



INSTITUTO TECNOLÓGICO DE MEXICALI

Base De Datos

Tarea 3 BD



ALUMNOS: FRANCISCO RAMOS VAZQUEZ

MAESTRA: JOSE RAMON BOGARIN VALENZUELA

CARRERA: INGENIERIA EN SISTEMAS COMPUTACIONALES

SEMESTRE: 4

## 1.-Sistema de gestión de Inventarios

Entidades Claves

**Producto**

**Proveedor**

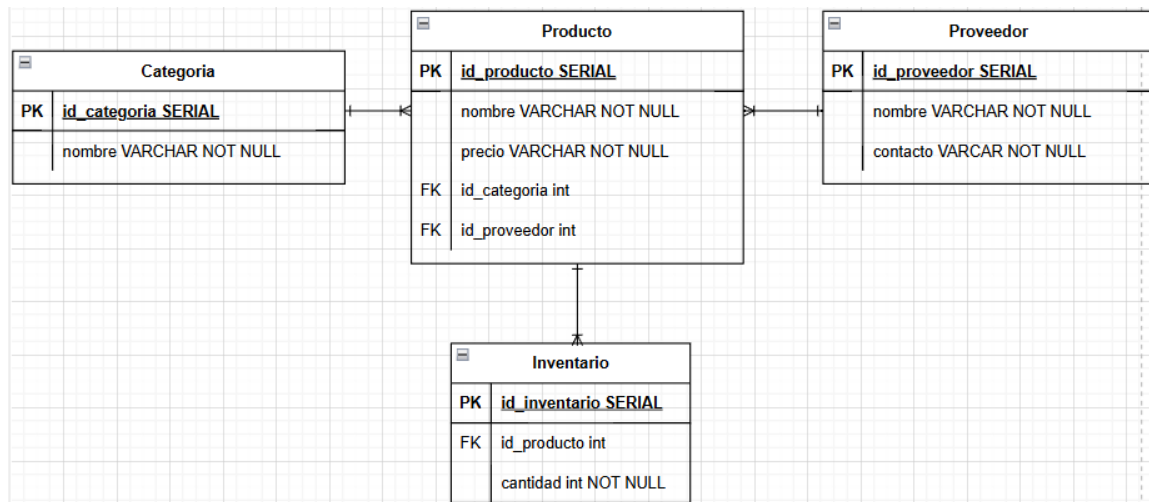
**Categoría**

**Inventario**

**Relaciones:**

- Un **Producto** pertenece a una **Categoría**.
- Un **Producto** es suministrado por un **Proveedor**.
- Un **Producto** está registrado en el **Inventario**.

Diseño del Modelo Conceptual y Conversión a Esquema Relacional



## Implementación en SQL




```
CREATE TABLE Categoria (  
    id_categoria SERIAL PRIMARY KEY,  
    nombre VARCHAR(100) NOT NULL  
);  
  
CREATE TABLE Proveedor (  
    id_proveedor SERIAL PRIMARY KEY,  
    nombre VARCHAR(100) NOT NULL,  
    contacto VARCHAR(100)  
);  
  
CREATE TABLE Producto (  
    id_producto SERIAL PRIMARY KEY,  
    nombre VARCHAR(100) NOT NULL,  
    precio DECIMAL(10,2) NOT NULL,  
    id_categoria INT REFERENCES Categoria(id_categoria),  
    id_proveedor INT REFERENCES Proveedor(id_proveedor)  
);  
  
CREATE TABLE Inventario (  
    id_inventario SERIAL PRIMARY KEY,  
    id_producto INT REFERENCES Producto(id_producto),  
    cantidad INT NOT NULL  
)
```

## 5. Consultas SQL

Consulta requerida: Obtener la lista de productos con sus respectivas categorías y proveedores, ordenados alfabéticamente por nombre de producto.

```
SELECT p.nombre AS producto, c.nombre AS categoria, pr.nombre AS proveedor  
FROM Producto p  
INNER JOIN Categoria c ON p.id_categoria = c.id_categoria  
INNER JOIN Proveedor pr ON p.id_proveedor = pr.id_proveedor  
ORDER BY p.nombre ASC;
```

## 6. Validaciones

	<input type="text"/> producto 	<input type="text"/> categoria 	<input type="text"/> proveedor 
1	Camiseta	Ropa	Proveedor B
2	Televisor	Electrónica	Proveedor A

