UD03. Los esquemas relacionales y su transformación.

Teo Rojas

Noviembre 2024

Sinopsis

Sinopsis de la unidad 03

Índice

1. Introducción a los esquemas relacionales

El **esquema relacional** es la representación formal de la estructura de una base de datos en el modelo relacional. Es el resultado de transformar el modelo conceptual, generalmente expresado en un diagrama Entidad-Relación (E/R), en una estructura que pueda implementarse en un sistema de gestión de bases de datos relacional (SGBDR).

El esquema relacional define cómo se almacenan, organizan y manipulan los datos en la base de datos, estableciendo las tablas, los campos, las relaciones entre ellas y las restricciones de integridad necesarias para garantizar la consistencia y validez de los datos.

1.1. ¿Qué es un esquema relacional?

Un **esquema relacional** es la descripción de la estructura de una base de datos en términos de:

- **Tablas** (o relaciones): Representan las entidades o conceptos principales del modelo conceptual.
- **Columnas** (o atributos): Representan las propiedades o características de las tablas.
- **Relaciones** (o asociaciones): Definen las conexiones lógicas entre diferentes tablas.

Cada tabla en un esquema relacional tiene:

- Un nombre único que la identifica.
- Un conjunto de atributos, donde cada atributo tiene un nombre y un dominio (tipo de dato permitido).
- Una o más claves primarias, que identifican de forma única cada fila de la tabla.
- Opcionalmente, claves foráneas, que establecen relaciones con otras tablas.

En términos simples, un esquema relacional actúa como un **mapa estructurado de la base de datos** que define cómo se organizan los datos y cómo interactúan entre sí.

1.2. Componentes de un esquema relacional: tablas, campos y relaciones

Tablas

- Son la representación de las entidades principales del modelo conceptual.
- Contienen filas (o tuplas), que representan instancias de la entidad.
- Ejemplo: Una tabla llamada `Empleado` podría tener filas que representen a cada empleado en la empresa.

Campos (o columnas)

- Representan las características o atributos de la entidad.
- Cada columna tiene un nombre único dentro de la tabla y pertenece a un dominio específico.
- Ejemplo: En la tabla `Empleado`, las columnas podrían ser `ID_Empleado`, `Nombre`, `Apellido`, `Departamento`.

Relaciones

- Establecen las conexiones lógicas entre diferentes tablas.
- Se representan mediante claves foráneas, que actúan como un enlace entre las tablas.
- Ejemplo: Una relación entre las tablas `Empleado` y `Departamento` podría indicar en qué departamento trabaja cada empleado.

Ejemplo práctico:

- **Modelo E/R:** Una entidad llamada `Empleado` tiene atributos como `ID_Empleado`, `Nombre`, `Apellido`, `Cargo`, `Sueldo`, `Nacimiento`, `Sexo`, `Estado Civil`.
- **Esquema relacional:** Se crea una tabla llamada `Empleado` con columnas `ID_Empleado`, `Nombre`, `Apellido`, `Cargo`, `Sueldo`, `Nacimiento`, `Sexo`, `Estado Civil`.

Tabla: Empleado

ID_Empleado	Nombre	Apellido	Cargo	Sueldo	Nacimiento	Sexo	Estado Civil
001	María	López	Gerente	45,000€	1985-03-12	F	Casada
002	Juan	Fernández	Analista	38,000€	1990-07-24	M	Soltero
003	Sofía	García	Vendedora	32,000€	1995-05-16	F	Soltera

Descripción:

1. **ID_Empleado:** Identificador único del empleado.
2. **Nombre:** Nombre del empleado.
3. **Apellido:** Apellido del empleado.
4. **Cargo:** Posición o rol del empleado en la empresa.
5. **Sueldo:** Salario anual del empleado.
6. **Nacimiento:** Fecha de nacimiento del empleado.
7. **Sexo:** Género del empleado (F para Femenino, M para Masculino).
8. **Estado Civil:** Estado civil actual del empleado.

En el **Modelo Relacional**, una **RELACIÓN** se refiere únicamente a la definición de la estructura de una tabla. Esto incluye su **nombre** y la lista de **atributos** que la forman. Una forma de representar esta definición podría ser la siguiente:

	Estado Civil

Para este caso, se obrendrá el siguiente esquema relacional:

- **Empleado** *(ID_Empleado, Nombre, Apellido, Cargo, Sueldo, Nacimiento, Sexo, Estado Civil)*

Para identificar de manera única un registro dentro de una tabla, se utiliza la **clave primaria** o **clave principal**.

Es posible que en una relación existan varias combinaciones de atributos que permitan distinguir de forma unívoca una fila. Estas combinaciones se denominan **claves candidatas**. Sin embargo, de todas las claves candidatas, se seleccionará una para que funcione como **clave primaria**.

Un punto importante es que los atributos que forman parte de la clave primaria no pueden contener valores nulos, lo que garantiza la unicidad y la consistencia de los datos en la tabla.

