TAREA 2: MODELOS ENTIDAD-RELACION

Fundamentos de base de datos

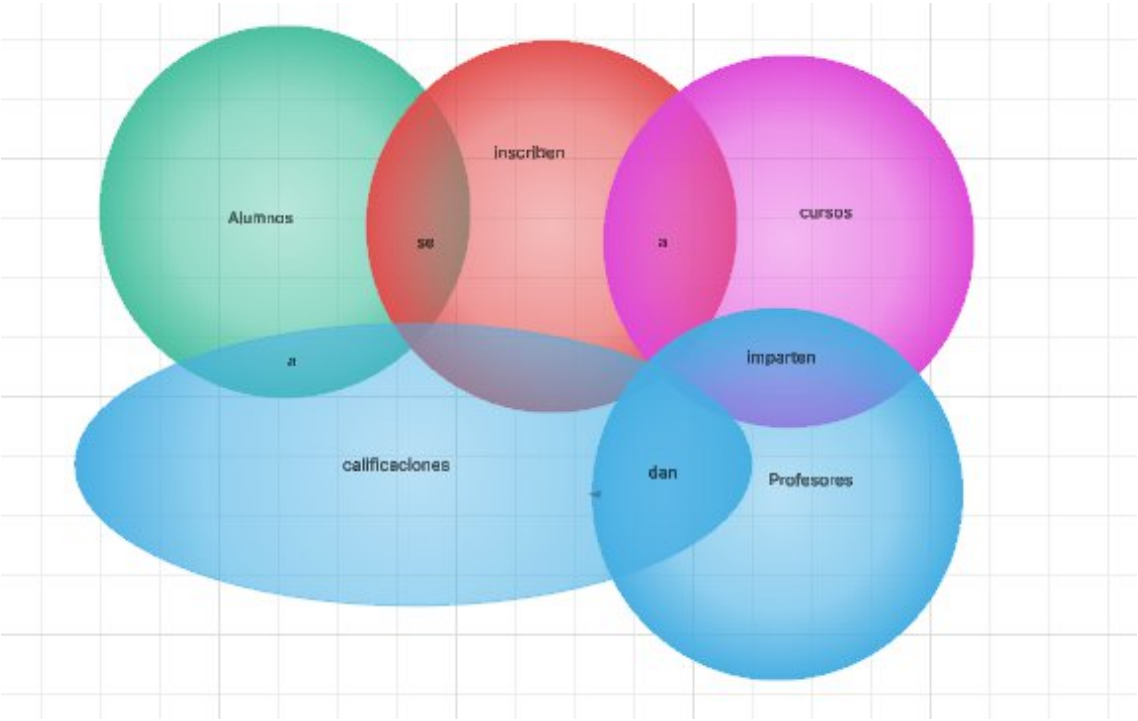
22 DE FEBRERO DE 2025

PROF. JOSE RAMON BOGARIN VALENZUELA
Damian Alexander Lugo Aguilar
Erick Hidekio Armas Diaz
Joel Alberto Revilla Moraila

Gestión Escolar

1.Las tablas principales.

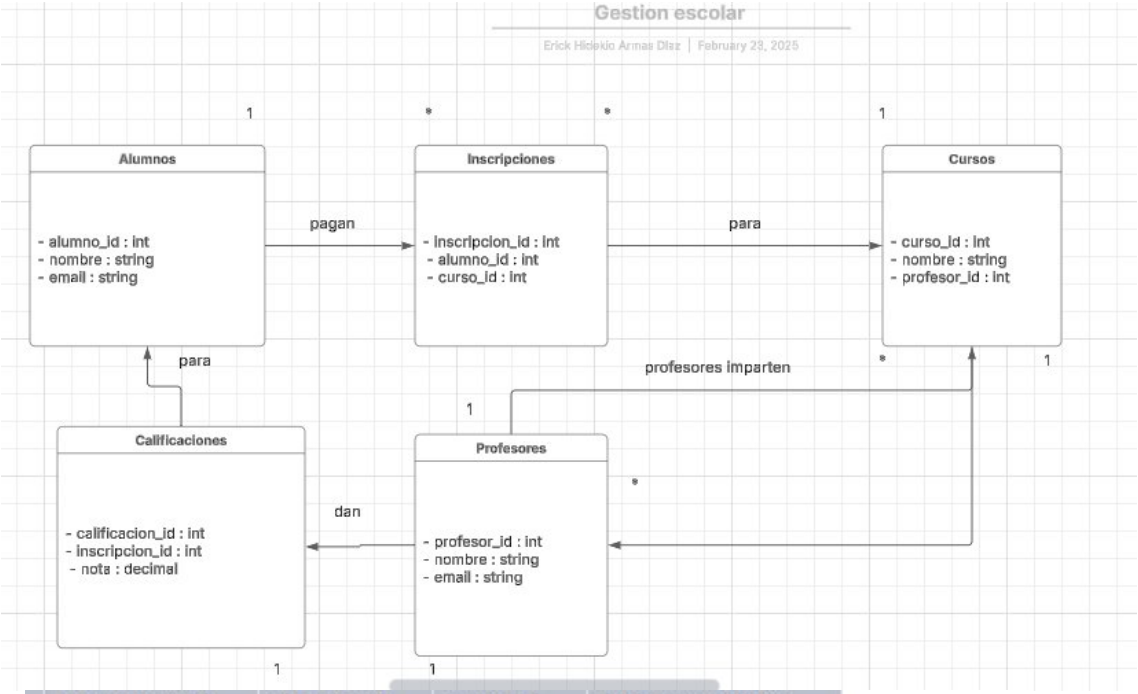
Son Alumnos y Profesores por que para que haya Alumnos debe de haber profesores y para que haya profesores se necesitan alumnos y la relación con las tablas de curso y calificaciones la mostrare a mas detalle con diagrama de ven.



La relación clave entre las tablas son las siguientes

Alumnos se inscriben en Cursos a través de Inscripciones. (Relación Muchos a Muchos resuelta con tabla intermedia) Profesores imparten Cursos. (Relación 1) Cada Inscripción puede tener una Calificación. (Relación 1:1)

UML



inscripcion_id	alumno_id	curso_id	fecha_inscripcion
[PK] integer	integer	integer	date

curso_id	nombre	descripcion	profesor_id
[PK] integer	character varying (100)	text	integer

inscripcion_id	alumno_id	curso_id	fecha_inscripcion
[PK] integer	integer	integer	date

calificacion_id	inscripcion_id	nota
[PK] integer	integer	numeric (3,1)

2.Compras en ONLINE.

