

Modelo Relacional: Conceptos Fundamentales y Aplicaciones Detalladas

1. Introducción al Modelo Relacional

El **Modelo Relacional** es la base sobre la cual se construyen la mayoría de las bases de datos modernas. Fue propuesto por **Edgar F. Codd** en 1970 como una forma de organizar y gestionar datos que fuera más eficiente, flexible y fácil de entender que los modelos anteriores, como los modelos jerárquico y de red.

¿Qué es una relación?

En términos simples, una **relación** en el modelo relacional es lo que comúnmente conocemos como una **tabla** en una base de datos. Cada tabla almacena datos sobre un tipo específico de entidad, como `Estudiantes`, `Cursos` o `Profesores`.

- **Filas (Tuplas):** Cada fila en una tabla representa una **tupla**, que es un solo registro o instancia de la entidad que la tabla describe. Por ejemplo, en la tabla `Estudiantes`, cada fila podría representar a un estudiante individual, con su nombre, apellido, fecha de nacimiento, etc.
- **Columnas (Atributos):** Las columnas de la tabla representan los **atributos** de la entidad. Un atributo es una característica o propiedad de la entidad que estamos almacenando. Por ejemplo, en la tabla `Estudiantes`, las columnas podrían ser `ID_Estudiante`, `Nombre`, `Apellido`, y `FechaNacimiento`. Cada columna tiene un tipo de dato específico que define qué tipo de información se puede almacenar en esa columna, como números enteros, cadenas de texto, o fechas.

¿Por qué es importante el modelo relacional?

El modelo relacional es extremadamente importante porque ofrece una forma lógica y consistente de organizar los datos, lo que facilita la manipulación y recuperación de grandes volúmenes de información de manera eficiente. Además, al estar basado en la teoría de conjuntos, permite realizar operaciones matemáticas y lógicas sobre los datos, lo que es esencial para el análisis y procesamiento de información en sistemas complejos.

2. Tipos de Relaciones

Dentro del modelo relacional, podemos identificar varios tipos de relaciones, que se diferencian según cómo se definen y de dónde provienen sus datos:

- **Relación Base:**
 - **Definición:** Es una tabla que tiene un nombre propio y almacena datos directamente. Es una estructura fundamental y autónoma, lo que significa que no depende de ninguna otra tabla para existir.
 - **Ejemplo:** La tabla `Estudiantes` es una relación base porque almacena los datos directamente (como el nombre y la fecha de nacimiento de cada estudiante) y tiene un nombre propio que la identifica en la base de datos.
- **Relación Derivada:**

- **Definición:** Es una tabla que no almacena datos propios, sino que se deriva de una o más relaciones base mediante una consulta o una expresión de álgebra relacional. En SQL, las **vistas** son ejemplos de relaciones derivadas.
- **Ejemplo:** Supongamos que tienes una vista llamada `EstudiantesActivos` que muestra solo los estudiantes que están actualmente inscritos en cursos. Esta vista no almacena los datos, sino que los recupera de la tabla base `Estudiantes` mediante una consulta SQL.
- **Instantánea:**
 - **Definición:** Es un tipo de relación derivada que captura y almacena datos en un momento específico del tiempo, pero no es actualizable. Aunque tiene datos asociados, estos datos no son propios, sino que reflejan el estado de otras relaciones en un momento dado.
 - **Ejemplo:** Una vista que captura las inscripciones en cursos a principios de cada semestre podría ser considerada una instantánea. Esta vista reflejaría quién estaba inscrito en cada curso al comienzo del semestre, pero no cambiaría si se actualizan las inscripciones después.
- **Resultados de Consultas:**
 - **Definición:** Son relaciones derivadas temporales que existen solo mientras dura la sesión de la base de datos en la que se ejecutó la consulta. No tienen un nombre propio y desaparecen una vez que termina la sesión o se cierra la conexión.
 - **Ejemplo:** Si ejecutas una consulta SQL que selecciona todos los estudiantes que aprobaron un curso específico, los resultados de esa consulta forman una relación derivada temporal que existe solo mientras la consulta está activa.

3. Relaciones y Predicados

En el modelo relacional, no solo es importante almacenar datos, sino también definir **reglas** que los datos deben cumplir para mantener la integridad y coherencia de la información.

- **Relaciones y su Interpretación:**
 - **Definición:** Cada relación en una base de datos relacional tiene un significado específico. Por ejemplo, la tabla `Estudiantes` podría representar "todos los estudiantes matriculados en la universidad".
 - **Importancia:** Definir claramente lo que cada relación representa es crucial para asegurar que los datos almacenados sean consistentes con la realidad que intentan modelar.
- **Predicado:**
 - **Definición:** Un predicado es una expresión lógica que, al evaluarse, devuelve verdadero o falso. En el contexto de las bases de datos, un predicado se utiliza para determinar si una tupla (una fila) puede ser parte de una relación.

- **Ejemplo:** En la tabla `Estudiantes`, un predicado podría ser "la fecha de nacimiento debe ser anterior a la fecha actual". Este predicado asegura que no se puedan ingresar fechas de nacimiento futuras, lo que mantendría la coherencia de los datos.
 - **Proposición:**
 - **Definición:** Una proposición es un predicado aplicado a una tupla específica. Cuando se inserta o actualiza una fila en una tabla, la proposición se evalúa para determinar si la fila cumple con las reglas de la relación.
 - **Ejemplo:** Si intentas agregar un nuevo estudiante con una fecha de nacimiento en el futuro, la proposición basada en el predicado "la fecha de nacimiento debe ser anterior a la fecha actual" fallará, y la base de datos rechazará la inserción.
-

4. Integridad y Reglas en el Modelo Relacional

Las bases de datos relacionales imponen varias reglas de integridad para asegurar que los datos sean consistentes y correctos:

