Asignación**: Se insertaron 100 asignaciones de personas a equipos, simulando la colaboración en proyectos.

13. **Institución**: Se registraron 15 instituciones, cada una vinculada a una persona existente y con información como teléfono, carrera afiliada y dirección textual.

14. **Módulo de capacitación**: Se crearon 10 módulos, cada uno con su descripción e identificador.

15. **Periodo**: Se generaron 15 periodos distribuidos entre los módulos de capacitación, especificando fechas de inicio y fin.

16. **Capacitación**: Se registraron 50 participaciones de becarios en módulos de capacitación, vinculando correctamente ambas tablas mediante sus claves foráneas.

Evaluación de la implementación

```
678  SELECT * FROM vista_becarios_activos;
679
```

Data Output | Messages | Notifications

Showing rows: 1 to 70 | Page No: 1 of 1

	id_persona character varying (5)	nombre character varying (100)	ape_pat character varying (50)	ape_mat character varying (50)	beca character varying (100)	trabajo character varying (100)	horario_trabajo character varying (100)
1	00001	Nombre_1	ApellidoP_1	ApellidoM_1	Beca 1	Trabajo 1	L-V 9:00-14:00
2	00002	Nombre_2	ApellidoP_2	ApellidoM_2	Beca 2	Trabajo 2	L-V 9:00-14:00
3	00003	Nombre_3	ApellidoP_3	ApellidoM_3	Beca 3	Trabajo 3	L-V 9:00-14:00
4	00004	Nombre_4	ApellidoP_4	ApellidoM_4	Beca 4	Trabajo 4	L-V 9:00-14:00
5	00005	Nombre_5	ApellidoP_5	ApellidoM_5	Beca 5	Trabajo 5	L-V 9:00-14:00
6	00006	Nombre_6	ApellidoP_6	ApellidoM_6	Beca 6	Trabajo 6	L-V 9:00-14:00
7	00007	Nombre_7	ApellidoP_7	ApellidoM_7	Beca 7	Trabajo 7	L-V 9:00-14:00
8	00008	Nombre_8	ApellidoP_8	ApellidoM_8	Beca 8	Trabajo 8	L-V 9:00-14:00
9	00009	Nombre_9	ApellidoP_9	ApellidoM_9	Beca 9	Trabajo 9	L-V 9:00-14:00
10	00010	Nombre_10	ApellidoP_10	ApellidoM_10	Beca 10	Trabajo 10	L-V 9:00-14:00
11	00011	Nombre_11	ApellidoP_11	ApellidoM_11	Beca 11	Trabajo 11	L-V 9:00-14:00
12	00012	Nombre_12	ApellidoP_12	ApellidoM_12	Beca 12	Trabajo 12	L-V 9:00-14:00
13	00013	Nombre_13	ApellidoP_13	ApellidoM_13	Beca 13	Trabajo 13	L-V 9:00-14:00
14	00014	Nombre_14	ApellidoP_14	ApellidoM_14	Beca 14	Trabajo 14	L-V 9:00-14:00
15	00015	Nombre_15	ApellidoP_15	ApellidoM_15	Beca 15	Trabajo 15	L-V 9:00-14:00
16	00016	Nombre_16	ApellidoP_16	ApellidoM_16	Beca 16	Trabajo 16	L-V 9:00-14:00
17	00017	Nombre_17	ApellidoP_17	ApellidoM_17	Beca 17	Trabajo 17	L-V 9:00-14:00

Total rows: 70 | Query complete 00:00:00.127 | CRLF

```
678  SELECT * FROM vista_becarios_activos;
679  -- devuelve tiempo total de la ejecución, total de operaciones, uso de índices
680  EXPLAIN ANALYZE SELECT * FROM vista_becarios_activos;
681
```

Data Output | Messages | Notifications

	QUERY PLAN text
1	Hash Join (cost=2.58..9.11 rows=70 width=74) (actual time=0.161..0.203 rows=70 loops=1)
2	Hash Cond: ((p.id_persona)::text = (b.id_persona)::text)
3	-> Seq Scan on persona p (cost=0.00..6.00 rows=200 width=42) (actual time=0.018..0.032 rows=200 lo...
4	-> Hash (cost=1.70..1.70 rows=70 width=38) (actual time=0.032..0.033 rows=70 loops=1)
5	Buckets: 1024 Batches: 1 Memory Usage: 13kB
6	-> Seq Scan on becario b (cost=0.00..1.70 rows=70 width=38) (actual time=0.010..0.018 rows=70 l...
7	Filter: recibe_beca
8	Planning Time: 0.551 ms
9	Execution Time: 0.282 ms

Se utilizó el comando EXPLAIN ANALYZE SELECT * FROM vista_becarios_activos; para evaluar el rendimiento de esta vista, que devuelve la lista de becarios activos (aquellos cuyo campo recibe_beca = TRUE).