En esta tabla son importantes la de **usuarios,pedidos y productos** ya que si no hubiera pedidos no habría necesidad de tener productos se necesitan productos para hacer pedidos y usuarios para que haya pedidos por ende al haber pedidos habría productos asi que la tabla intermedia terminaría siendo la de pedidos y las principales usuarios y productos.

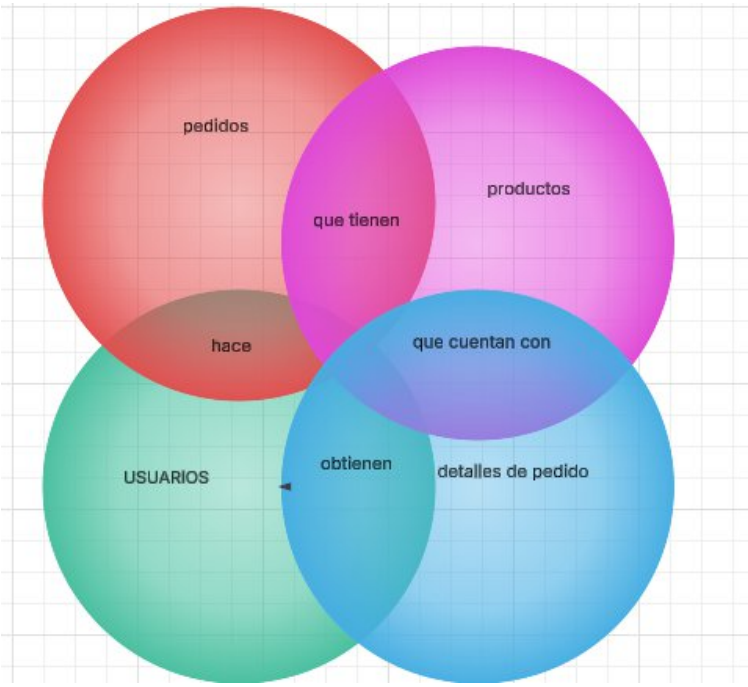
Relaciones Clave.

Usuarios a Pedidos

Un usuario puede tener muchos pedidos pero un pedido solo puede pertenecer a un Usuario la relación entre este es de 1 a muchos y de 1 a muchos.

Pedidos y productos la relación entre ambos es de Muchos a Muchos ya que un pedido puede tener varios productos y varios productos pueden pertenecer a varios pedidos.

Diagramas de ven.

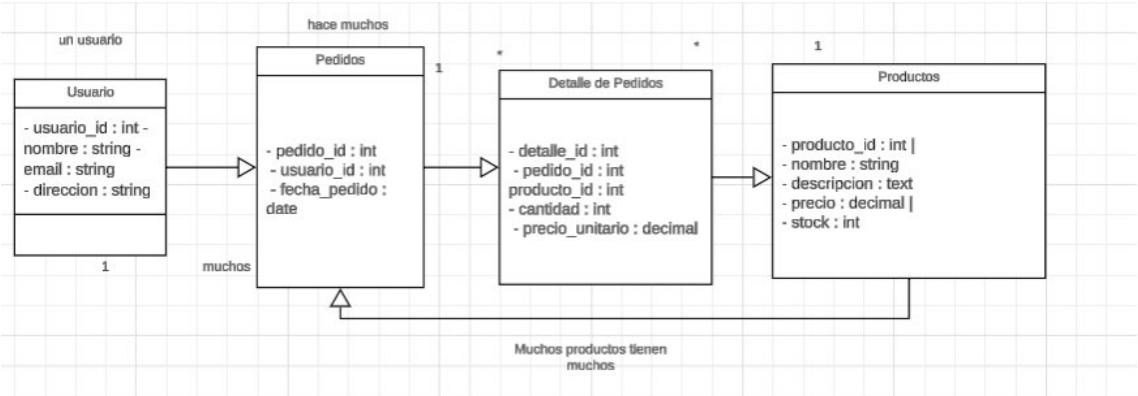


	table_name name
1	detallepedido
2	productos
3	pedidos

ta Output Messages Notifications

cliente_id [PK] integer	nombre character varying (100)	email character varying (100)	direccion character varying (255)

UML



3.Mensajeria.

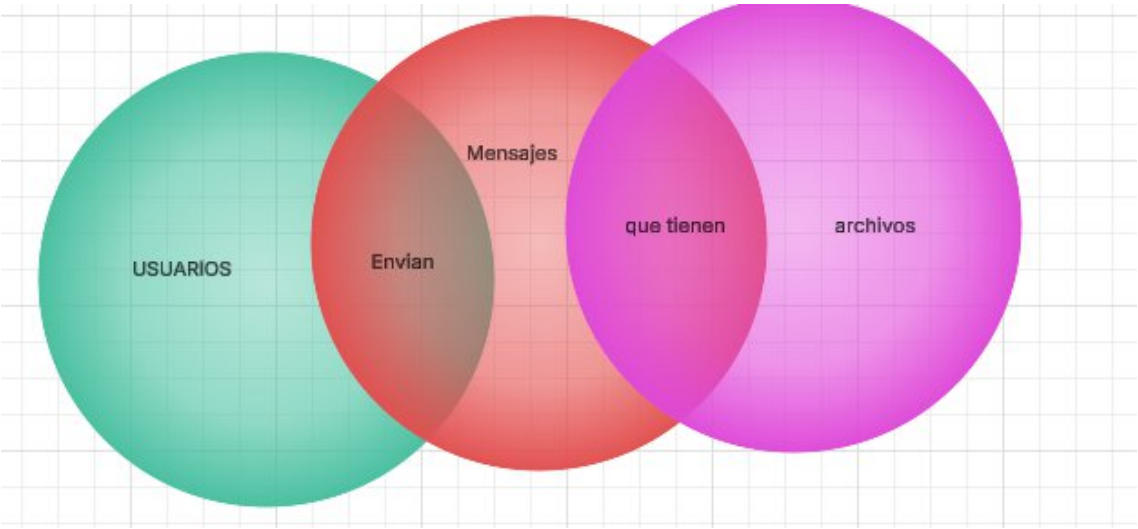
Las Tablas de Mensajería y Adjuntos ya que están muy enlazados al momento de enviar un mensaje que incluya algún archivo este se almacena en la tabla de adjuntos al igual usuarios es la conexión entre mensajes y adjuntos.