- **Integridad de Entidad:**
 - **Definición:** Esta regla asegura que cada relación (tabla) tenga una clave primaria y que no haya dos filas en la tabla con la misma clave primaria. Esto garantiza que cada registro sea único.
 - **Ejemplo:** En la tabla `Estudiantes`, `ID_Estudiante` debe ser único para cada estudiante. No pueden existir dos estudiantes con el mismo `ID_Estudiante`, lo que asegura que cada registro se pueda identificar de manera única.
 - **Integridad Referencial:**
 - **Definición:** La integridad referencial asegura que las claves foráneas en una tabla apunten siempre a valores válidos en otra tabla. Esto significa que si una tabla tiene una clave foránea que apunta a la clave primaria de otra tabla, no se pueden insertar o actualizar valores en la clave foránea que no existan en la tabla referenciada.
 - **Ejemplo:** Si tienes una tabla `Inscripciones` que relaciona estudiantes con cursos, la columna `ID_Estudiante` en `Inscripciones` debe referenciar un `ID_Estudiante` existente en la tabla `Estudiantes`. No se puede inscribir a un estudiante en un curso si ese estudiante no existe en la tabla `Estudiantes`.
-

5. Claves Candidatas y Claves Primarias

En el modelo relacional, las **claves** son esenciales para identificar de manera única las filas dentro de una tabla:

- **Claves Candidatas:**

- **Definición:** Son subconjuntos de atributos que podrían actuar como una clave primaria porque cumplen con las propiedades de unicidad y minimalidad. Es decir, no hay dos filas en la tabla con los mismos valores para estos atributos, y no se puede eliminar ningún atributo de la clave sin perder esta propiedad.
 - **Ejemplo:** En una tabla de *Personas*, tanto el número de identificación (ID) como el número de seguro social podrían ser claves candidatas, ya que ambos identifican de manera única a una persona.
 - **Claves Primarias:**
 - **Definición:** De todas las claves candidatas, se selecciona una para ser la clave primaria. La clave primaria identifica de manera única cada fila de la tabla y no puede contener valores nulos.
 - **Ejemplo:** En la tabla *Estudiantes*, *ID_Estudiante* se selecciona como la clave primaria porque identifica de manera única a cada estudiante y es un valor que siempre estará presente.
 - **Claves Simples y Compuestas:**
 - **Claves Simples:** Una clave que consiste en un solo atributo.
 - **Claves Compuestas:** Una clave que consiste en dos o más atributos. La combinación de estos atributos debe ser única para cada fila.
 - **Ejemplo:** En la tabla *Inscripciones*, podrías tener una clave compuesta formada por *ID_Estudiante* y *ID_Curso*, que juntos identifican de manera única una inscripción específica.
-

6. Claves Foráneas

Las **claves foráneas** son esenciales para mantener la integridad referencial en las bases de datos relacionales:

- **Definición:** Una clave foránea es un atributo en una tabla que hace referencia a la clave primaria de otra tabla. Las claves foráneas permiten establecer y reforzar las relaciones entre diferentes tablas.
 - **Ejemplo:** En la tabla *Inscripciones*, *ID_Estudiante* podría ser una clave foránea que apunta a *ID_Estudiante* en la tabla *Estudiantes*. Esto asegura que cada inscripción esté vinculada a un estudiante existente.
-

7. Diagramas de Entidad-Relación vs Diagramas de Base de Datos

- **Diagramas de Entidad-Relación (ERD):**
 - **Definición:** Son representaciones gráficas que muestran las entidades de una base de datos (por ejemplo, *Estudiantes*, *Cursos*) y las relaciones entre ellas. Los ERD se utilizan principalmente durante la fase de diseño conceptual para representar cómo se relacionan los diferentes elementos de la base de datos.

- **Componentes:** Entidades (tablas), relaciones (cómo las tablas están conectadas), atributos (columnas), y cardinalidad (relaciones uno a uno, uno a muchos, etc.).
- **Diagramas de Base de Datos:**
 - **Definición:** Son diagramas técnicos que detallan cómo se implementan las entidades y relaciones en la base de datos física. Estos diagramas incluyen detalles sobre tipos de datos, claves primarias y foráneas, índices, y otros aspectos técnicos.
 - **Diferencia con ERD:** Mientras que los ERD se centran en la estructura conceptual, los diagramas de base de datos muestran cómo se implementa esa estructura en la base de datos real, incluyendo detalles específicos de la implementación en SQL Server u otras plataformas.

Ejemplo Práctico Completo

Para integrar todos estos conceptos, consideremos una base de datos académica:

1. Tabla Estudiantes:

- Definición:

```
CREATE TABLE Estudiantes (
    ID_Estudiante INT PRIMARY KEY,
    Nombre VARCHAR(50),
    Apellido VARCHAR(50),
    FechaNacimiento DATE
);
```

- **Explicación:** Esta tabla representa a los estudiantes. La clave primaria ID_Estudiante asegura que cada estudiante se identifique de manera única.

2. Tabla Cursos:

- Definición:

```
CREATE TABLE Cursos (
    ID_Curso INT PRIMARY KEY,
    NombreCurso VARCHAR(100)
);
```

- **Explicación:** Esta tabla contiene los cursos ofrecidos. Cada curso tiene un identificador único ID_Curso.

3. Tabla Inscripciones:

- Definición:

```
CREATE TABLE Inscripciones (
    ID_Inscripcion INT PRIMARY KEY,
    ID_Estudiante INT,
    ID_Curso INT,
    FechaInscripcion DATE,
    FOREIGN KEY (ID_Estudiante) REFERENCES
    Estudiantes(ID_Estudiante),
```

```
FOREIGN KEY (ID_Curso) REFERENCES Cursos(ID_Curso)
);
```

- **Explicación:** La tabla *Inscripciones* establece una relación entre estudiantes y cursos, mostrando qué estudiantes están inscritos en qué cursos. Las claves foráneas *ID_Estudiante* y *ID_Curso* aseguran que cada inscripción esté vinculada a un estudiante y un curso existente.

Conclusión

El modelo relacional proporciona una estructura sólida para organizar, almacenar y gestionar datos en bases de datos. A través de conceptos como relaciones, predicados, integridad de datos y claves, este modelo asegura que la información sea precisa, coherente y fácilmente accesible. Comprender estos conceptos es crucial para diseñar bases de datos efectivas y eficientes, y para poder realizar operaciones complejas sobre los datos almacenados.