## 2. Sistema de Gestión de Eventos

### Entidades Claves

**Evento**

**Participante**

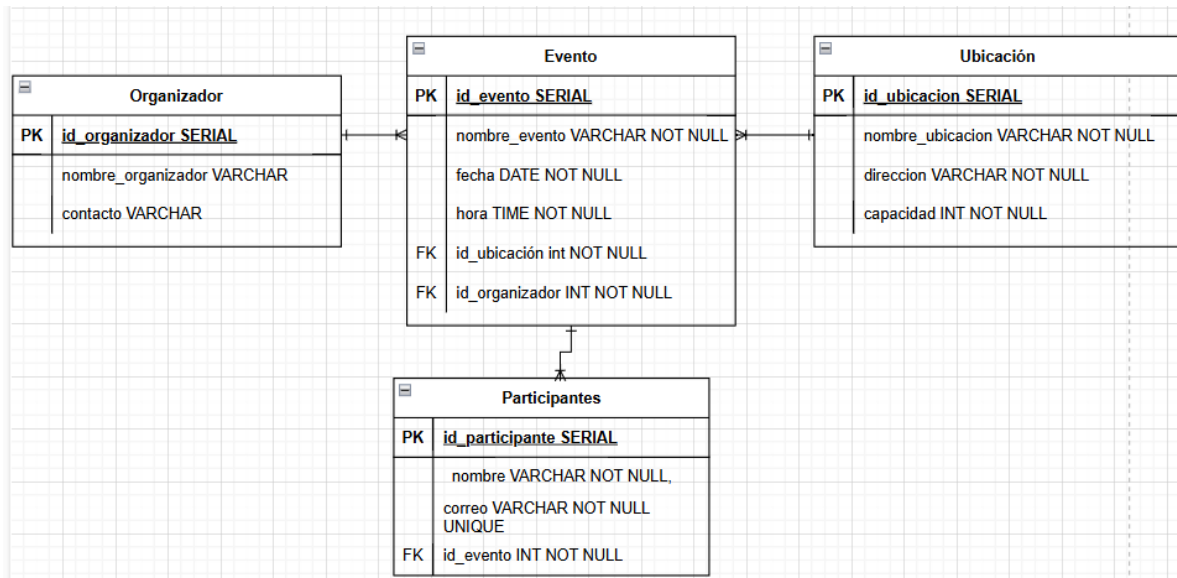
**Ubicación**

**Organizador**

**Relaciones:**

- **Un Evento tiene una Ubicación.**
- **Un Evento es organizado por un Organizador.**
- **Un Participante puede asistir a múltiples Eventos** (relación muchos a muchos).
- **Un Evento puede tener múltiples Participantes** (relación muchos a muchos).

### Diseño del Modelo Conceptual y Conversión a Esquema Relacional



## Implementación en SQL

```
CREATE TABLE Organizador (
    id_organizador SERIAL PRIMARY KEY,
    nombre_organizador VARCHAR(100) NOT NULL,
    contacto VARCHAR(100) NOT NULL
);

CREATE TABLE Ubicacion (
    id_ubicacion SERIAL PRIMARY KEY,
    nombre_ubicacion VARCHAR(100) NOT NULL,
    direccion VARCHAR(200) NOT NULL,
    capacidad INT NOT NULL
);

CREATE TABLE Evento (
    id_evento SERIAL PRIMARY KEY,
    nombre_evento VARCHAR(100) NOT NULL,
    fecha DATE NOT NULL,
    hora TIME NOT NULL,
    id_ubicacion INT NOT NULL,
    id_organizador INT NOT NULL,
    CONSTRAINT fk_ubicacion FOREIGN KEY (id_ubicacion) REFERENCES Ubicacion(id_ubicacion),
    CONSTRAINT fk_organizador FOREIGN KEY (id_organizador) REFERENCES Organizador(id_organizador)
);

CREATE TABLE Participante (
    id_participante SERIAL PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    correo VARCHAR(100) NOT NULL UNIQUE,
    id_evento INT NOT NULL,
    CONSTRAINT fk_evento FOREIGN KEY (id_evento) REFERENCES Evento(id_evento)
);
```