2. Elementos y Propiedades del Modelo Relacional

2.1. Relación (Tabla)

En el modelo relacional, las **relaciones** representan las entidades sobre las que se desea almacenar información en la base de datos. Una relación está compuesta por:

- **Filas**: También llamadas **registros** o **tuplas**, representan cada ocurrencia de la entidad.
- **Columnas**: Conocidas como **atributos** o **campos**, corresponden a las propiedades de la entidad.

Siendo estrictos, una relación incluye únicamente los atributos (columnas) y no las tuplas.

2.2. Propiedades de las Relaciones

Las relaciones tienen las siguientes características clave:

1. Cada relación tiene un **nombre único**, distinto del nombre de cualquier otra relación en la base de datos.
2. En una relación, no puede haber dos atributos con el mismo nombre.
3. El **orden de los atributos** no importa; no están ordenados.
4. Las tuplas son **únicas**; no puede haber tuplas duplicadas. (Al menos deben diferenciarse por la clave principal).
5. El **orden de las tuplas** tampoco importa; no están ordenadas.

2.3. Tipos de Claves

- **Clave Candidata**: Es un atributo (o conjunto de atributos) que identifica de manera única una tupla. Cualquiera de estas claves podría ser elegida como clave principal.
- **Clave Principal**: Es la clave candidata seleccionada para identificar las tuplas de manera única.
- **Clave Alternativa**: Son las claves candidatas que no fueron seleccionadas como clave principal.

2.4. Reglas de Integridad

- Una **clave principal** no puede contener valores nulos, lo que garantiza la **integridad de la entidad**.

- El **dominio de un atributo** define el conjunto de valores que dicho atributo puede asumir.

2.5. Clave Externa o Foránea

La **clave externa** (también llamada clave ajena) es un atributo o conjunto de atributos que forman la clave principal de otra relación. Cumple las siguientes condiciones:

- Los valores en la clave externa deben coincidir con valores existentes en la clave principal correspondiente en otra tabla.
- Esto asegura la **integridad referencial**, es decir, que los datos relacionados entre tablas sean válidos y consistentes.

3. Transformación del modelo E/R al modelo relacional

La transformación del modelo Entidad-Relación (E/R) al modelo relacional es un proceso fundamental para implementar una base de datos diseñada conceptualmente en un sistema de gestión de bases de datos. Este proceso consiste en convertir las entidades, atributos, relaciones y restricciones definidas en el modelo E/R en tablas, columnas, claves y reglas que representen correctamente la información en el modelo relacional.

3.1. Reglas básicas para transformar entidades

El proceso general de transformación se lleva a cabo siguiendo estos pasos principales:

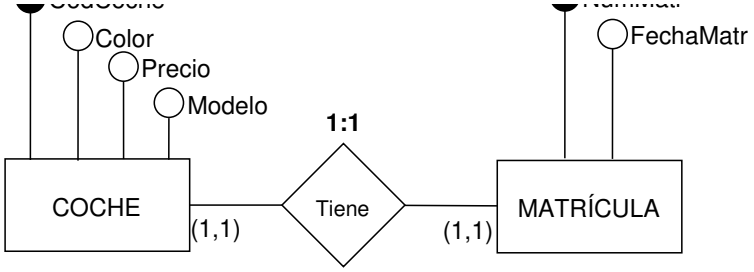
1. **Identificar las entidades y convertirlas en tablas:** Cada entidad del modelo E/R se transforma en una tabla en el modelo relacional.
2. **Definir los atributos:** Los atributos de cada entidad se convierten en columnas de la tabla correspondiente.
3. **Establecer claves primarias:** Se selecciona un atributo o combinación de atributos como clave primaria para cada tabla.
4. **Transformar las relaciones:** Se crean nuevas tablas o columnas para representar las relaciones entre entidades, dependiendo de su cardinalidad.
5. **Añadir restricciones:** Las restricciones del modelo E/R, como claves foráneas, unicidad y restricciones de integridad, se aplican en el modelo relacional.

3.2. Reglas para transformar relaciones Uno a Uno (1:1)

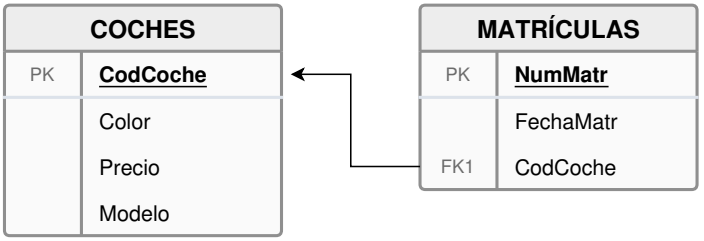
En las relaciones uno a uno, el proceso de transformación **dependerá del valor de la participación mínima** asociada a cada tipo de entidad en la interrelación.

1. **Ambas participaciones mínimas con valor 1**: cuando en una relación binaria ambas entidades tienen una cardinalidad mínima igual a 1, significa que ambas participan de manera obligatoria. En estos casos, **no es necesario crear una tabla adicional** aparte de las que ya representan a las entidades. La relación se implementa incorporando la clave primaria de una de las tablas como clave foránea en la otra, manteniendo así la conexión entre ambas entidades.