La Relaciones Claves son.

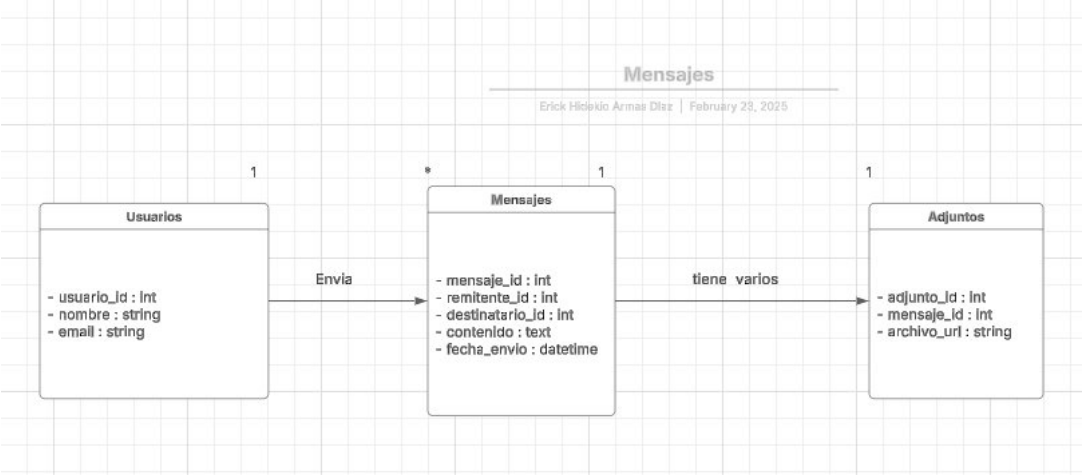
Se relaciona con Usuarios a través de remitente y destinatario.

Se conecta con Adjuntos si el mensaje tiene archivos.

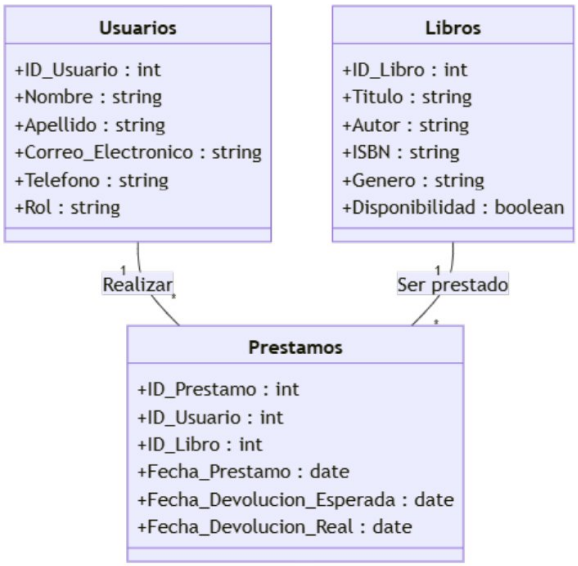
Diagramas de ven.