## 5. Consultas SQL

Consulta requerida: Obtener la lista de eventos programados junto con la cantidad de participantes registrados por evento.

```
SELECT e.nombre_evento, COUNT(p.id_participante) AS cantidad_participantes, e.fecha
FROM Evento e
LEFT JOIN Participante p ON e.id_evento = p.id_evento
GROUP BY e.id_evento, e.fecha
ORDER BY e.fecha;

SELECT
    e.nombre_evento,
    e.fecha,
    o.nombre_organizador
FROM Evento e
JOIN Organizador o ON e.id_organizador = o.id_organizador
WHERE o.nombre_organizador = 'Eventos X'
ORDER BY e.fecha DESC;

SELECT p.nombre, p.correo, e.nombre_evento
FROM Participante p
JOIN Evento e ON p.id_evento = e.id_evento
WHERE p.nombre LIKE '%Carlos%'
ORDER BY p.nombre;
```

6. Validaciones

Consulta del Problema

	nombre_evento	cantidad_participantes	fecha
1	Conferencia de Innovación	2	2025-05-10
2	Expo Tecnología 2025	2	2025-06-15
3	Feria de Emprendedores	2	2025-07-20
4	Festival Cultural	2	2025-08-05
5	Cumbre Empresarial	2	2025-09-18

	nombre_evento	fecha	nombre_organizador
1	Conferencia de Innovación	2025-05-10	Eventos X

	nombre	correo	nombre_evento
1	Carlos Ruiz	carlos.ruiz@mail.com	Expo Tecnología 2025

### 3. Plataforma de Streaming de Música

Entidades Claves

**Usuario**

**Artista**

**Álbum**

**Canción**

**Relaciones:**

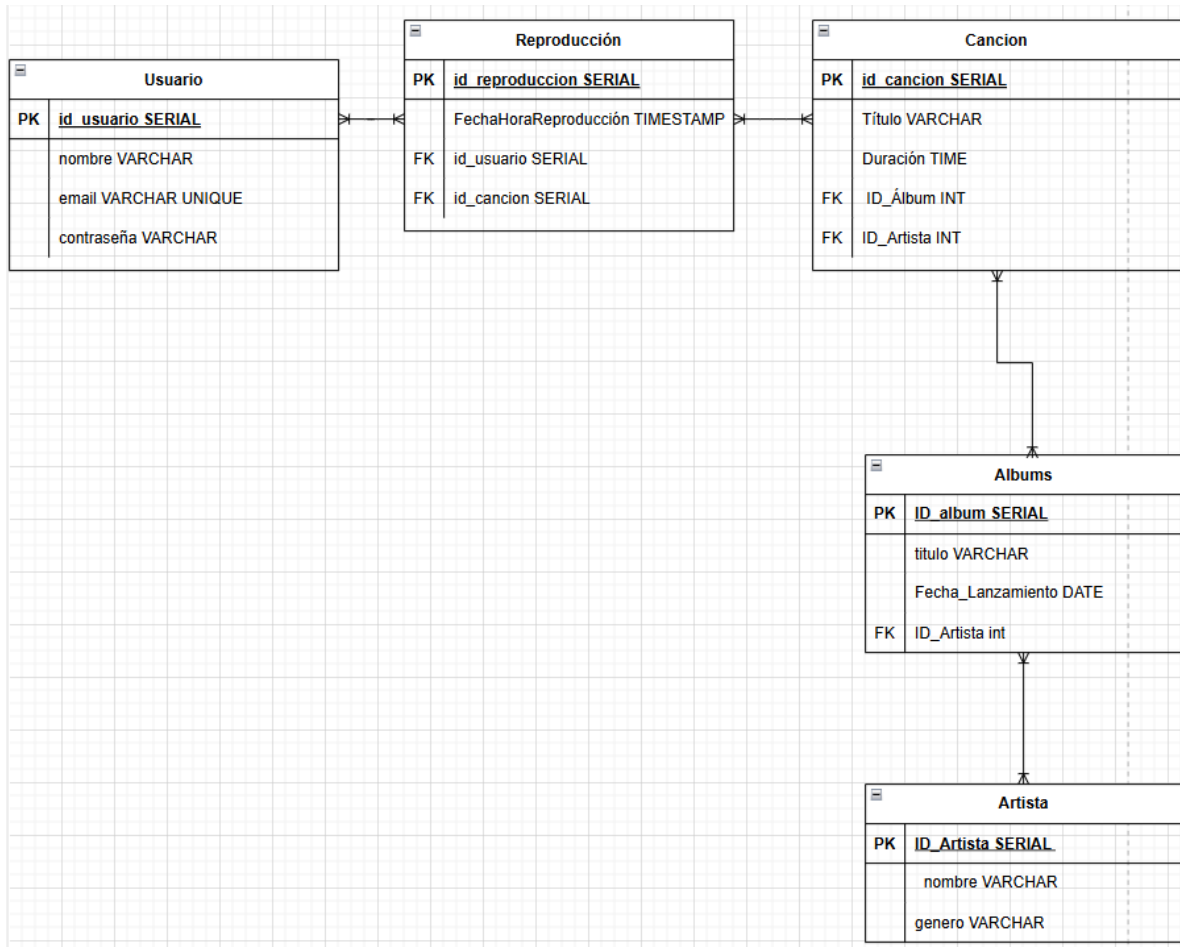
Un usuario puede reproducir varias canciones, y cada canción puede ser reproducida por muchos usuarios (relación muchos a muchos).