Este desarrollo detallado debe proporcionar a los estudiantes una comprensión completa del modelo relacional, sus componentes, y cómo se implementan en bases de datos modernas.

Creación de Tablas y Definición de Claves en SQL Server

1. Sintaxis para Crear Tablas en SQL Server

El comando `CREATE TABLE` en SQL Server se utiliza para definir una nueva tabla en la base de datos. Al crear una tabla, es necesario especificar los nombres de las columnas, los tipos de datos que almacenarán, y cualquier restricción que deba aplicarse, como las claves primarias y foráneas.

Sintaxis Básica:

```
CREATE TABLE nombre_tabla (
    nombre_columna tipo_dato [restricciones],
    nombre_columna tipo_dato [restricciones],
    ...
);
```

Tipos de Datos Comunes:

- `INT`: Número entero.
- `VARCHAR(n)`: Cadena de texto de longitud variable, donde `n` es la longitud máxima.
- `DATE`: Fecha.
- `DECIMAL(p, s)`: Número decimal, donde `p` es la precisión total y `s` es la escala (número de decimales).

Definición de Claves:

- **Clave Primaria**: Se define utilizando la palabra clave `PRIMARY KEY`. Una tabla puede tener solo una clave primaria, que asegura la unicidad de los registros.
- **Clave Foránea**: Se define utilizando la palabra clave `FOREIGN KEY` y se utiliza para establecer una relación entre dos tablas.

2. Ejemplo: Creación de Tablas Estudiantes y Cursos

Veamos cómo se crean dos tablas básicas, `Estudiantes` y `Cursos`, y cómo se establece una relación entre ellas mediante una clave foránea.

Tabla Estudiantes

```
CREATE TABLE Estudiantes (
    ID_Estudiante INT PRIMARY KEY,
    Nombre VARCHAR(50),
    Apellido VARCHAR(50),
    FechaNacimiento DATE
);
```

Explicación:

- ID_Estudiante es la clave primaria (PRIMARY KEY), que asegura que cada estudiante tenga un identificador único.
- Nombre, Apellido, y FechaNacimiento son atributos que almacenan la información relevante de cada estudiante.

Tabla Cursos

```
CREATE TABLE Cursos (
    ID_Curso INT PRIMARY KEY,
    NombreCurso VARCHAR(100)
);
```

Explicación:

- ID_Curso es la clave primaria para la tabla Cursos, asegurando que cada curso tenga un identificador único.
- NombreCurso almacena el nombre del curso.

3. Ejemplo: Creación de Tablas Clientes, Pedidos, Productos y Proveedores

Ahora crearemos una serie de tablas más complejas para gestionar un sistema de pedidos, que incluye las tablas Clientes, Pedidos, Productos, y Proveedores.

Tabla Clientes

```
CREATE TABLE Clientes (
    ID_Cliente INT PRIMARY KEY,
    NombreCliente VARCHAR(100),
    Direccion VARCHAR(150),
    Telefono VARCHAR(15)
);
```

Explicación:

- ID_Cliente es la clave primaria.
- NombreCliente, Direccion, y Telefono son los atributos que almacenan la información de contacto de cada cliente.

Tabla Proveedores

```
CREATE TABLE Proveedores (
    ID_Proveedor INT PRIMARY KEY,
    NombreProveedor VARCHAR(100),
    DireccionProveedor VARCHAR(150),
    TelefonoProveedor VARCHAR(15)
);
```

Explicación:

- ID_Proveedor es la clave primaria.

- NombreProveedor, DireccionProveedor, y TelefonoProveedor son los atributos que almacenan la información de los proveedores.

Tabla Productos

```
CREATE TABLE Productos (
    ID_Producto INT PRIMARY KEY,
    NombreProducto VARCHAR(100),
    Precio DECIMAL(10, 2),
    Stock INT,
    ID_Proveedor INT,
    FOREIGN KEY (ID_Proveedor) REFERENCES Proveedores(ID_Proveedor)
);
```

Explicación:

- ID_Producto es la clave primaria.
- NombreProducto es el nombre del producto.
- Precio es el precio del producto, almacenado con dos decimales.
- Stock es la cantidad disponible en inventario.
- ID_Proveedor es una clave foránea que establece la relación entre Productos y Proveedores, asegurando que cada producto esté asociado con un proveedor existente.

Tabla Pedidos

```
CREATE TABLE Pedidos (
    ID_Pedido INT PRIMARY KEY,
    FechaPedido DATE,
    ID_Cliente INT,
    ID_Producto INT,
    Cantidad INT,
    PUnitario DECIMAL (10, 2),
    FOREIGN KEY (ID_Cliente) REFERENCES Clientes(ID_Cliente),
    FOREIGN KEY (ID_Producto) REFERENCES Productos(ID_Producto)
);
```

Explicación:

- ID_Pedido es la clave primaria para la tabla Pedidos.
- FechaPedido almacena la fecha en que se realizó el pedido.
- ID_Cliente es una clave foránea que vincula el pedido con un cliente específico.
- ID_Producto es una clave foránea que vincula el pedido con un producto específico.
- Cantidad almacena la cantidad de productos solicitados en el pedido.

4. Resumen y Práctica

Este paso a paso cubre la creación de tablas con claves primarias y foráneas, estableciendo relaciones entre ellas. Los estudiantes deben practicar creando estas tablas

en SQL Server, ejecutando los comandos SQL proporcionados, y verificando que las relaciones entre las tablas funcionen correctamente.

Tarea Práctica:

- Crear las tablas `Clientes`, `Pedidos`, `Productos`, y `Proveedores` en SQL Server.
- Insertar algunos datos de prueba en cada tabla.
- Ejecutar consultas para verificar que las relaciones entre las tablas se han establecido correctamente. Por ejemplo, obtener todos los pedidos de un cliente específico o listar todos los productos suministrados por un proveedor determinado.

Consultas Básicas: SELECT, FROM y WHERE

Ahora que hemos poblado las tablas con datos, podemos proceder a realizar consultas básicas para recuperar y filtrar información.