UML



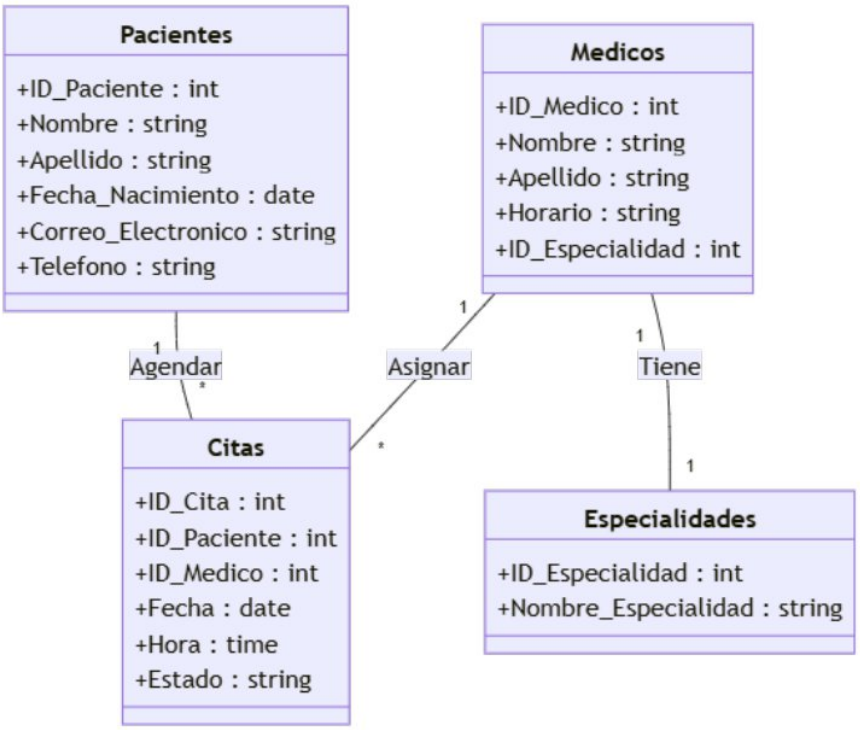
Biblioteca



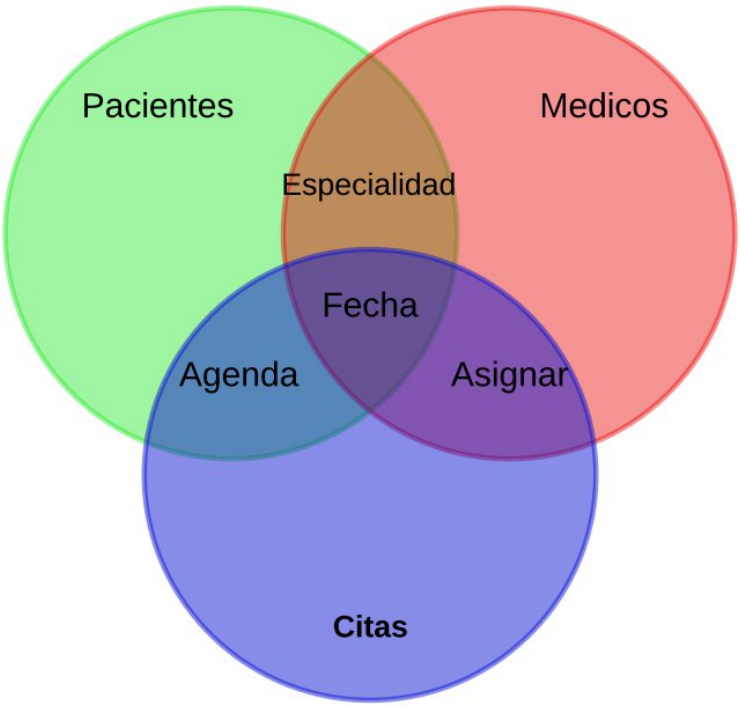
Biblioteca

```sql
-- Creación de tablas para el sistema de Biblioteca Digital
CREATE TABLE Usuarios (
    ID_Usuario INT PRIMARY KEY,
    Nombre VARCHAR(100),
    Apellido VARCHAR(100),
    EsAdministrador BOOLEAN DEFAULT FALSE
);

CREATE TABLE Libros (
    ID_Libro INT PRIMARY KEY,
    Titulo VARCHAR(200),
    Autor VARCHAR(100),
    Disponibilidad BOOLEAN DEFAULT TRUE
);

CREATE TABLE Prestamos (
    ID_Prestamo INT PRIMARY KEY,
    ID_Usuario INT,
    ID_Libro INT,
    Fecha_Prestamo DATE,
    Fecha_Devolucion_Esperada DATE,
    Fecha_Devolucion DATE,
    FOREIGN KEY (ID_Usuario) REFERENCES Usuarios(ID_Usuario),
    FOREIGN KEY (ID_Libro) REFERENCES Libros(ID_Libro)
);

-- Consulta para listar los préstamos activos de un usuario específico
SELECT Prestamos.ID_Prestamo, Libros.Titulo AS Titulo_Libro, Libros.Autor AS Autor_Libro,
       Prestamos.Fecha_Prestamo, Prestamos.Fecha_Devolucion_Esperada
FROM Prestamos
JOIN Libros ON Prestamos.ID_Libro = Libros.ID_Libro
WHERE Prestamos.ID_Usuario = 1 AND Prestamos.Fecha_Devolucion IS NULL;
```

Citas Medicas



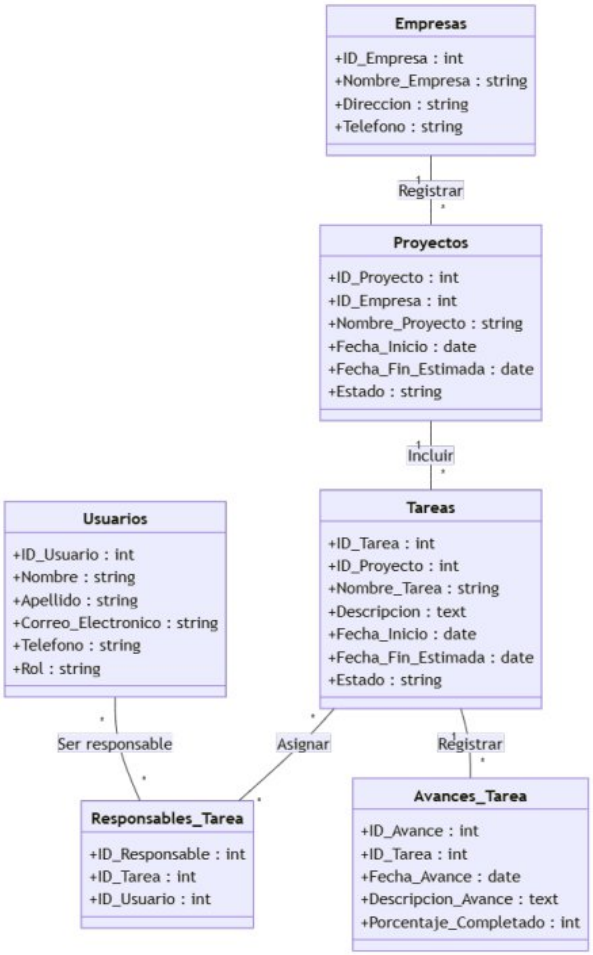
Citas Medicas



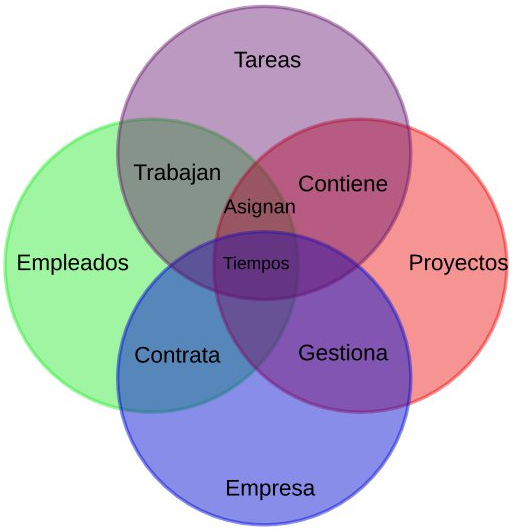

```sql
-- Creación de tablas para el sistema de gestión de citas médicas
CREATE TABLE Pacientes (
    ID_Paciente INT PRIMARY KEY,
    Nombre VARCHAR(100),
    Apellido VARCHAR(100),
    Fecha_Nacimiento DATE
);

CREATE TABLE Medicos (
    ID_Medico INT PRIMARY KEY,
    Nombre VARCHAR(100),
    Apellido VARCHAR(100),
    Especialidad VARCHAR(100)
);

CREATE TABLE Horarios (
    ID_Horario INT PRIMARY KEY,
    ID_Medico INT,
    Fecha DATE,
    Hora_Inicio TIME,
    Hora_Fin TIME,
    FOREIGN KEY (ID_Medico) REFERENCES Medicos(ID_Medico)
);

CREATE TABLE Citas (
    ID_Cita INT PRIMARY KEY,
    ID_Paciente INT,
    ID_Medico INT,
    Fecha DATE,
    Hora TIME,
    Estado VARCHAR(50) CHECK (Estado IN ('pendiente', 'atendida', 'cancelada')),
    FOREIGN KEY (ID_Paciente) REFERENCES Pacientes(ID_Paciente),
    FOREIGN KEY (ID_Medico) REFERENCES Medicos(ID_Medico)
);

-- Consulta para listar las citas pendientes de un médico específico
SELECT Citas.ID_Cita, Pacientes.Nombre AS Nombre_Paciente, Pacientes.Apellido AS Apellido_Paciente,
       Citas.Fecha, Citas.Hora, Citas.Estado
```

```sql
FROM Citas
JOIN Pacientes ON Citas.ID_Paciente = Pacientes.ID_Paciente
WHERE Citas.ID_Medico = 1 AND Citas.Estado = 'pendiente';
```