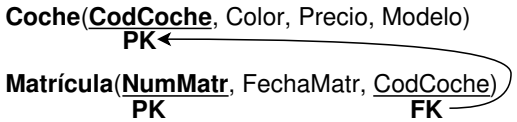
   Para ilustrar este caso, consideremos el siguiente diagrama Entidad-Relación (E/R), donde ambas entidades tienen una participación mínima obligatoria igual a 1:



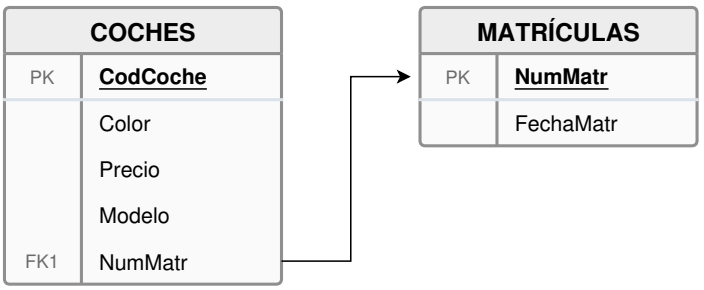
A partir de este diagrama, se obtiene el siguiente modelo relacional, donde la clave primaria de una entidad se utiliza como clave foránea en la otra:



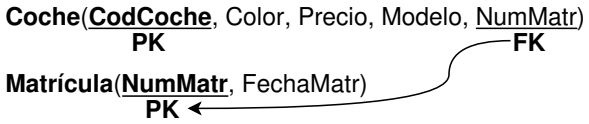
La conexión entre las tablas se puede representar gráficamente con el siguiente grafo, que refleja las claves foráneas utilizadas para establecer la relación:



Otra solución igualmente válida sería representar la relación intercambiando las entidades en la clave foránea. En este caso, el modelo relacional sería el siguiente:



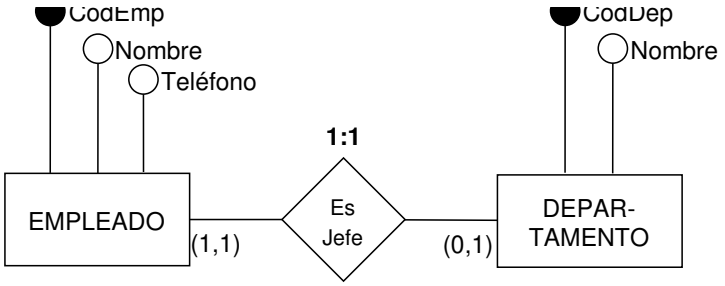
Y el grafo correspondiente a esta representación alternativa sería:



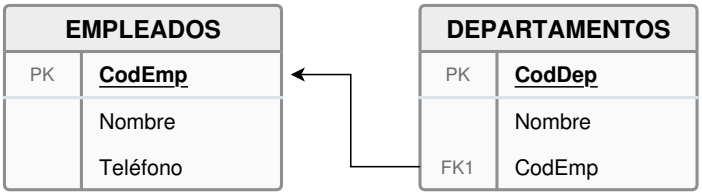
Ambas opciones son correctas y permiten implementar de manera eficiente la relación uno a uno con participaciones mínimas obligatorias en un sistema de bases de datos.

2. **Una participación mínima tiene valor 0**: cuando en una relación binaria solo una de las entidades tiene una participación mínima igual a 0 (es decir, participa de forma opcional), **no es necesario crear una tabla adicional**. En este caso, la clave primaria de la entidad con participación obligatoria se incorpora como clave foránea en la tabla de la entidad con participación opcional.

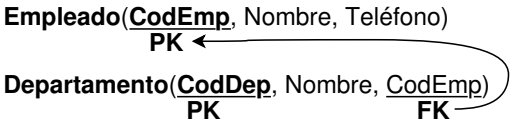
   Para ilustrar este caso, consideremos el siguiente diagrama Entidad-Relación (E/R), donde una entidad tiene participación mínima de valor 0:



A partir de este diagrama, se obtiene el siguiente modelo relacional, donde la clave primaria de la entidad con participación mínima de 0 se utiliza como clave foránea en la otra:

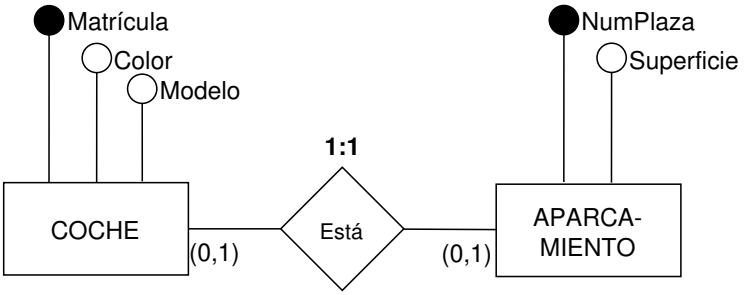


Y el grafo correspondiente a esta representación sería:



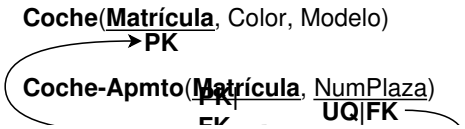
3. **Ambas participaciones mínimas con valor 0**: cuando en una relación binaria ambas entidades tienen una cardinalidad mínima igual a 0 (es decir, ambas participan de forma opcional), **sí es necesario crear una tabla adicional**. En esta nueva tabla, las claves primarias de ambas entidades se incluyen como claves foráneas, y una de ellas se designa como clave primaria de la nueva tabla y la otra como clave única (UQ).

   Para ilustrar este caso, consideremos el siguiente diagrama Entidad-Relación (E/R), donde una entidad tiene participación mínima de valor 0:



A partir de este diagrama, se obtiene el siguiente modelo relacional, donde se crea la nueva tabla:

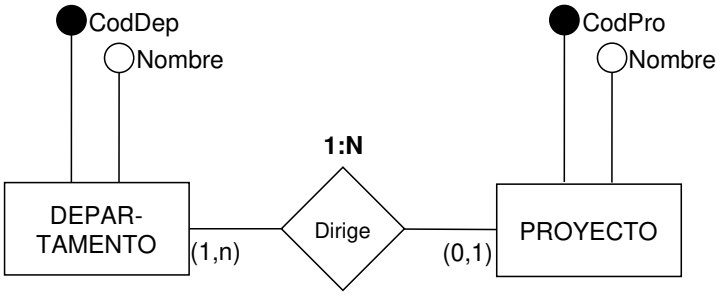
Y el grafo correspondiente a esta representación sería:



3.3. Reglas para transformar relaciones Uno a Muchos (1:N)

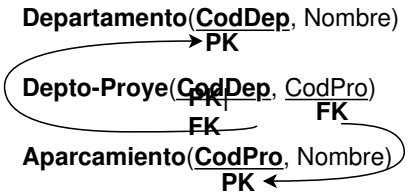
1. **Participación mínima 0 en el lado 1**: cuando en una relación 1:N la entidad del lado con cardinalidad máxima 1 tiene una participación mínima igual a 0, **sí es necesario crear una tabla adicional**. En esta nueva tabla, las claves primarias de ambas entidades se incluyen como claves foráneas, y **la clave primaria de la entidad del lado N se define como la clave principal** de la nueva tabla.

   Para ilustrar este caso, consideremos el siguiente diagrama Entidad-Relación (E/R):



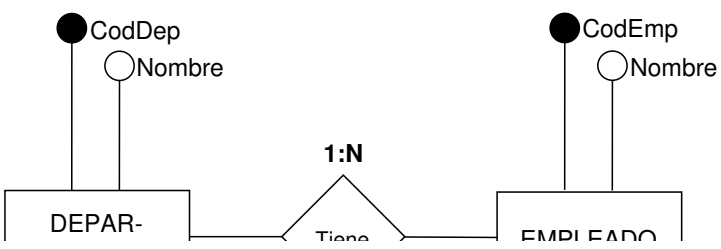
   A partir de este diagrama, se obtiene el siguiente modelo relacional, donde en la nueva tabla adicional las claves primarias de ambas entidades se incluyen como claves foráneas, y **la clave primaria de la entidad del lado N se define como la clave principal**.

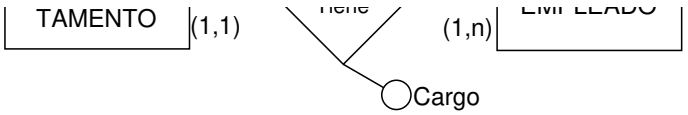
   Y el grafo correspondiente a esta representación sería:



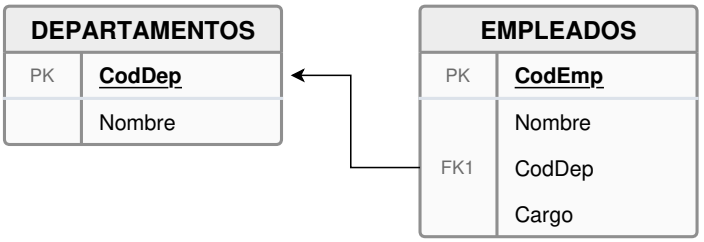
2. **Participación mínima 1 en el lado 1**: cuando en una relación 1:N la entidad del lado con cardinalidad 1 tiene una participación mínima igual a 1, **no se crea una tabla adicional**. En este caso, la clave primaria de la entidad del lado 1 se incorpora como clave foránea en la tabla de la entidad que participa con cardinalidad máxima muchos (N) o lo que es lo mismo, la tabla que representa a la entidad del lado N 'atrae' como clave foránea a la clave primaria del lado 1.