Conceptos Generales

SELECT

- **Definición:** El comando `SELECT` se utiliza para especificar las columnas que deseas recuperar de una o más tablas en una base de datos.
- **Concepto:**
 - Es el comando principal en SQL para extraer datos.
 - Puedes seleccionar una o varias columnas especificando sus nombres después de la palabra clave `SELECT`.
 - También puedes usar `SELECT *` para seleccionar todas las columnas de la(s) tabla(s).
 - Además, permite realizar operaciones sobre los datos, como concatenación de columnas, funciones de agregación (por ejemplo, `SUM`, `AVG`), y más.

Ejemplo:

```
SELECT nombre, edad FROM empleados;
```

Este comando selecciona las columnas `nombre` y `edad` de la tabla `empleados`.

2. FROM

- **Definición:** El comando `FROM` se utiliza para especificar la tabla o tablas de las que se van a recuperar los datos en una consulta SQL.
- **Concepto:**

- Después de `FROM`, se indica el nombre de la tabla de la cual se van a extraer los datos.
- Si se necesitan datos de varias tablas, se pueden unir estas tablas mediante la cláusula `JOIN`.
- También puedes usar alias para tablas para simplificar la consulta.

Ejemplo:

```
SELECT * FROM empleados;
```

Aquí, `empleados` es la tabla de la que se recuperan todos los datos.

3. WHERE

- **Definición:** El comando `WHERE` se utiliza para especificar condiciones que los datos deben cumplir para ser seleccionados.
- **Concepto:**
 - Se emplea para filtrar los resultados de una consulta SQL.
 - Solo las filas que cumplen las condiciones especificadas en `WHERE` serán incluidas en el resultado.
 - Puedes combinar múltiples condiciones usando operadores lógicos como `AND`, `OR`, y `NOT`.
 - También permite el uso de operadores de comparación como `=`, `!=`, `<`, `>`, `<=`, `>=`, entre otros.

Ejemplo:

```
SELECT nombre FROM empleados WHERE edad > 30;
```

Este comando selecciona los nombres de los empleados cuya edad es mayor que 30.

En conjunto, estos comandos permiten realizar consultas en bases de datos SQL para recuperar datos específicos basados en ciertos criterios.

Sintaxis Básica de SELECT

```
SELECT columnas
FROM tabla
WHERE condición;
```

Ejemplos de Consultas Básicas

1. Consultar todos los clientes:

```
SELECT * FROM Clientes;
```

Explicación:

- `SELECT *` indica que queremos seleccionar todas las columnas.
- `FROM Clientes` especifica la tabla de la que queremos extraer los datos.

2. Consultar todos los productos con su nombre y precio:

```
SELECT NombreProducto, Precio FROM Productos;
```

Explicación:

- Esta consulta selecciona solo las columnas `NombreProducto` y `Precio` de la tabla `Productos`.

3. Filtrar pedidos realizados por un cliente específico (por ejemplo, cliente con ID 1):

```
SELECT * FROM Pedidos
WHERE ID_Cliente = 1;
```

Explicación:

- `WHERE ID_Cliente = 1` filtra los resultados para mostrar solo los pedidos realizados por el cliente con `ID_Cliente` igual a 1.

4. Consultar el total de productos pedidos por cada cliente:

```
SELECT ID_Cliente, SUM(Cantidad) AS TotalProductos
FROM Pedidos
GROUP BY ID_Cliente;
```

Explicación:

- Esta consulta suma la cantidad de productos pedidos (`SUM(Cantidad)`) para cada cliente (`GROUP BY ID_Cliente`).

5. Consultar los pedidos realizados en una fecha específica (por ejemplo, '2024-08-02'):

```
SELECT * FROM Pedidos
WHERE FechaPedido = '2024-08-02';
```

Explicación:

- Filtra los pedidos para mostrar solo aquellos realizados en la fecha especificada.

6. Consultar el nombre del cliente, nombre del producto y cantidad para cada pedido:

```
SELECT Clientes.NombreCliente, Productos.NombreProducto,
Pedidos.Cantidad
FROM Pedidos
JOIN Clientes ON Pedidos.ID_Cliente = Clientes.ID_Cliente
JOIN Productos ON Pedidos.ID_Producto = Productos.ID_Producto;
```

Explicación:

- Esta consulta combina las tablas `Pedidos`, `Cientes`, y `Productos` para mostrar el nombre del cliente, el nombre del producto, y la cantidad de cada pedido.

DISEÑO CONCEPTUAL

Cuando hablamos de **diseño conceptual** en bases de datos (y en general en ingeniería de sistemas) nos referimos a la **etapa inicial del diseño**, donde se construye un **modelo abstracto de los datos y sus relaciones**, sin preocuparnos todavía por aspectos técnicos del sistema de gestión que usaremos (SQL Server, Oracle, etc.).

Características principales del diseño conceptual:

1. **Abstracción:**
Se trabaja a un nivel alto, independiente de la implementación. No pensamos en tablas ni tipos de datos, sino en **entidades, atributos y relaciones**.
2. **Representación:**
Se suele usar un **diagrama entidad-relación (ER)** o algún modelo conceptual similar, que permite visualizar de forma clara cómo se relacionan los elementos de información del negocio.
3. **Orientado al negocio:**
Refleja cómo la organización ve y utiliza la información. Por ejemplo: "Clientes realizan Pedidos", "Productos pertenecen a Categorías", "Empleados trabajan en Sucursales".
4. **Independencia de la tecnología:**
No importa si después la base se implementa en SQL Server, Oracle, MongoDB, etc. El diseño conceptual describe **qué datos** deben almacenarse, no **cómo**.
5. **Objetivo:**
Asegurar que todos los requerimientos de información del usuario estén incluidos y correctamente representados, sirviendo como **punto de encuentro entre los analistas y los técnicos**.

Ejemplo simple:

- Entidad: **Cliente** (con atributos: DNI, Nombre, Dirección)
- Entidad: **Pedido** (con atributos: NºPedido, Fecha, Importe)
- Relación: **Cliente realiza Pedido** (1:N)

En este punto **no hablamos de claves primarias, índices ni tipos de datos**, solo del **modelo lógico del negocio**.