Un artista puede tener múltiples álbumes.

Un álbum contiene varias canciones.

**Diseño del Modelo Conceptual y Conversión a Esquema Relacional**





## Implementación en SQL

```
CREATE TABLE Usuarios (  
    ID_Usuario SERIAL PRIMARY KEY,  
    Nombre VARCHAR(100),  
    Email VARCHAR(100) UNIQUE,  
    Contraseña VARCHAR(100)  
);  
  
CREATE TABLE Artistas (  
    ID_Artista SERIAL PRIMARY KEY,  
    Nombre VARCHAR(100),  
    Género VARCHAR(50)  
);  
  
CREATE TABLE Álbumes (  
    ID_Álbum SERIAL PRIMARY KEY,  
    Título VARCHAR(100),  
    Fecha_Lanzamiento DATE,  
    ID_Artista INT REFERENCES Artistas(ID_Artista)  
);  
  
CREATE TABLE Canciones (  
    ID_Canción SERIAL PRIMARY KEY,  
    Título VARCHAR(100),  
    Duración TIME,  
    ID_Álbum INT REFERENCES Álbumes(ID_Álbum),  
    ID_Artista INT REFERENCES Artistas(ID_Artista)  
);  
  
CREATE TABLE Reproducción (  
    ID_Usuario INT REFERENCES Usuarios(ID_Usuario),  
    ID_Canción INT REFERENCES Canciones(ID_Canción),  
    FechaReproducción TIMESTAMP,  
    PRIMARY KEY (ID_Usuario, ID_Canción)  
);
```

## 5. Consultas SQL

Consulta requerida: Listar las canciones reproducidas por un usuario específico

```
SELECT e.nombre_evento, COUNT(p.id_participante) AS cantidad_participantes, e.fecha
FROM Evento e
LEFT JOIN Participante p ON e.id_evento = p.id_evento
GROUP BY e.id_evento, e.fecha
ORDER BY e.fecha;

SELECT
    e.nombre_evento,
    e.fecha,
    o.nombre_organizador
FROM Evento e
JOIN Organizador o ON e.id_organizador = o.id_organizador
WHERE o.nombre_organizador = 'Eventos X'
ORDER BY e.fecha DESC;

SELECT p.nombre, p.correo, e.nombre_evento
FROM Participante p
JOIN Evento e ON p.id_evento = e.id_evento
WHERE p.nombre LIKE '%Carlos%'
ORDER BY p.nombre;
```

Consulta para listar artistas y el número de canciones que han sido reproducidas por un usuario específico

```
SELECT
    A.Nombre AS Artista,
    COUNT(C.ID_Canción) AS Canciones_Reproducidas
FROM
    Reproducción R
    INNER JOIN Canciones C ON R.ID_Canción = C.ID_Canción
    INNER JOIN Artistas A ON C.ID_Artista = A.ID_Artista
WHERE
    R.ID_Usuario = 6
GROUP BY
    A.ID_Artista
ORDER BY
    Canciones_Reproducidas DESC;
```