   Para ilustrar este caso, consideremos el siguiente diagrama Entidad-Relación (E/R):

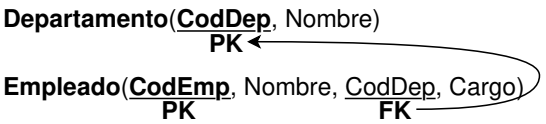




A partir de este diagrama, se obtiene el siguiente modelo relacional, donde la entidad del lado N atrae a la clave principal del lado 1 como foránea (y a los atributos de la relación si los hay, en este caso sin llegar a ser foráneos):



Y el grafo correspondiente a esta representación sería:

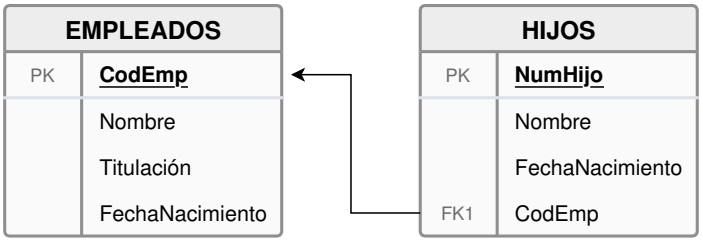


3.4. Reglas para transformar relaciones Muchos a Muchos (N:M)

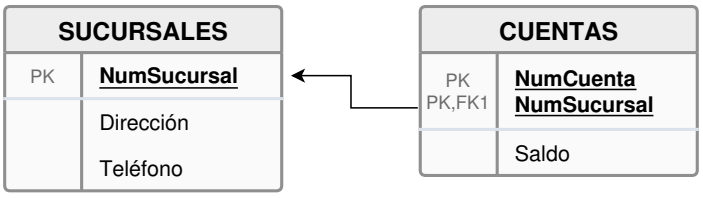
En una relación N:M, **siempre se crea una tabla adicional**. Esta tabla está compuesta por los atributos identificadores de las entidades que participan en la relación, así como por todos los atributos asociados al tipo de interrelación. La clave primaria de esta tabla se define como la combinación de todas las claves primarias de las entidades participantes en la relación.

3.5. Reglas para transformar relaciones de dependencia

1. **Débil en existencia**: en el caso de una relación débil por existencia, **no es necesario crear una tabla adicional**. Para implementar esta relación, **la clave primaria de la entidad del lado 1**(la entidad fuerte) **se transfiere como clave foránea a la tabla de la entidad del lado N** (la entidad débil).

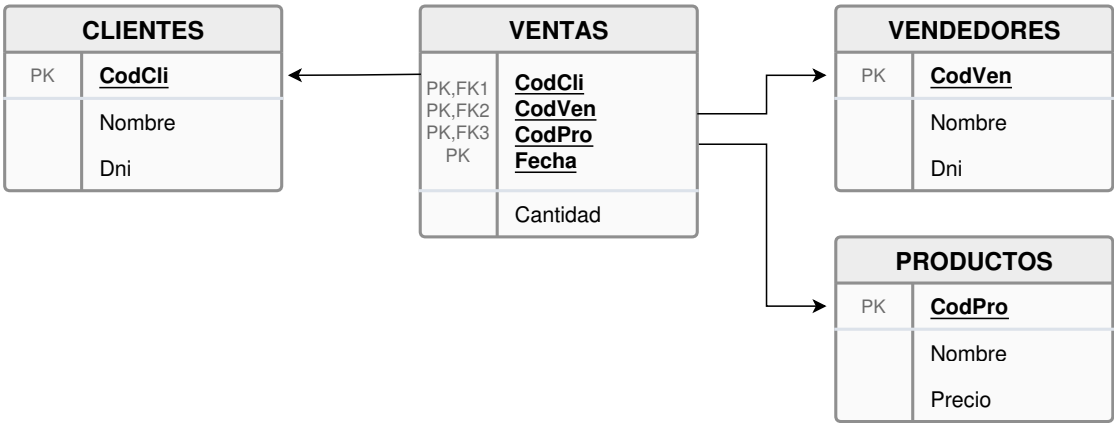


1. **Débil en identificación**: en el caso de una relación débil por identificación, **no es necesario crear una tabla adicional**. Para implementar esta relación en la base de datos, la clave primaria de la entidad del lado 1 (la entidad fuerte) se transfiere a la tabla de la entidad del lado N (débil) como **foránea y principal**. Esto asegura que cada instancia de la entidad débil esté identificada de manera única en el contexto de su asociación con la entidad fuerte.



3.6. Reglas para transformar relaciones N-arias

En una relación N-aria, además de las tablas correspondientes a cada entidad participante, **se genera una tabla adicional** para representar la relación. La clave primaria de esta nueva tabla se construye combinando las claves primarias de las entidades que participan con cardinalidad máxima "Muchos". Estas claves primarias también se marcan como claves foráneas para garantizar la integridad referencial. Las claves primarias de las entidades con cardinalidad distinta de "Muchos" se incluyen en la tabla de la relación únicamente como claves foráneas, sin formar parte de la clave primaria.