Un **Diagrama Entidad-Relación (DER)** es la **herramienta gráfica principal** que usamos en el **diseño conceptual** de bases de datos.

Sirve para **representar de manera visual** las entidades del negocio, sus atributos y las relaciones entre ellas.

Elementos básicos del DER:

1. **Entidades (rectángulos):**

- Representan objetos del mundo real sobre los que se quiere guardar información.
- Ejemplo: *Cliente, Producto, Pedido*.

2. **Atributos (óvalos):**

- Son las propiedades o características de las entidades.
- Ejemplo: *Cliente* → DNI, Nombre, Dirección.
- El atributo que identifica de manera única se llama **Clave primaria (PK)**.

3. **Relaciones (rombos o líneas):**

- Expresan cómo se vinculan las entidades.
- Ejemplo: *Cliente realiza Pedido*.

4. **Cardinalidad:**

- Define cuántas ocurrencias de una entidad pueden asociarse con otra.
- Tipos más comunes:
 - **1:1** → un cliente tiene un único pasaporte.
 - **1:N** → un cliente puede hacer muchos pedidos, pero cada pedido pertenece a un solo cliente.
 - **N:M** → un alumno cursa muchas materias y una materia tiene muchos alumnos.

5. **Restricciones:**

- Se suelen anotar en el diagrama para indicar reglas adicionales, como obligatoriedad (mínimo 1) o posibilidad de ser opcional (0).

Ejemplo simple de DER (descripción textual):

- Entidad: **Cliente** (ID_Cliente, Nombre, Email)
- Entidad: **Pedido** (ID_Pedido, Fecha, Total)
- Relación: **Cliente realiza Pedido** (1:N)

Representación:

Cliente (1) – Realiza -- (N) Pedido

Esto significa que **un cliente puede hacer muchos pedidos**, pero cada pedido pertenece a un **único cliente**.

Cuando observamos y operamos con una base de datos, realmente no estamos observando todo el engranaje de la misma. Al usuario no le interesa toda la estructura interna del diseño de la base para poder procesar, editar y consultar los datos.

Esto es posible gracias al Modelado de Dato, que es la forma en la que se puede ocultar toda esa estructura que no necesitamos ver.

Tenemos por un lado los modelos de datos de alto nivel, o modelos conceptuales, frente a los modelos de bajo nivel o modelos físicos.

Los modelos físicos están orientados al programador de la base de datos y no hacia el usuario final. Los modelos conceptuales sí se acercan más a la visión de cómo el usuario percibe cómo se guardan los datos. En los modelos conceptuales podemos comprobar una serie de elementos en su uso:

- **Entidad.** Representa un objeto real. En nuestro ejemplo de la empresa de seguros, la entidad puede ser un cliente. Podemos bajar el registro y también podría ser el coste que supuso su último reporte de siniestro. Es el objeto de estudio.
- **Atributo.** Es una característica de la entidad. Si hemos determinado que la entidad es un cliente, el atributo podría ser la fecha de alta en la que contrató el seguro. Por tanto la diferencia con la entidad es clara. La entidad es la parte que se quiere analizar y el atributo es una de sus características fijas.
- **Relación.** Es el tipo de relación entre varias entidades. Y se puede dividir en varios casos: relación uno a uno, uno a muchos, muchos a muchos. Se indicará el grado de relación y las claves.

Este esquema de datos entidad-relación (ER) se implementa posteriormente con un gestor de sistema SGBD.

Determinamos la relación entre las diferentes entidades.

Por ejemplo, las entidades que podemos obtener de nuestro ejemplo de empresa de seguros son los objetos físicos.

Clientes, siniestros que reportan, peritos que se asocian, el coste de cada reporte.

Las relaciones se muestran en formato de rombo.



Gráfico 1. Ejemplo de relación.

En este gráfico hemos relacionado dos entidades, cliente y prima de seguro, y la relación que hay entre ellas: los clientes pagan la prima de seguro.

Las relaciones muestran cómo se relacionan las entidades. Por ejemplo, podríamos relacionar dos entidades respecto a un día de la semana, o si el reporte de un siniestro se realiza por teléfono, por mail o personándose en la oficina.

CLIENTES	SEGURO CONTRATADO	PRIMA	REPORTE
Luis	Hogar	Mensual	Teléfono
Carmen	Hogar	Anual	Teléfono
Rosa	Salud	Trimestral	Mail
Clara	Vehículo	Anual	Teléfono
Carlos	Salud	Mensual	Oficina

De esta forma se puede crear la base de datos, donde la relación entre las diferentes entidades es la forma de reportar el siniestro. Algo que se podría sustituir por un código que solo conozcan aquellas personas que necesiten esa información directa y que quede oculto al resto de usuarios de la base de datos. Es una forma de dar privacidad a ciertos datos y de poder establecer una seguridad en el uso de la base de datos.

CLIENTES	SEGURO CONTRATADO	PRIMA	REPORTE
Luis	Hogar	Mensual	ID001
Carmen	Hogar	Anual	ID001
Rosa	Salud	Trimestral	ID002
Clara	Vehículo	Anual	ID001
Carlos	Salud	Mensual	ID003

Existen diferentes tipos de relaciones en cuanto a su cardinalidad.

- Uno a uno. Una entidad se relaciona únicamente con otra. Es el ejemplo que hemos puesto anteriormente: cada cliente se relaciona con su prima de seguro. Se representa de la siguiente forma:



- Gráfico 2. Ejemplo relación uno a uno.
- Uno a muchos. Caso en el que una entidad puede estar relacionada con varios registros de otra entidad, pero en esta entidad existe solo una vez. Como

tenemos varios comerciales captando clientes, reflejamos en nuestra base de datos qué cliente ha sido captado por cada comercial. Así vemos cómo varios comerciales que trabajan para la empresa salen a contratar -lo identificamos con (0,n)-, pero cada cliente es captado solo por un comercial, por lo tanto la relación del cliente es uno a uno (1,1).



Gráfico 3. Ejemplo relación uno a muchos.

- Muchos a muchos. Aquí vemos cómo varias entidades pueden relacionarse con varios registros de otras entidades.



Gráfico 4. Ejemplo relación muchos a muchos.