Consulta para buscar canciones cuyo título contenga una palabra específica y listarlas por artista, ordenadas alfabéticamente

```
SELECT
  C.Título AS Canción,
  A.Nombre AS Artista,
  Al.Título AS Álbum
FROM
  Canciones C
  INNER JOIN Artistas A ON C.ID_Artista = A.ID_Artista
  INNER JOIN Álbumes Al ON C.ID_Álbum = Al.ID_Álbum
WHERE
  C.Título LIKE '%Lights%'
ORDER BY
  C.Título ASC;
```

## 6. Validaciones

	<input type="text"/> canción <input type="text"/>	<input type="text"/> artista <input type="text"/>	<input type="text"/> Álbum <input type="text"/>
1	Hips Don't Lie	Shakira	El Dorado
2	Despacito	Luis Fonsi	Despacito

	<input type="text"/> artista <input type="text"/>	<input type="text"/> canciones_reproducidas <input type="text"/>
1	The Weeknd	1

	<input type="text"/> canción <input type="text"/>	<input type="text"/> artista <input type="text"/>	<input type="text"/> Álbum <input type="text"/>
1	Blinding Lights	The Weeknd	After Hours

## 4. Sistema de Control de Proyectos

Entidades Claves

**Proyecto**

**Empleado**

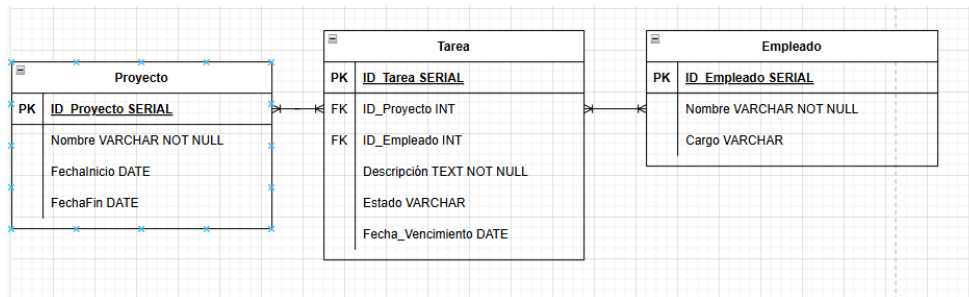
**Tarea**

**Relaciones:**

Un Proyecto puede tener muchas Tareas, pero cada Tarea pertenece a un solo Proyecto (relación 1 a muchos).

Un Empleado puede estar asignado a varias Tareas, pero una Tarea tiene un solo Empleado asignado (relación 1 a muchos).

**Diseño del Modelo Conceptual y Conversión a Esquema Relacional**



**Implementación en SQL**

```
CREATE TABLE Proyecto (  
    ID_Proyecto SERIAL PRIMARY KEY,  
    Nombre VARCHAR(100) NOT NULL,  
    FechaInicio DATE,  
    FechaFin DATE  
);  
  
CREATE TABLE Empleado (  
    ID_Empleado SERIAL PRIMARY KEY,  
    Nombre VARCHAR(100) NOT NULL,  
    Cargo VARCHAR(50)  
);  
  
CREATE TABLE Tarea (  
    ID_Tarea SERIAL PRIMARY KEY,  
    ID_Proyecto INT REFERENCES Proyecto(ID_Proyecto),  
    ID_Empleado INT REFERENCES Empleado(ID_Empleado),  
    Descripción TEXT NOT NULL,  
    Estado VARCHAR(20) CHECK (Estado IN ('Pendiente', 'En Progreso', 'Completada')),  
    Fecha_Vencimiento DATE
```

## 5. Consultas SQL

Mostrar todas las tareas pendientes de un proyecto específico, ordenadas por fecha de vencimiento:

```
SELECT t.Descripción, t.Estado, t.Fecha_Vencimiento, e.Nombre AS Empleado
FROM Tarea t
JOIN Proyecto p 1..n<->1: ON t.ID_Proyecto = p.ID_Proyecto
JOIN Empleado e 1..n<->1: ON t.ID_Empleado = e.ID_Empleado
WHERE p.ID_Proyecto = ? AND t.Estado = 'Pendiente'
ORDER BY t.Fecha_Vencimiento ASC;
```