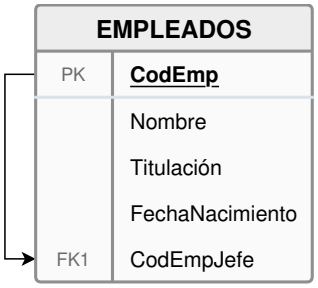


3.7. Reglas para transformar relaciones reflexivas

En las relaciones reflexivas, el proceso de transformación a tabla es similar al de las relaciones binarias. Sin embargo, un aspecto fundamental a tener en cuenta es que no puede haber dos campos con el mismo nombre en una misma tabla. Por ello, al propagar las claves foráneas en relaciones reflexivas, es necesario renombrarlas para evitar conflictos. A continuación, se explica cómo transformar estas relaciones según sus cardinalidades y participaciones.

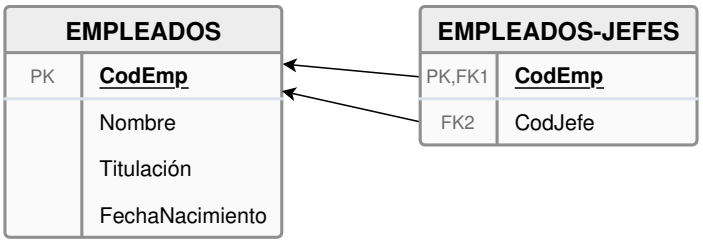
3.7.1. Relaciones reflexivas 1:1

En las relaciones reflexivas 1:1, si ambas entidades participan con cardinalidad máxima igual a 1, el tratamiento sigue siendo similar a las relaciones binarias 1:1. Si ambas participaciones mínimas son iguales a 1, **no es necesario crear una tabla adicional**. La clave primaria de la entidad se incluye como clave foránea dentro de la misma tabla, renombrada para evitar conflictos.



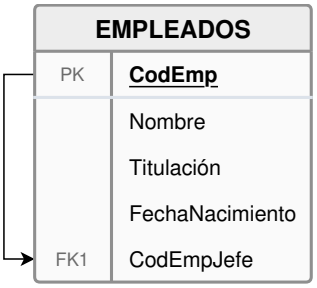
3.7.2. Relaciones reflexivas 1:N con participación mínima 0 en el lado 1

Cuando la participación mínima es 0 en el lado 1, **se debe generar una tabla adicional** para representar la relación reflexiva. Esta tabla incluirá la clave primaria de la entidad participante en dos campos distintos: uno de ellos actuará como clave primaria y clave foránea simultáneamente, mientras que el otro se utilizará únicamente como clave foránea. Además, la tabla podrá contener cualquier atributo propio de la relación reflexiva, y éste no será foráneo.



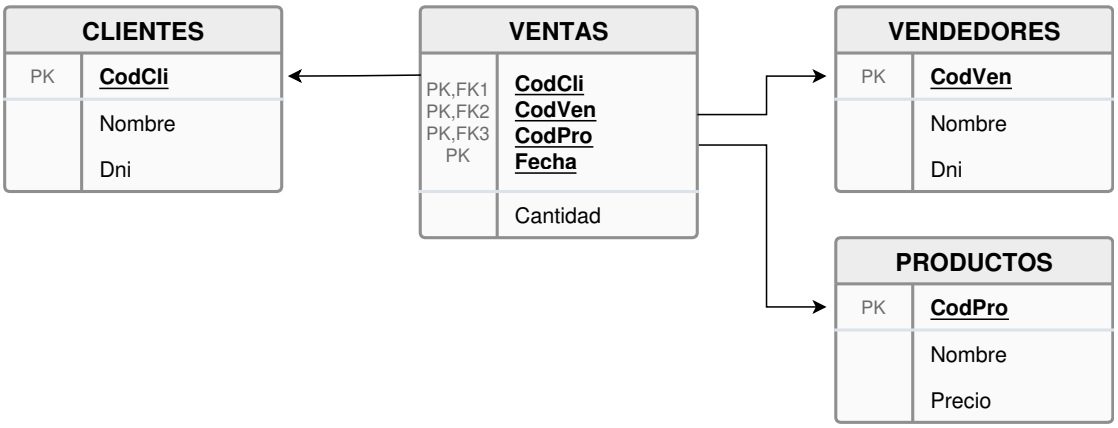
3.7.3. Relaciones reflexivas 1:N con participación mínima 1 en el lado 1

En una relación reflexiva 1:N con participación mínima igual a 1 en el lado 1, **no se genera una tabla** adicional. En este caso, la clave primaria de la entidad se incluye como clave foránea dentro de la misma tabla para representar la relación reflexiva. Este campo debe ser renombrado, por ejemplo, como `ClaveForanea`, para evitar duplicidades.