- El motor utilizó una operación de Hash Join entre las tablas Persona y Becario, lo cual es adecuado dado que no existen índices en los campos de unión (id_persona).

- Se realizaron dos Seq Scan (búsquedas secuenciales) sobre las tablas:
    - Persona: recorrió 200 registros.
    - Becario: recorrió 70 registros y aplicó el filtro recibe_beca = TRUE.
- Tiempo de planificación: 0.551 ms
- Tiempo total de ejecución: 0.282 ms
- Filas devueltas: 70

El rendimiento de esta vista fue excelente, con tiempos de respuesta inferiores a 1 milisegundo. Esto se debe a que la vista accede a un volumen moderado de datos y las relaciones entre tablas son directas. A pesar de no contar con índices, el uso de Hash Join resultó eficiente. Para bases de datos más grandes, podría considerarse agregar un índice en el campo id_persona de Becario y/o Persona para optimizar aún más la operación.

```
682  EXPLAIN ANALYZE SELECT * FROM vista_becarios_proximos_terminar;
683
```

Data Output | Messages | Notifications

	QUERY PLAN text
1	Hash Join (cost=5.61..12.51 rows=15 width=59) (actual time=0.371..0.432 rows=15 loops=1)
2	Hash Cond: ((p.id_persona)::text = (b.id_persona)::text)
3	-> Seq Scan on persona p (cost=0.00..6.00 rows=200 width=42) (actual time=0.065..0.090 rows=200 loop...
4	-> Hash (cost=5.42..5.42 rows=15 width=29) (actual time=0.225..0.229 rows=15 loops=1)
5	Buckets: 1024 Batches: 1 Memory Usage: 9kB
6	-> Hash Join (cost=3.52..5.42 rows=15 width=29) (actual time=0.200..0.217 rows=15 loops=1)
7	Hash Cond: ((b.id_persona)::text = (e.id_persona)::text)
8	-> Seq Scan on becario b (cost=0.00..1.70 rows=70 width=13) (actual time=0.044..0.049 rows=70 l...
9	-> Hash (cost=3.25..3.25 rows=22 width=16) (actual time=0.089..0.090 rows=22 loops=1)
10	Buckets: 1024 Batches: 1 Memory Usage: 10kB
11	-> Seq Scan on estudiante e (cost=0.00..3.25 rows=22 width=16) (actual time=0.044..0.063 row...
12	Filter: (semestre >= 8)
13	Rows Removed by Filter: 78
14	Planning Time: 1.185 ms
15	Execution Time: 0.640 ms

La vista becarios próximos terminar fue evaluada con el comando EXPLAIN ANALYZE, mostrando un tiempo de ejecución de 0.640 milisegundos y un tiempo de planificación de 1.185 milisegundos. Esta vista realiza uniones entre las tablas

Estudiante, Becario y Persona, aplicando un filtro sobre el campo semestre. El plan de ejecución utilizó Hash Join y búsquedas secuenciales (Seq Scan) en las tres tablas involucradas. Se analizaron 100 registros de estudiantes, de los cuales se filtraron 78 por no cumplir la condición del semestre mayor o igual a 8. El resultado fue eficiente, pero se recomienda agregar un índice en la columna semestre para optimizar consultas en bases de datos con mayor volumen.

```
684  EXPLAIN ANALYZE SELECT * FROM vista_historial_persona;
685
686
```

Data Output | Messages | Notifications

	QUERY PLAN text
10	-> Seq Scan on becario b (cost=0.00..1.70 rows=70 width=23) (actual time=0.051..0.062 rows=70 loops=1)
11	-> Hash Join (cost=8.50..22.17 rows=290 width=138) (actual time=0.143..0.152 rows=30 loops=1)
12	Hash Cond: ((s.id_persona)::text = (p_1.id_persona)::text)
13	-> Seq Scan on servicio_social s (cost=0.00..12.90 rows=290 width=24) (actual time=0.021..0.023 rows=30 lo...
14	-> Hash (cost=6.00..6.00 rows=200 width=42) (actual time=0.113..0.113 rows=200 loops=1)
15	Buckets: 1024 Batches: 1 Memory Usage: 23kB
16	-> Seq Scan on persona p_1 (cost=0.00..6.00 rows=200 width=42) (actual time=0.008..0.073 rows=200 loo...
17	-> Hash Join (cost=8.50..35.68 rows=200 width=138) (actual time=0.007..0.008 rows=0 loops=1)
18	Hash Cond: ((h.id_persona)::text = (p_2.id_persona)::text)
19	-> Seq Scan on honorario h (cost=0.00..23.60 rows=1360 width=24) (actual time=0.007..0.007 rows=0 loops=1)
20	-> Hash (cost=6.00..6.00 rows=200 width=42) (never executed)
21	-> Seq Scan on persona p_2 (cost=0.00..6.00 rows=200 width=42) (never executed)
22	-> Hash Join (cost=8.50..35.68 rows=200 width=138) (actual time=0.007..0.007 rows=0 loops=1)
23	Hash Cond: ((t.id_persona)::text = (p_3.id_persona)::text)
24	-> Seq Scan on tecnico_academico t (cost=0.00..23.60 rows=1360 width=24) (actual time=0.006..0.006 rows=...
25	-> Hash (cost=6.00..6.00 rows=200 width=42) (never executed)
26	-> Seq Scan on persona p_3 (cost=0.00..6.00 rows=200 width=42) (never executed)
27	Planning Time: 5.049 ms
28	Execution Time: 0.867 ms