Cantidad de tareas pendientes por cada proyecto

```
SELECT p.ID_Proyecto, p.Nombre AS Proyecto,
       COUNT(t.ID_Tarea) AS Tareas_Pendientes
FROM Proyecto p
JOIN Tarea t 1<->1..n: ON p.ID_Proyecto = t.ID_Proyecto
WHERE t.Estado = 'Pendiente'
GROUP BY p.ID_Proyecto, p.Nombre
ORDER BY Tareas_Pendientes DESC;
```

Buscar empleados con un nombre que contenga "an" y listar sus tareas

```
SELECT e.ID_Empleado, e.Nombre AS Empleado,
       t.ID_Tarea, t.Descripción, t.Estado, t.Fecha_Vencimiento
FROM Empleado e
JOIN Tarea t 1<->1..n: ON e.ID_Empleado = t.ID_Empleado
WHERE e.Nombre LIKE '%an%'
ORDER BY t.Fecha_Vencimiento;
```

6. Validaciones

	descripción	estado	fecha_vencimiento	empleado
1	Desarrollar módulo de autenticación	Pendiente	2025-03-15	Luis Ramírez

	id_proyecto	proyecto	tareas_pendientes
1	1	Desarrollo de Aplicación Móvil	1
2	3	Plataforma E-Learning	1
3	4	Automatización de Procesos	1
4	6	Rediseño del Sitio Web	1
5	8	Sistema de Facturación Electrónica	1

	id_proyecto	proyecto	tareas_pendientes
1	1	Desarrollo de Aplicación Móvil	1
2	3	Plataforma E-Learning	1
3	4	Automatización de Procesos	1
4	6	Rediseño del Sitio Web	1
5	8	Sistema de Facturación Electrónica	1

## 5.- Sistema de Evaluación Académica

### Entidades Claves

**Estudiante**

**Curso**

**Profesor**

**Calificación**

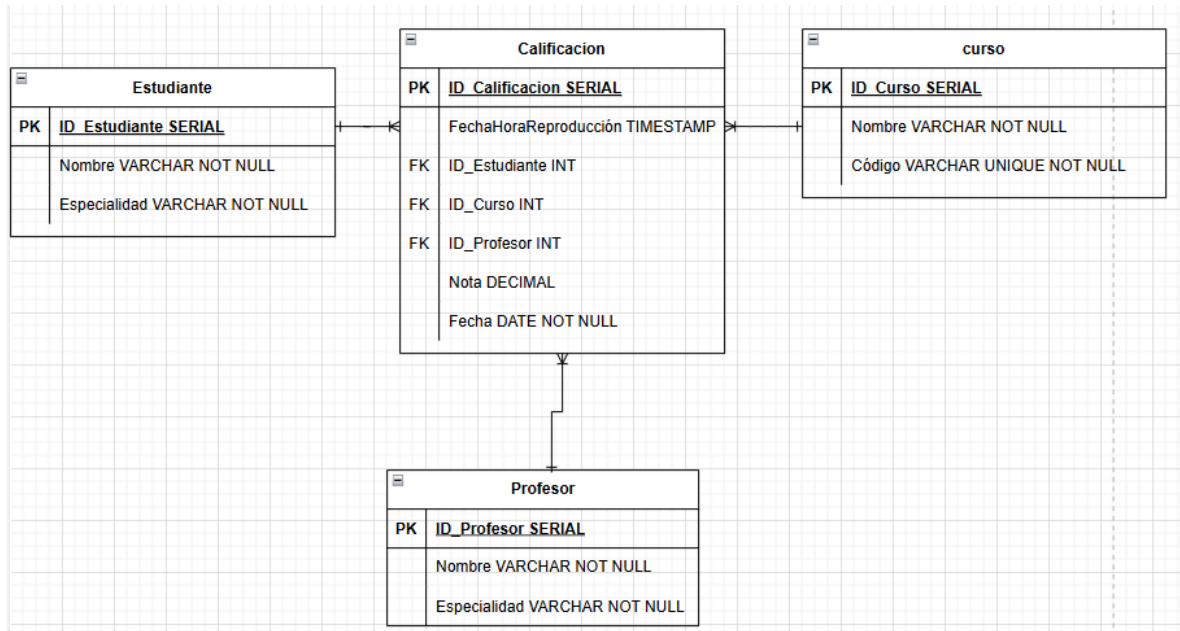
**Relaciones:**

Un Estudiante puede estar en varios Cursos y un Curso tiene varios Estudiantes (relación muchos a muchos).