3.7.4. Relaciones reflexivas N:M

En las relaciones reflexivas N:M, **siempre es necesario generar una tabla adicional** para representar la relación. Esta tabla incluirá dos claves foráneas que referencian la clave primaria de la entidad participante. Estas claves deben ser renombradas, por ejemplo, como `Clave1` y `Clave2`, para evitar conflictos dentro de la tabla. Además, la tabla puede contener los atributos propios de la relación reflexiva, si los hubiera. La clave primaria de esta tabla será la combinación de las dos claves foráneas, lo que garantiza que cada par de relaciones reflexivas se represente de forma única.



3.8. Reglas para transformar relaciones jerárquicas

- En el caso de jerarquías, se debe crear una tabla para la entidad supertipo, salvo que tenga tan pocos atributos que resulte más práctico no representarla como una tabla independiente.
- Si una entidad subtipo no tiene atributos propios ni está relacionada con ninguna otra entidad,

desaparece del modelo. Sin embargo, si la entidad subtipo tiene algún atributo, se creará una tabla para ella. Si no posee una clave primaria propia, heredará la clave de la entidad supertipo.
- Cuando la relación es exclusiva, el atributo que define la jerarquía se incorpora en la tabla de la entidad supertipo. No obstante, si se ha optado por crear una tabla independiente para cada una de las entidades subtipo, no es necesario incluir este atributo en la tabla de la entidad supertipo.

4. Normalización

La normalización es un proceso fundamental en el diseño de bases de datos que tiene como objetivo estructurar la información de manera eficiente, eliminando redundancias y garantizando la integridad de los datos. A través de la aplicación de reglas y formas normales, se consigue una organización lógica de los datos que minimiza los riesgos de inconsistencia y mejora el rendimiento del sistema.

4.1. Objetivos de la Normalización de Relaciones

El proceso de normalización busca alcanzar varios objetivos clave que son esenciales para el correcto funcionamiento de una base de datos. Entre los principales se encuentran:

- **Eliminar redundancias innecesarias:** La normalización evita la duplicidad de datos en distintas tablas, lo que reduce el tamaño de la base de datos y mejora su mantenimiento.
- **Evitar anomalías en las operaciones:** Al eliminar redundancias y estructurar adecuadamente las tablas, se previenen anomalías que pueden surgir durante las operaciones de inserción, actualización o eliminación de datos.
- **Garantizar la integridad de los datos:** La normalización ayuda a definir reglas claras que aseguran la consistencia de los datos almacenados, protegiéndolos de inconsistencias.
- **Optimizar las consultas:** Al organizar los datos de manera lógica, las operaciones de consulta pueden ejecutarse de forma más eficiente, lo que mejora el rendimiento general del sistema.
- **Facilitar el mantenimiento y la escalabilidad:** Una base de datos bien normalizada es más fácil de mantener y actualizar, además de ser más adaptable a cambios o ampliaciones futuras.

Ejemplo Práctico: Eliminación de Redundancias

Antes de la Normalización (Redundancias Presentes)

Imagina una tabla que registra información sobre pedidos en una tienda online:

Pedido_ID	Cliente_ID	Cliente	Cliente_Dirección	Producto_ID	Producto	Uds
1	C001	Juan Pérez	Calle Sol, 15	P001	Auriculares	2
2	C001	Juan Pérez	Calle Sol, 15	P002	Teclado	1
3	C002	Ana López	Calle Luna, 8	P001	Auriculares	1

Problemas:

1. La información del cliente (`Cliente_Nombre`, `Cliente_Dirección`) se repite para cada pedido que realiza, ocupando más espacio de almacenamiento.
2. Si cambia la dirección de un cliente, es necesario actualizar múltiples filas, lo que aumenta el

riesgo de inconsistencias.
3. No hay separación clara entre los datos del cliente, los productos y los pedidos, lo que dificulta la gestión.

Después de la Normalización (Primera Forma Normal)

Después de aplicar la normalización, un proceso que se verá más adelante en este mismo tema, la tabla anterior quedará representada por la siguiente distribución de tablas:

Tabla: Clientes

Cliente_ID	Cliente	Dirección
C001	Juan Pérez	Calle Sol, 15
C002	Ana López	Calle Luna, 8

Tabla: Productos

Producto_ID	Producto
P001	Auriculares
P002	Teclado

Tabla: Pedidos

Pedido_ID	Cliente_ID	Producto_ID	Uds
1	C001	P001	2
2	C001	P002	1
3	C002	P001	1

Ventajas:

- La información del cliente y de los productos se almacena una sola vez en sus respectivas tablas.
- Si cambia la dirección de un cliente, solo es necesario actualizar una fila en la tabla `Clientes`.
- La estructura es modular, lo que facilita agregar nuevos datos o modificar los existentes.

Mejoras inherentes a la normalización

- Antes de la normalización, no puedes agregar un nuevo cliente sin que realice un pedido, porque el cliente y el pedido están en la misma tabla. Después de la normalización, puedes agregar un cliente directamente en la tabla `Clientes` sin necesidad de crear un pedido.
- En la tabla original, si eliminas el único pedido de un cliente, también pierdes toda la información del cliente. Después de la normalización, puedes eliminar un pedido sin afectar los datos del cliente en la tabla `Clientes`.
- Si el cliente `Juan Pérez` cambia de dirección, debes actualizar todas las filas que contienen su información en la tabla original. Después de la normalización, solo necesitas actualizar una fila en la tabla `Clientes`.