Gestion de Proyectos



Gestion de Proyectos



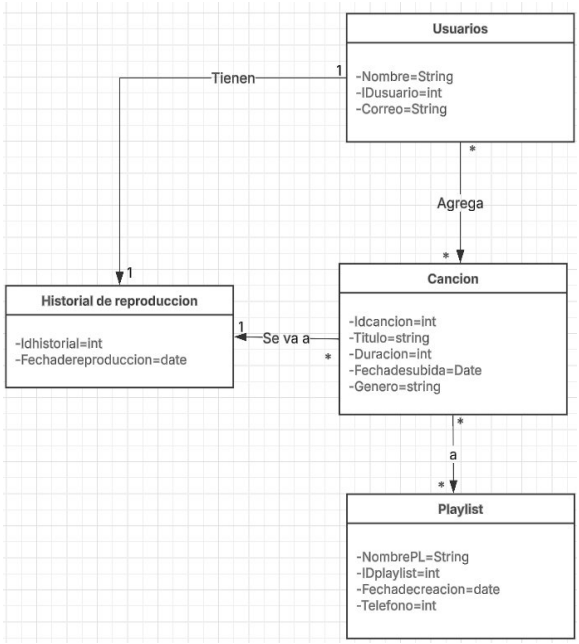
```sql
-- Creación de tablas para el sistema de Gestión de Proyectos
CREATE TABLE Empresas (
    ID_Empresa INT PRIMARY KEY,
    Nombre VARCHAR(100)
);

CREATE TABLE Usuarios (
    ID_Usuario INT PRIMARY KEY,
    Nombre VARCHAR(100),
    Apellido VARCHAR(100),
    ID_Empresa INT,
    FOREIGN KEY (ID_Empresa) REFERENCES Empresas(ID_Empresa)
);

CREATE TABLE Proyectos (
    ID_Proyecto INT PRIMARY KEY,
    Nombre VARCHAR(200),
    Descripcion TEXT,
    ID_Empresa INT,
    FOREIGN KEY (ID_Empresa) REFERENCES Empresas(ID_Empresa)
);

CREATE TABLE Tareas (
    ID_Tarea INT PRIMARY KEY,
    Nombre VARCHAR(200),
    Descripcion TEXT,
    Estado VARCHAR(50) CHECK (Estado IN ('pendiente', 'en progreso', 'completada')),
    Avance INT CHECK (Avance BETWEEN 0 AND 100),
    ID_Proyecto INT,
    ID_Responsable INT,
    FOREIGN KEY (ID_Proyecto) REFERENCES Proyectos(ID_Proyecto),
    FOREIGN KEY (ID_Responsable) REFERENCES Usuarios(ID_Usuario)
);

-- Consulta para listar las tareas pendientes de un proyecto específico
SELECT Tareas.ID_Tarea, Tareas.Nombre AS Titulo_Tarea, Tareas.Descripcion,
       Tareas.Estado, Tareas.Avance, Usuarios.Nombre AS Responsable
FROM Tareas
JOIN Usuarios ON Tareas.ID_Responsable = Usuarios.ID_Usuario
WHERE Tareas.ID_Proyecto = 1 AND Tareas.Estado = 'pendiente';
```

Sistema de gestion de proyectos

Los requerimientos que requiere en este ejercicio son los siguientes:

- Los usuarios pueden crear playlists con canciones
- Los artistas pueden subir canciones
- Se debe almacenar el historial de reproducción

Con estos puntos podemos encontrar nuestras entidades principales: Usuarios, canciones, playlists e historial de reproducción. ahora con esto tenemos que hacer las uniones de entidad-relacion y los atributos de cada uno:



Las relaciones son las siguientes:

- Usuarios a cancion= muchos a muchos
- Cancion a playlist= muchos a muchos
- Usarios a historial= 1 a 1

Nuestras claves primarias seran:

- IDusuario
- IDcancion
- IDplaylist
- IDhistorial

Una vez teniendo esto podemos hacer nuestros diagramas de venn para tener de una mejor manera y mas simple de ver todo los datos

Una vez teniendo un poco claro como es el acomodo de todo podemos hcaer el query que nos quedaria asi:

```sql
CREATE TABLE Cancion (
    IDcancion SERIAL PRIMARY KEY,
    titulo VARCHAR(100) NOT NULL,
    duracion INT NOT NULL,
    fechadesubida DATE NOT NULL,
    genero VARCHAR(100)
);

CREATE TABLE Usuario (
    IDusuario SERIAL PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    correo VARCHAR(100) UNIQUE NOT NULL
);

CREATE TABLE Playlist (
    IDplaylist SERIAL PRIMARY KEY,
    nombredeplaylist VARCHAR(100) NOT NULL,
    fecha_creacion DATE NOT NULL,
    IDusuario INT,
    FOREIGN KEY (IDusuario) REFERENCES Usuario(IDusuario) ON DELETE CASCADE
);

CREATE TABLE HistorialReproduccion (
    IDhistorial SERIAL PRIMARY KEY,
    fechareproduccion TIMESTAMP NOT NULL,
    IDusuario INT,
    IDcancion INT,
    FOREIGN KEY (IDusuario) REFERENCES Usuario(IDusuario) ON DELETE CASCADE,
    FOREIGN KEY (IDcancion) REFERENCES Cancion(IDcancion) ON DELETE CASCADE
)
```

Las tablas se verian asi:

	idusuario [PK] integer	nombre character varying (100)	correo character varying (100)
0+	110	Damian	correo@gmail.com

Tabla de usuarios

	idcancion [PK] integer	titulo character varying (100)	duracion integer	fechadesubida date	genero character varying (100)
0+	1023	Cancion buena	3:15	15/2/19	Electronica