Un Profesor puede impartir varios Cursos, pero un Curso tiene un solo Profesor (relación 1 a muchos).

Una Calificación pertenece a un Estudiante en un Curso y es asignada por un Profesor.

### Diseño del Modelo Conceptual y Conversión a Esquema Relacional





## Implementación en SQL

```
CREATE TABLE Estudiante (  
    ID_Estudiante SERIAL PRIMARY KEY,  
    Nombre VARCHAR(50) NOT NULL,  
    Apellido VARCHAR(50) NOT NULL,  
    Correo VARCHAR(100) UNIQUE NOT NULL  
);  
  
CREATE TABLE Profesor (  
    ID_Profesor SERIAL PRIMARY KEY,  
    Nombre VARCHAR(50) NOT NULL,  
    Especialidad VARCHAR(50) NOT NULL  
);  
  
CREATE TABLE Curso (  
    ID_Curso SERIAL PRIMARY KEY,  
    Nombre VARCHAR(100) NOT NULL,  
    Código VARCHAR(20) UNIQUE NOT NULL  
);  
  
CREATE TABLE Calificacion (  
    ID_Calificacion SERIAL PRIMARY KEY,  
    ID_Estudiante INT REFERENCES Estudiante(ID_Estudiante) ON DELETE CASCADE,  
    ID_Curso INT REFERENCES Curso(ID_Curso) ON DELETE CASCADE,  
    ID_Profesor INT REFERENCES Profesor(ID_Profesor) ON DELETE SET NULL,  
    Nota DECIMAL(5,2) CHECK (Nota >= 0 AND Nota <= 100),  
    Fecha DATE NOT NULL  
);
```

## 5. Consultas SQL

Obtener el promedio de calificaciones de un estudiante en todos sus cursos.

```
SELECT e.Nombre, e.Apellido, AVG(c.Nota) AS Promedio  
FROM Calificacion c  
JOIN Estudiante e 1..n<->1: ON c.ID_Estudiante = e.ID_Estudiante  
WHERE e.ID_Estudiante = ?  
GROUP BY e.ID_Estudiante;
```

Promedio de calificaciones de cada estudiante en sus cursos

```
SELECT e.ID_Estudiante, e.Nombre, e.Apellido,  
       c.Nombre AS Curso, AVG(cal.Nota) AS Promedio  
FROM Calificacion cal  
JOIN Estudiante e 1..n<->1: ON cal.ID_Estudiante = e.ID_Estudiante  
JOIN Curso c 1..n<->1: ON cal.ID_Curso = c.ID_Curso  
GROUP BY e.ID_Estudiante, e.Nombre, e.Apellido, c.Nombre  
ORDER BY Promedio DESC;
```

Listar los estudiantes que tienen un apellido que empieza con "L" y sus calificaciones

```
SELECT e.ID_Estudiante, e.Nombre, e.Apellido,
       c.Nombre AS Curso, cal.Nota, cal.Fecha
FROM Calificacion cal
JOIN Estudiante e 1..n<->1: ON cal.ID_Estudiante = e.ID_Estudiante
JOIN Curso c 1..n<->1: ON cal.ID_Curso = c.ID_Curso
WHERE e.Apellido LIKE 'L%'
ORDER BY cal.Fecha DESC;
```

6. Validaciones

	nombre ▾	apellido ▾	promedio ▾
1	Carlos	Gómez	78

	id_estudiante ▾	nombre ▾	apellido ▾	curso ▾	promedio ▾
1	8	Laura	Sánchez	Inglés Avanzado	95
2	4	María	Ruiz	Historia Universal	92
3	2	Ana	López	Física General	90
4	7	Pedro	Ramírez	Ética y Filosofía	89
5	5	José	Martínez	Literatura Clásica	88
6	10	Marta	Ortiz	Macroeconomía	87
7	1	Juan	Pérez	Álgebra	85
8	9	Diego	Castro	Programación en C++	80
9	3	Carlos	Gómez	Química Orgánica	78
10	6	Lucía	Fernández	Biología Celular	75

	id_estudiante ▾	nombre ▾	apellido ▾	curso ▾	nota ▾	fecha ▾
1	2	Ana	López	Física General	90.00	2025-03-11