- El (0,n) y (1,1) son la **notación de cardinalidad (mínimo, máximo)**.
 - (0,n) significa que puede no haber relación o haber muchas.
 - (1,1) significa que **es obligatorio** y además **único**.
- Este nivel de detalle (con mínimo y máximo) no siempre se ve en los DER más simples, pero es muy útil porque expresa **restricciones adicionales** del negocio.
- En el modelo lógico posterior (tablas SQL Server), esto se traduciría en:
 - Una tabla **Comerciales**
 - Una tabla **Cientes** con una **clave foránea** hacia Comerciales (porque cada cliente pertenece a un comercial).

En este ejemplo vemos claramente cómo una entidad con varios registros, como son los peritos (cada perito es un registro), puede acudir a valorar diferentes siniestros (por ejemplo, incendios, inundaciones, roturas cristal, robos, etc.).

La forma de identificar cada tipo de relación es en base a unos indicadores numéricos, (1,1) o (0,n). Con esto, lo que queremos decir es la cantidad de registros y relaciones que se producen, cantidad mínima (el primer valor de la coordenada) y la cantidad máxima (segundo valor de la coordenada). Cuando la relación es uno a uno, el indicador numérico es (1,1), pero cuando es de varios, el primer valor representa el número mínimo de registros de una relación, por norma es 0, y el segundo es el máximo, si es ilimitado ponemos 'n'.

En el ejemplo de los comerciales, si la empresa cuenta con tres comerciales, se podría poner (0,3), pero eso nos limitaría en el caso de que en un futuro se pudiese contratar algún comercial más. Mejor sería dejarlo abierto y poner 'n'.

Existe otro elemento usado por los modelos conceptuales, las claves. Las claves son los atributos de una entidad. Es el hecho diferencial entre registros. Podría ser la fecha de nacimiento de cada cliente, aunque se podría correr el riesgo de encontrarnos con dos clientes con la misma fecha de nacimiento y entonces no sería un hecho diferencial.

Así que aunque creamos que tenemos un atributo diferencial, debemos asegurarnos de ello. Si contamos los clientes como unidad familiar, entonces una clave podría ser la dirección de su domicilio. Necesitamos un atributo que pertenezca unívocamente a la entidad.

Restricciones

Cuando hablamos de **restricciones** en bases de datos (particularmente en el **diseño conceptual y lógico**), nos referimos a las **reglas que limitan o condicionan los datos** que pueden almacenarse y las relaciones entre ellos.

Son esenciales porque aseguran que la base de datos mantenga la **integridad** y refleje correctamente la realidad del negocio.

Tipos de restricciones principales:

1. **Restricciones de dominio (atributos):**
 - Definen los valores válidos que puede tomar un atributo.
 - Ejemplo: la edad de un cliente debe ser un número mayor o igual a 18.
 - En SQL luego se expresan como CHECK, tipos de datos, o reglas de validación.
2. **Restricciones de clave (unicidad):**
 - Identifican de manera única a cada registro.
 - Ejemplo: en la entidad *Cliente*, el DNI o ID_Cliente debe ser único.
 - Se implementan con PRIMARY KEY o UNIQUE.
3. **Restricciones de integridad referencial (entre tablas/entidades):**
 - Garantizan coherencia entre datos relacionados.
 - Ejemplo: un *Pedido* no puede existir sin un *Cliente* asociado.
 - Se implementan con FOREIGN KEY.
4. **Restricciones de cardinalidad (en el modelo ER):**
 - Indican cuántas ocurrencias de una entidad pueden estar asociadas a otra.
 - Ejemplo: un cliente puede hacer **muchos** pedidos (1:N), pero cada pedido pertenece a **un solo** cliente.

5. Restricciones de negocio (reglas específicas):

- Dependen del dominio o la organización.
- Ejemplo: “un empleado no puede tener más de un jefe directo” o “no se pueden vender productos sin stock disponible”.
- A veces no se pueden expresar solo en el modelo conceptual y se aplican con procedimientos o triggers en SQL.




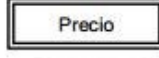








Tips para representar un DER

1. Entidades

- Se dibujan como **rectángulos**.
- Usá **sustantivos en singular** (*Cliente, Producto, Empleado*).
- La clave primaria suele **subrayarse** en los atributos (*ID_Cliente*).

2. Atributos

- Se representan con **óvalos** unidos a la entidad.
- Clasificación:
 - **Clave primaria:** subrayada.
 - **Multivaluado:** óvalo doble (ej: un cliente puede tener varios teléfonos).
 - **Derivado:** óvalo con línea discontinua (ej: Edad, calculada a partir de FechaNacimiento).
 - **Compuesto:** se puede descomponer en subatributos (ej: Dirección → Calle, Número, Ciudad). Óvalo principal desde donde salen otros óvalos más pequeños representando sus componentes

Símbolo	Significado	Ejemplo
	Entidad Fuerte	
	Entidad Débil	
	Atributo	
	Relación	
	Atributo multivaluado	
	Atributo Derivado	

3. Relaciones

- Se dibujan como **rombos** conectando entidades.
- Usá **verbos** en el nombre (*Realiza, Contiene, Pertenece*).
- Podés poner **atributos de la relación** si hace falta (ej: en la relación *Alumno cursa Materia* → atributo *Nota*).

4. Cardinalidad

- Mostrá **cuántas veces** una entidad puede participar en la relación:
 - Notación simple: (1:1, 1:N, N:M).
 - Notación min-máx: (0,1), (0,N), (1,1), (1,N).
- Tip: siempre aclará la **obligatoriedad**:
 - (0,N) → opcional y múltiple.
 - (1,N) → obligatorio y múltiple.

5. Relaciones N:M

- Muy comunes, pero en la implementación **se transforman en una tabla intermedia**.
- Ejemplo: *Alumno (N) – Cursa – (M) Materia*.
 - Se resuelve con una tabla *AlumnoMateria* con claves foráneas.

6. Claridad visual

- Usá **nombres consistentes** (Cliente, Pedido, Producto).
- Evitá líneas que se crucen, moviendo las entidades en el diagrama.
- Diferenciá entidades fuertes (rectángulo normal) y débiles (doble rectángulo).