Tabla de canciones

	idhistorial [PK] integer	fechareproduccion timestamp without time zone	idusuario integer	idcancion integer
0+	1101	17/2/25	110	1023

Tabla de historial de reproduccion

	idplaylist [PK] integer	nombredeplaylist character varying (100)	fecha_creacion date	idusuario integer
0+	515	Musicafavorita	12/11/24	110

Tabla de playlist

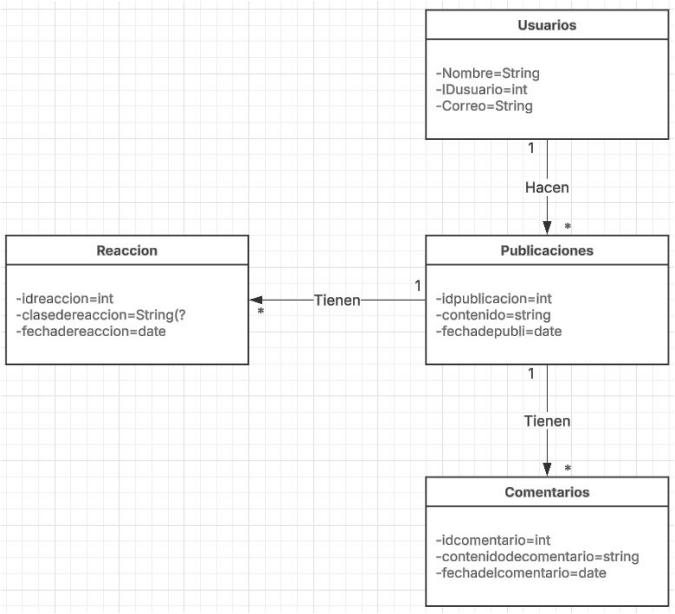
Conclusion: En este ejercicio tenemos 4 entidades importantes: Usuarios, canciones, historial de reproduccion y playlist. Todos los usuarios pueden agreagar muchas canciones mientras que muchas canciones pueden ser agreagadas por todos los usuarios dando a entender que su relacion es de muchos a muchos, muchas canciones pueden ser agregadas a muchas playlist e viceversa por lo tanto su relacion es de muchos a muchos, por ulitmo tenemos una tabla pivote que es lista de reproduccion la cual esta unida con usuario y canciones, con usuario tiene una relacion de de 1 a 1 mientras que con canciones tiene una relacion de 1 a muchos.

Red social

Los requerimientos para este ejercicio seran los siguientes:

- Los usuarios pueden hacer publicaciones y reaccionar a ellas
- Se debe almacenar el historial de interacciones
- Los usuarios pueden seguir a otros usuarios

Una vez teniendo estos requerimientos podemos encontrar nuestras entidades principales: Usuarios, publicaciones, comentarios y reacciones. Teniendo esto podemos crear tanto nuestro diagrama entidad relacion y diagrama de venn, primero empezaremos con el diagrama de entidad relacion que es el siguiente:



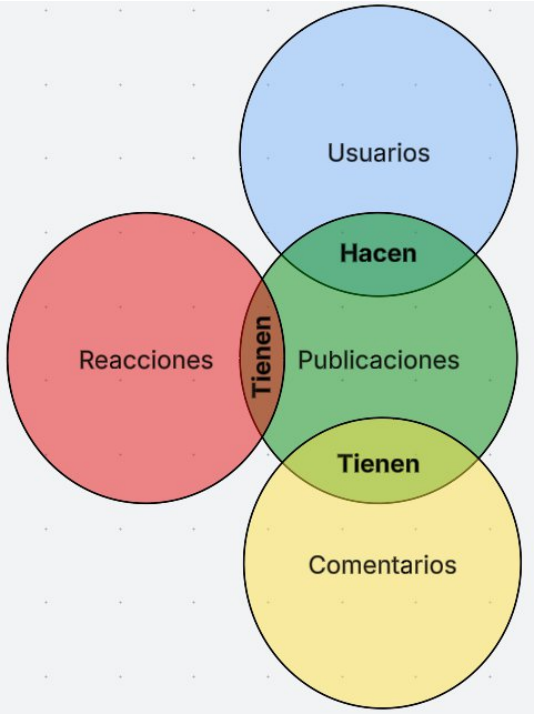
Las relaciones de las entidades son las siguientes:

•Usuarios a publicaciones= 1 a muchos
•Publicaciones a comentarios= 1 a muchos
•Publicaiones a reacciones= 1 a muchos

Con esto tambien podemos sacar nuestras claves primarias que son:

•IDusuario
•IDpublicacion
•IDcomentario
•IDreaccion

Ahora teniendo esto hacemos nuestro diagrama de venn para que sea mas visible:



Una vez hecho tanto el diagrama venn como el de entidad-relacion podemos hacernos la idea de como acomodar todo en el query, quedandonos asi:

```sql
CREATE TABLE Usuario (
    IDusuario SERIAL PRIMARY KEY,
    nombreusuario VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL
);

CREATE TABLE Publicacion (
    IDpublicacion SERIAL PRIMARY KEY,
    contenido TEXT NOT NULL,
    fechapublicacion TIMESTAMP NOT NULL,
    IDusuario INT,
    FOREIGN KEY (IDusuario) REFERENCES Usuario(IDusuario) ON DELETE CASCADE
);

CREATE TABLE Comentario (
    IDcomentario SERIAL PRIMARY KEY,
    contenidocomentario TEXT NOT NULL,
    fechadelcomentario TIMESTAMP NOT NULL,
    IDusuario INT,
    IDpublicacion INT,
    FOREIGN KEY (IDusuario) REFERENCES Usuario(IDusuario) ON DELETE CASCADE,
    FOREIGN KEY (IDpublicacion) REFERENCES Publicacion(IDpublicacion) ON DELETE CASCADE
);

CREATE TABLE Reaccion (
    IDreaccion SERIAL PRIMARY KEY,
    clasedereaccion VARCHAR(50) NOT NULL,
    fechadereaccion TIMESTAMP NOT NULL,
    IDusuario INT,
    IDpublicacion INT,
    FOREIGN KEY (IDusuario) REFERENCES Usuario(IDusuario) ON DELETE CASCADE,
    FOREIGN KEY (IDpublicacion) REFERENCES Publicacion(IDpublicacion) ON DELETE CASCADE
)
```