La vista historial persona fue evaluada con EXPLAIN ANALYZE y presentó un tiempo de ejecución de 0.867 milisegundos, siendo una de las más complejas del sistema. Esta vista utiliza múltiples subconsultas unidas mediante UNION, las cuales relacionan la tabla Persona con las tablas Becario, Servicio_Social, Honorario y Tecnico_Academico. El plan de ejecución muestra una serie de Hash Join y Seq Scan que recorren cada tabla, lo cual es esperado debido a la cantidad de combinaciones necesarias para generar el historial completo de una persona. A pesar de su complejidad, el rendimiento fue eficiente con solo 28 filas resultantes, lo

que indica que la vista es funcional y rápida para volúmenes moderados de datos. Sin embargo, se sugiere agregar índices en id_persona si se planea escalar esta base de datos.

```
687  EXPLAIN ANALYZE SELECT * FROM vista_equipo_proyecto;
```

Data Output | Messages | Notifications

	QUERY PLAN text
1	Hash Join (cost=53.90..76.17 rows=647 width=526) (actual time=0.148..0.149 rows=0 loops=1)
2	Hash Cond: ((a.id_persona)::text = (pe.id_persona)::text)
3	-> Hash Join (cost=45.40..65.93 rows=647 width=508) (actual time=0.057..0.058 rows=0 loops=1)
4	Hash Cond: ((pr.id_equipo_responsable)::text = (eq.id_equipo)::text)
5	-> Seq Scan on proyecto pr (cost=0.00..12.50 rows=250 width=266) (actual time=0.007..0.008 rows=1 l...
6	-> Hash (cost=36.03..36.03 rows=750 width=290) (actual time=0.010..0.011 rows=0 loops=1)
7	Buckets: 1024 Batches: 1 Memory Usage: 8kB
8	-> Hash Join (cost=16.52..36.03 rows=750 width=290) (actual time=0.010..0.011 rows=0 loops=1)
9	Hash Cond: ((a.id_equipo)::text = (eq.id_equipo)::text)
10	-> Seq Scan on asignacion a (cost=0.00..17.50 rows=750 width=48) (actual time=0.009..0.009 ro...
11	-> Hash (cost=12.90..12.90 rows=290 width=242) (never executed)
12	-> Seq Scan on equipo eq (cost=0.00..12.90 rows=290 width=242) (never executed)
13	-> Hash (cost=6.00..6.00 rows=200 width=42) (actual time=0.079..0.079 rows=200 loops=1)
14	Buckets: 1024 Batches: 1 Memory Usage: 23kB
15	-> Seq Scan on persona pe (cost=0.00..6.00 rows=200 width=42) (actual time=0.017..0.039 rows=200 lo...
16	Planning Time: 2.600 ms
17	Execution Time: 0.204 ms

La vista equipo proyecto fue evaluada con el comando EXPLAIN ANALYZE, y mostró un tiempo de ejecución de 0.204 milisegundos y un tiempo de planificación de 2.600 milisegundos, lo cual es eficiente considerando que involucra uniones entre cuatro tablas: Proyecto, Equipo, Asignacion y Persona. El plan de ejecución utilizó varias operaciones Hash Join y Seq Scan para unir registros basados en id_equipo e id_persona, y recorrió un volumen considerable de filas, especialmente en la tabla Asignación (750 registros). A pesar del número de combinaciones procesadas, el tiempo de respuesta fue rápido, y el uso de Hash Join permitió un buen rendimiento sin necesidad de índices. Para mejorar aún más esta vista en entornos con datos masivos, se recomienda indexar id_equipo en Asignación y Proyecto.

```
689  EXPLAIN ANALYZE SELECT * FROM vista_becarios_proyectos;
690  ----
691
```