7. Niveles de detalle

- **Simplificado (didáctico)**: Cliente (1) – Realiza – (N) Pedido.
- **Formal (académico)**: usando min-máx (0,N) y (1,1).
- Podés mostrar ambos a los alumnos: uno para **comprensión rápida**, otro para **precisión técnica**.

8. Restricciones adicionales

- A veces se agregan en notas o con símbolos:
 - Exclusividad (*un empleado puede ser gerente o supervisor, pero no ambos*).
 - Totalidad (*todo empleado debe estar asignado a un departamento*).

Entidad Fuerte

- Es la entidad que **existe por sí misma** y no depende de otra para estar definida.
- Tiene una **clave primaria propia**, que la identifica de manera única.
- Ejemplos:
 - **Cliente** (ID_Cliente, Nombre, Email)
 - **Producto** (ID_Producto, Nombre, Precio)

Se representa en el DER con un **rectángulo simple**.

Entidad Débil

- Es la entidad que **no puede existir sin una entidad fuerte** que la identifique.
- **No tiene clave primaria propia completa**, sino que depende de la clave de otra entidad (se dice que tiene **dependencia de identificación**).
- Suele tener un atributo parcial (**discriminador**) que, junto con la clave de la entidad fuerte, forma la clave primaria.
- Ejemplos:
 - **DetallePedido** → depende de *Pedido*. (No tiene sentido sin él).
 - **Cuota** → depende de *Préstamo*. (Se identifica por N° de Cuota + ID_Préstamo).

Se representa en el DER con un **rectángulo doble**.

La relación que une la entidad débil con la fuerte se representa con un **rombo doble**.

DISEÑO CONCEPTUAL DEL CASO

Planteamiento del problema y diagnóstico

La empresa **Book Online Rental** gestiona actualmente un sistema de alquiler de libros digitales que involucra múltiples entidades relacionadas: libros, autores, estudiantes y préstamos. Sin embargo, el modelo actual presenta importantes limitaciones, como duplicidad de datos, baja trazabilidad de préstamos, dificultad para relacionar autores con libros y ausencia de control en la disponibilidad del material. Estas deficiencias se deben, en parte, a la falta de un modelo de datos estructurado que represente adecuadamente la lógica del negocio.

El diagnóstico realizado permite observar que el sistema requiere una representación que contemple relaciones de tipo uno a muchos y muchos a muchos, con validaciones estructurales y semánticas que aseguren la consistencia de las operaciones. Para resolver este escenario, se propone el diseño de un modelo entidad-relación que contemple no solo las entidades y relaciones, sino también las restricciones necesarias para que el modelo sea natural, funcional y escalable.

Solución

A continuación se presenta el diseño lógico del modelo, donde tomamos la sugerencia del caso y ampliamos la concepción original de forma tal que pudiera representar con más claridad la problemática planteada. En primer lugar, se decide tener una apertura de Libros por un lado y otra de Ejemplares que nos permita administrar individualmente las cantidades de un mismo título o libro prestado a los estudiantes. Basado en las siguientes entidades y relaciones:

Entidades

- **Libro:** ISBN (clave primaria), Título, Editorial, Año de publicación, Género
- **Autor:** ID_Autor (clave primaria), Nombre, Nacionalidad

- **Escribe:** Entidad Asociativa: ISBN (clave foránea), ID_Autor (clave foránea)
- **Ejemplar:** ID_Ejemplar (clave primaria), ISBN (clave foránea), Estado, Disponible (S/N, Default = N), Formato (PDF, ePub, MOBI, eBook, Audiolibro, Impreso), Observaciones
- **Estudiante:** ID_Estudiante (clave primaria), Nombre, Apellido, Número de estudiante (único), Carrera
- **Préstamo:** ID_Prestamo (clave primaria), Fecha de préstamo, Fecha de devolución prevista, Fecha devolución real, ID_Estudiante (clave foránea), ID_Ejemplar (clave foránea)

Relaciones

- Autor --- **Escribe** --- Libro (muchos a muchos): Un libro puede tener uno o más autores. Un autor puede haber escrito uno o más libros. Esta relación se modela mediante una **entidad asociativa “Escribe”** con claves foráneas hacia la entidad *Libro* y *Autor*
- Libro --- **Inventariado en** --- Ejemplar (uno a muchos): un libro puede tener varios ejemplares en el inventario del sistema pero un ejemplar representa solo un libro.
- Ejemplar --- **Está_en** --- Préstamo (uno a muchos): Un ejemplar puede estar en varios préstamos pero cada préstamo gestiona un único ejemplar.
- Estudiante – **Solicita** – Préstamo (uno a muchos): Un estudiante puede tener varios préstamos y cada préstamo se asocia a un único estudiante.

Restricciones y reglas de negocio

- Integridad referencial: Cada préstamo debe estar vinculado a un ejemplar y un estudiante válidos.
- Restricción de unicidad: El número de estudiante debe ser único.
- Consistencia de fechas: La fecha de devolución prevista debe ser posterior a la fecha de préstamo.

- Cardinalidad máxima por período (regla lógica, a implementar en la aplicación): un mismo libro no puede ser prestado a dos estudiantes en el mismo intervalo de fechas
- Un ejemplar puede estar disponible o no disponible, ingresando al inventario con el valor “N” por defecto para evitar que se muestre disponible un título que aún no está clasificado. El ejemplar debe estar disponible = S para que pueda ser prestado.
- El formato del libro puede ser, *PDF* (Formato de maquetación fija), *ePub* (Formato abierto y flexible), *eBook* (denominación general para libros digitales incluyendo formatos varios como MOBI o AZW de Amazon, IBA de Apple, HTML, RTF, y otros electrónicos), *Audiolibro* (incluye los formatos MP3, M4B, AAC, WMA entre otros), *Impreso* (para los libros en papel con encuadernación)