Las tablas se ven asi:

	idusuario [PK] integer	nombreusuario character varying (100)	email character varying (100)
0+	404	Damian	correo2@mail.com

-Tabla de usuario

	idpublicacion [PK] integer	contenido text	fechapublicacion timestamp without time zone	idusuario integer
0+	220	Perritos	25/03/24	404

-Tabla de publicacion

	idcomentario [PK] integer	contenidocomentario text	fechadelcomentario timestamp without time zone	idusuario integer	idpublicacion integer
0+	300	Perro bonito	25/03/24	103	220

-Tabla de comentario

	idreaccion [PK] integer	clasedereaccion character varying (50)	fechadereaccion timestamp without time zone	idusuario integer	idpublicacion integer
0+	646	Me_encanta	25/03/24	103	220

-Tabla de reaccion

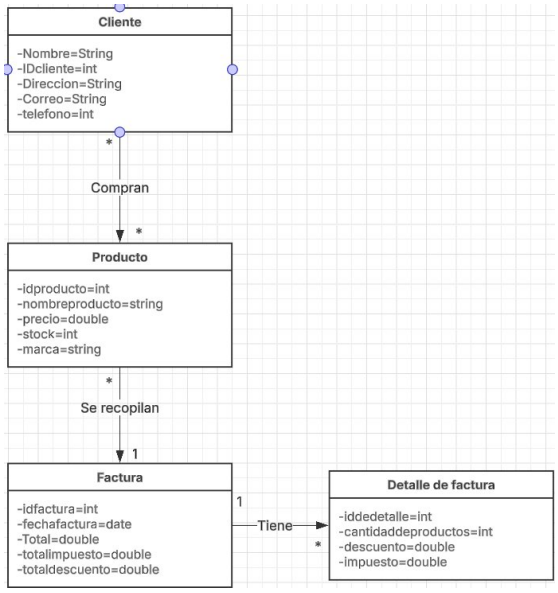
Conclusion: Este ejercicio podia tener diferentes formas de hacerse en las cuales en su mayoria terminaban en la creacion de mas tablas pivote asi que las dejamos de una forma que sea simple de hacer y entender, es sencillo de explicar: Usuarios pueden hacer muchas publicaciones, las publicaciones pueden tener muchas reacciones y las publicaciones tambien pueden tener muchos comentarios, podemos agregar mas cosas como dije al principio por ejemplo el hecho de que los muchos usuarios pueden hacer solo 1 reaccion, tambien identificamos que nuestras entidades principales son usuarios y comentarios y nuestra entidad de union es publicacion.

Sistema de facturacion

Por ultimo en este ejercicio se nos dan los siguientes requerimientos:

- Generar facturas para clientes por productos comprados
- Registrar impuestos y descuentos
- Asociar cada factura a un cliente

Nuestras entidades principales seran: Clientes, facturas, detalles de factura y productos. Con esto podremos crear el ultimo diagrama entidad-relacion indicando las los atributos clave para cada entidad, nos quedaria asi:



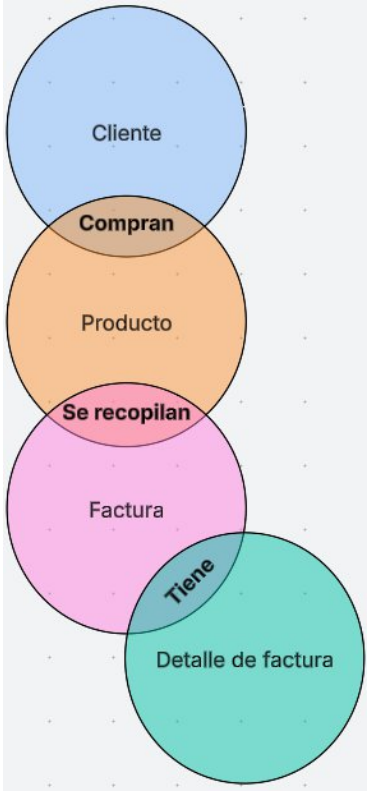
Como podemos ver las relaciones entre las entidades son las siguientes:

- Cliente a producto= muchos a muchos
- Producto a factura= muchos a 1
- Factura a detalle de factura= 1 a muchos

Mientras que las claves primarias son:

- IDcliente
- IDproducto
- IDfactura
- IDdetalle

Con esto pasamos a la creacion del ultimo diagrama de venn el cual es el siguiente:



Teniendo todo esto finalmente podemos pasar a la creacion del query el cual nos queda asi:

```sql
CREATE TABLE Cliente (
    IDcliente SERIAL PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    direccion TEXT,
    telefono VARCHAR(20),
    correo VARCHAR(100) UNIQUE NOT NULL
);

CREATE TABLE Producto (
    IDproducto SERIAL PRIMARY KEY,
    nombreproducto VARCHAR(255) NOT NULL,
    marca TEXT,
    precio DECIMAL(10, 2) NOT NULL,
    stock INT NOT NULL
);

CREATE TABLE Factura (
    IDfactura SERIAL PRIMARY KEY,
    fechafactura TIMESTAMP NOT NULL,
    IDcliente INT,
    total DECIMAL(10, 2) NOT NULL,
    totalimpuesto DECIMAL(10, 2),
     total_descuento DECIMAL(10, 2),
    FOREIGN KEY (IDcliente) REFERENCES Cliente(IDcliente) ON DELETE CASCADE
);

CREATE TABLE DetalleFactura (
    IDdetalle SERIAL PRIMARY KEY,
    IDfactura INT,
    IDproducto INT,
    cantidaddeproductos INT NOT NULL,
    descuento DECIMAL(10, 2),
    impuesto DECIMAL(10, 2),
    FOREIGN KEY (IDfactura) REFERENCES Factura(IDfactura) ON DELETE CASCADE,
    FOREIGN KEY (IDproducto) REFERENCES Producto(IDproducto) ON DELETE CASCADE
)
```

Las tablas se verian asi una vez hechas:

	idcliente [PK] integer	nombre character varying (100)	direccion text	telefono character varying (20)	correo character varying (100)
0+	333	Damian	calle 2	6862971143	correo2@gmail.com

–Tabla cliente

	idproducto [PK] integer	nombreproducto character varying (255)	marca text	precio numeric (10,2)	stock integer
0+	227	Agua natural ciel	Ciel	23.50	28