Data Output | Messages | Notifications

	QUERY PLAN text
1	Hash Join (cost=26.70..59.03 rows=328 width=268) (actual time=0.019..0.021 rows=0 loops=1)
2	Hash Cond: ((a.id_equipo)::text = (pr.id_equipo_responsable)::text)
3	-> Hash Join (cost=11.07..31.29 rows=262 width=66) (actual time=0.019..0.021 rows=0 loops=1)
4	Hash Cond: ((b.id_persona)::text = (p.id_persona)::text)
5	-> Hash Join (cost=2.58..22.09 rows=262 width=54) (actual time=0.019..0.021 rows=0 loops=1)
6	Hash Cond: ((a.id_persona)::text = (b.id_persona)::text)
7	-> Seq Scan on asignacion a (cost=0.00..17.50 rows=750 width=48) (actual time=0.018..0.018 rows=...
8	-> Hash (cost=1.70..1.70 rows=70 width=6) (never executed)
9	-> Seq Scan on becario b (cost=0.00..1.70 rows=70 width=6) (never executed)
10	-> Hash (cost=6.00..6.00 rows=200 width=42) (never executed)
11	-> Seq Scan on persona p (cost=0.00..6.00 rows=200 width=42) (never executed)
12	-> Hash (cost=12.50..12.50 rows=250 width=250) (never executed)
13	-> Seq Scan on proyecto pr (cost=0.00..12.50 rows=250 width=250) (never executed)
14	Planning Time: 0.435 ms
15	Execution Time: 0.086 ms

La vista vista becarios proyectos fue evaluada con EXPLAIN ANALYZE, obteniendo un tiempo de ejecución de 0.086 milisegundos y un tiempo de planificación de 0.435 milisegundos, siendo la más rápida de todas las vistas analizadas. Esta vista une las tablas Becario, Asignacion, Proyecto y Persona, utilizando varias operaciones Hash Join y búsquedas secuenciales (Seq Scan). Aunque se recorrieron hasta 750 registros en la tabla Asignación, el plan de ejecución fue eficiente gracias a las relaciones claras entre claves foráneas. El sistema no necesitó ejecutar todos los escaneos de las tablas intermedias, lo cual demuestra una optimización adecuada por parte del motor de PostgreSQL. Este excelente desempeño la hace ideal para reportes frecuentes, aunque el uso de índices en id_persona e id_equipo podría garantizar tiempos similares en bases de datos más grandes

Conclusión

En conclusión, este proyecto de base de datos representa un sistema integral para la gestión de becarios y personal académico, desarrollado mediante un proceso metodológico que abarcó desde el modelado conceptual hasta la implementación técnica. Partiendo de un modelo Entidad-Relación detallado que capturó fielmente los requisitos del sistema, se realizó una cuidadosa conversión al modelo relacional con normalización hasta tercera forma normal (3FN), garantizando la eliminación de redundancias y anomalías. La implementación final incorporó mecanismos avanzados como triggers para validar reglas de negocio complejas - incluyendo el control de solapamientos entre actividades, la restricción de servicio social único y la validación de carreras e instituciones autorizadas - junto con un robusto sistema de roles y permisos que asegura la integridad y confidencialidad de los datos. El diseño resultante no solo cumple con los requisitos actuales, sino que ofrece flexibilidad para futuras expansiones, gracias a su estructura normalizada y vistas estratégicamente diseñadas. La combinación de un diseño teórico sólido con consideraciones prácticas de rendimiento y seguridad ha dado como resultado un sistema confiable que optimiza los procesos administrativos, facilita la generación de reportes y proporciona una base estable para la toma de decisiones institucionales, demostrando la importancia de seguir un proceso sistemático en el desarrollo de bases de datos que va desde el análisis conceptual hasta la implementación técnica, considerando siempre tanto las necesidades actuales como los posibles requerimientos futuros.

Referencias

Database Star. (2022, 18 octubre). *pgAdmin Tutorial - How to Use pgAdmin* [Vídeo].YouTube. https://www.youtube.com/watch?v=WFT5MaZN6g4

TecnoBinaria. (2018, 19 abril). *Como hacer un «disparador» y su función - TRIGGER | PostgreSQL #34* [Vídeo]. YouTube. https://www.youtube.com/watch?v=KT6TJ7NChcg

TecnoBinaria. (2018b, mayo 2). *Como crear una vista - VIEW | PostgreSQL #38* [Vídeo]. YouTube. https://www.youtube.com/watch?v=9mK2ex6MkvA

José Antonio Pereira. (2020, 27 mayo). *SGBD: ejemplos de VISTAS y TRIGGERS en POSTGRESQL* [Vídeo]. YouTube. https://www.youtube.com/watch?v=jNtGhweJUQE