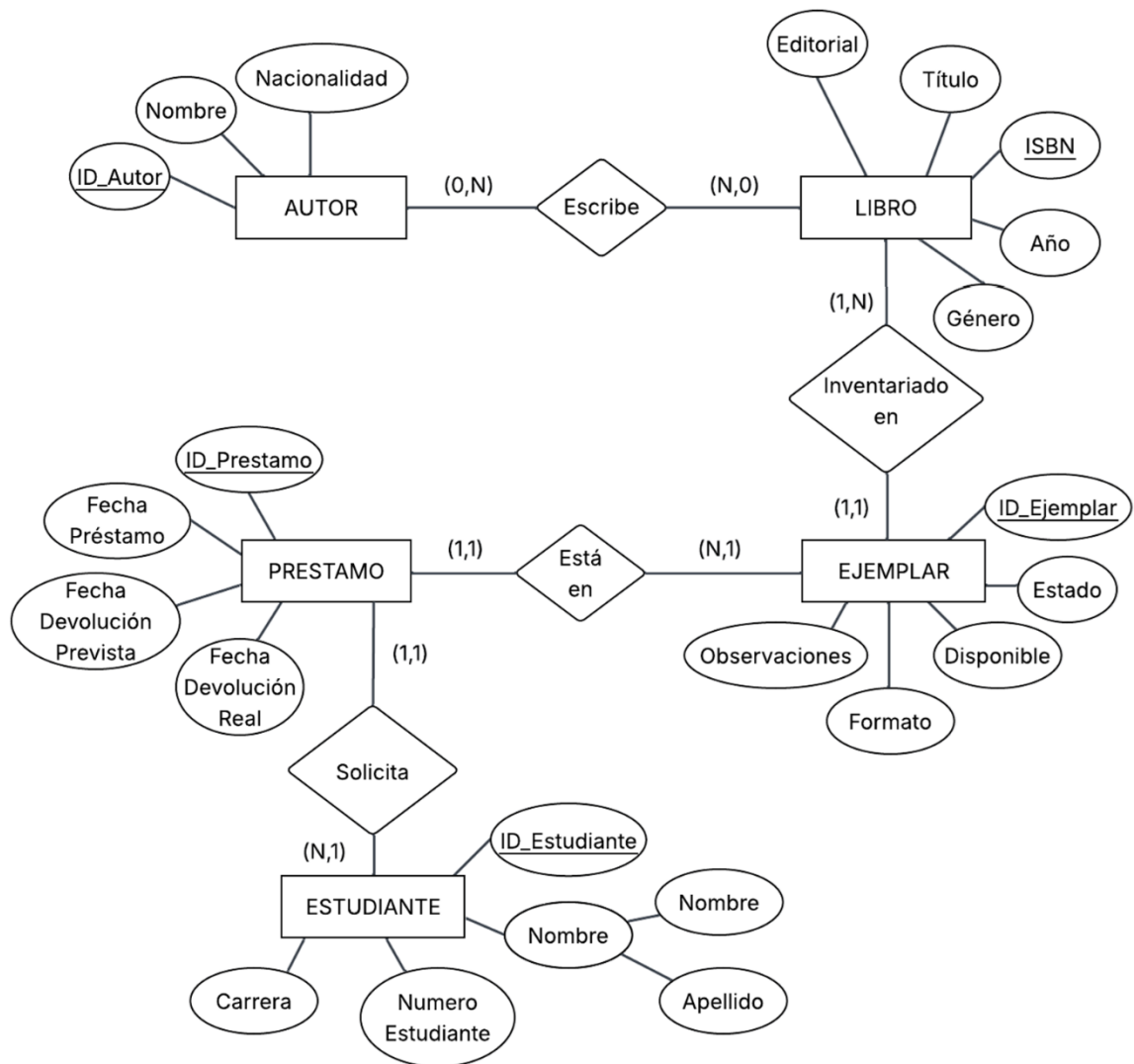
Las restricciones de integridad referencial y unicidad son implementables directamente en el modelo relacional. Otras reglas como el control de fechas o la validación del estado de disponibilidad constituyen reglas de negocio, que si bien no son impuestas por el modelo lógico, deben contemplarse en la lógica de aplicación o mediante mecanismos adicionales (triggers, procedimientos almacenados, validaciones de front-end).

Diagrama Entidad-Relación

El diagrama Entidad-Relación (ERD) es una herramienta fundamental en la fase de diseño conceptual de bases de datos, ya que permite representar gráficamente los elementos clave de un sistema de información, sus relaciones, y las restricciones semánticas del negocio. Este tipo de diagrama facilita una visión abstracta de los datos, independiente de cualquier implementación física, y constituye la base sobre la cual se derivará posteriormente el modelo lógico-relacional.

Según Elmasri y Navathe (2007), ERD permite describir de forma clara y comprensible las entidades, atributos, relaciones y cardinalidades, lo que lo convierte en una técnica indispensable para lograr una base de datos bien estructurada, coherente con los requerimientos del negocio y capaz de evolucionar con las necesidades del sistema. C. J. Date (2001) también resalta la importancia de utilizar este modelo en la etapa inicial del diseño, ya que permite detectar posibles redundancias, anomalías o ambigüedades antes de pasar a fases más técnicas como la normalización o la implementación física.

En el presente trabajo, se adopta el ERD para representar el universo de datos de **Book Online Rental**, haciendo uso de las convenciones propias del modelo conceptual (estilo Chen), tal como se propone en los apuntes de la asignatura Bases de Datos de ENEB (2024). Este enfoque permite identificar claramente las entidades relevantes del negocio —como Libro, Autor, Estudiante, Ejemplar y Préstamo—, así como las relaciones entre ellas, sus restricciones de cardinalidad, y los atributos principales. Además, se contemplan reglas de negocio clave, que permitirán una implementación futura más robusta, segura y escalable.



Nota: Se ha optado por establecer una cardinalidad mínima de 0 en la relación entre Autor y Libro a través de la entidad asociativa Escribe, con el fin de permitir mayor flexibilidad operativa. Esto contempla escenarios en los que se cargan libros aún no asignados a un autor, o autores registrados que todavía no tienen obras vinculadas en el sistema. Esta decisión busca reflejar un enfoque más realista en el uso progresivo de los datos, sin comprometer la integridad del modelo.