–Tabla producto

	idfactura [PK] integer	fechafactura timestamp without time zone	idcliente integer	total numeric (10,2)	totalimpuesto numeric (10,2)	totaldescuento numeric (10,2)
0+	503	23/02/25	333	23.50	1.50	0.00

–Tabla factura

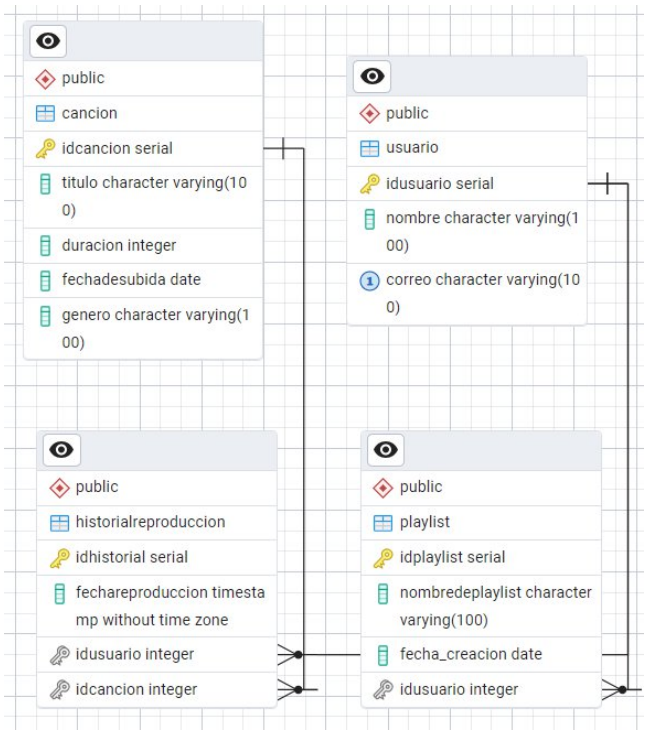
	iddetalle [PK] integer	idfactura integer	idproducto integer	cantidaddeproductos integer	descuento numeric (10,2)	impuesto numeric (10,2)
0+	523	503	227	1	1.50	0.00

–Tabla detalle de factura

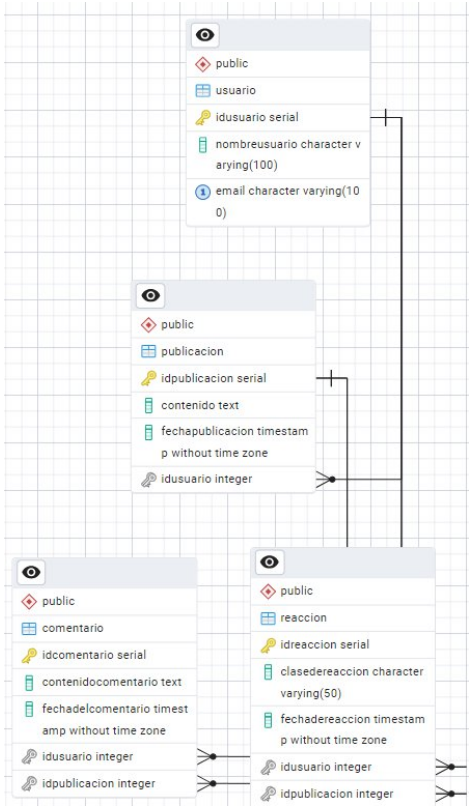
Conclusion: No hay mucho mas que agregar en este ejercicio, todo va directo al grano, muchos clientes compran muchos productos, muchos productos se recopilan en una sola factura, una sola factura puede tener muchos detalles de factura, nuestras entidades principales en este caso son cliente y factura.

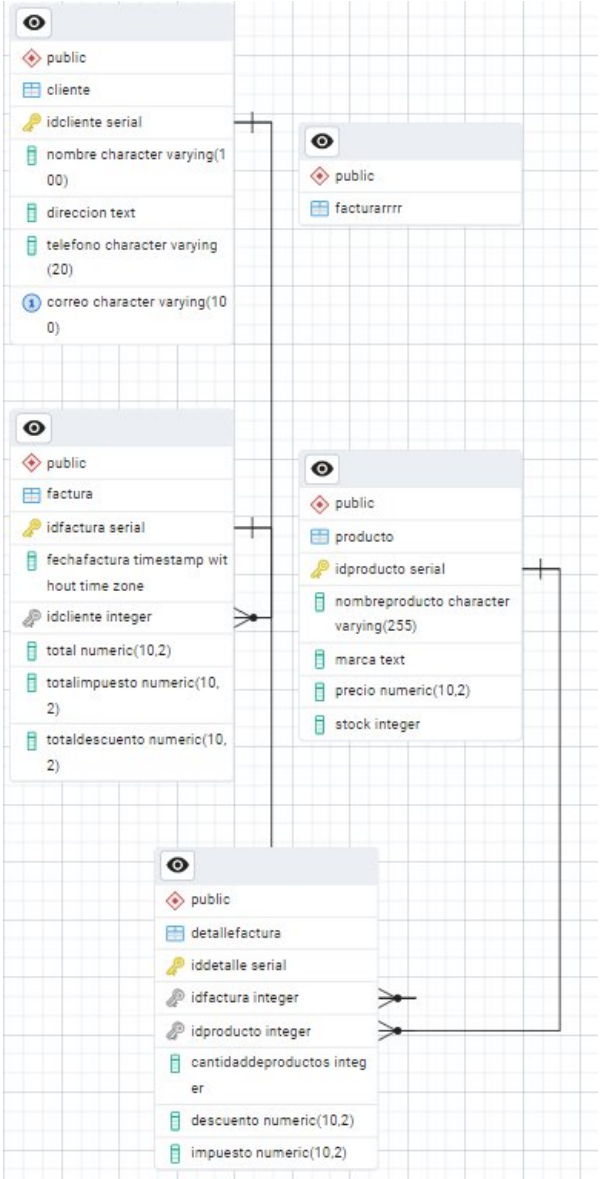
Como pequeño agregado me acabo de dar cuenta que el programa de pgAdmin hace un diagrama de entidad-relacion una vez hecho todo el query, aqui los diagramas que salen con los ejercicios que hice:

Streaming de musica:



Red social:



Y por ultimo

Sistema de facturacion