Nota: La normalización en este ejemplo no corresponde directamente a un proceso inverso que derive en un diagrama Entidad-Relación (E/R), ya que su punto de partida no es un modelo conceptual planificado, sino una tabla desestructurada que combina datos de múltiples entidades y relaciones. La normalización tiene como objetivo reorganizar los datos existentes para eliminar redundancias y dependencias indebidas, usando claves como

`Cliente_ID` y `Producto_ID` para facilitar la referencia entre tablas. Si intentáramos reconstruir un modelo E/R desde las tablas normalizadas, el resultado reflejaría la estructura optimizada de las tablas y no necesariamente las entidades y relaciones originales del problema inicial. Esto refuerza la idea de que la normalización es un proceso orientado a la optimización de datos existentes, mientras que el diseño E/R se enfoca en la planificación conceptual de un sistema desde cero.

4.2. Dependencias Funcionales

Las dependencias funcionales son un concepto clave en el proceso de normalización. Representan la relación entre los atributos de una tabla y cómo uno o más atributos determinan el valor de otros. Identificar y entender estas dependencias es fundamental para estructurar correctamente una base de datos y avanzar en las distintas formas normales.

Una dependencia funcional se expresa como `A → B`, donde el valor del atributo `A` (o conjunto de atributos) determina de manera única el valor del atributo `B`. En el contexto del ejemplo anterior, donde organizamos pedidos, clientes y productos en tablas normalizadas, las dependencias funcionales nos ayudan a identificar relaciones clave entre los datos. A continuación se detallan los diferentes tipos de dependencias con respecto al ejemplo anterior.

4.2.1. Dependencia Funcional

Una **dependencia funcional** se da cuando el valor de un atributo o conjunto de atributos (`A`) determina el valor de otro atributo (`B`). Esto significa que no puede haber dos registros en la tabla con el mismo valor para `A` y un valor diferente para `B`. Por ejemplo en la tabla `Clientes`, existe una dependencia funcional:

- Cliente_ID → Nombre, Dirección

Esto significa que el identificador único del cliente (`Cliente_ID`) determina de forma unívoca el nombre y la dirección del cliente. Si `Cliente_ID` es 1, entonces sabemos que el nombre es `Juan Pérez` y la dirección es `Calle Sol, 15`. No puede haber dos clientes con el mismo `Cliente_ID` y nombres o direcciones diferentes.

El principal propósito de las dependencias funcionales es evitar duplicación innecesaria e inconsistencias en los datos. Si los valores de un atributo se repiten sin control, esto aumenta la probabilidad de errores al actualizar o eliminar información. Por ejemplo, si se actualiza la dirección de un cliente en un registro pero no en otro, se genera una inconsistencia en la base de datos. Este tipo de dependencia está estrechamente ligado al cumplimiento de la Primera Forma Normal (1FN), que exige que los datos estén organizados en tablas con valores atómicos y filas únicas, estableciendo la base para un diseño relacional sólido.

4.2.2. Dependencia Funcional Completa

Una **dependencia funcional completa** ocurre cuando un atributo depende de todo un conjunto de atributos y no solo de una parte de ellos. Esto significa que si un atributo depende de una clave primaria compuesta, debe depender completamente de todos los elementos de esa clave. Por ejemplo, en una tabla que relaciona pedidos y productos, la cantidad pedida (`Cantidad`) está determinada por la combinación de `Pedido_ID` y `Producto_ID`:

- (Pedido_ID, Producto_ID) → Cantidad

Esto implica que la cantidad de un producto en un pedido específico solo puede conocerse si consideramos ambos atributos juntos. Si eliminamos cualquiera de ellos, como `Producto_ID`, no

podríamos determinar la cantidad correctamente, ya que un pedido puede contener múltiples productos.

El objetivo principal de las dependencias funcionales completas es evitar dependencias parciales que generen redundancias y complicaciones en el mantenimiento de la base de datos. Si un atributo depende solo de una parte de la clave primaria, se corre el riesgo de que la información quede mal estructurada, lo que puede llevar a duplicaciones innecesarias y dificultades para realizar consultas o actualizaciones.

Esta dependencia está estrechamente ligada al cumplimiento de la Segunda Forma Normal (2FN), que exige que cada atributo de una tabla dependa completamente de la clave primaria, eliminando las dependencias parciales y mejorando la estructura de los datos.

4.2.3. Dependencia Transitiva

Una **dependencia transitiva** ocurre cuando un atributo depende funcionalmente de otro a través de un tercer atributo. Esto significa que si un atributo `A` determina `B`, y `B` determina `C`, entonces existe una dependencia transitiva entre `A` y `C`. Por ejemplo, en una tabla que combina pedidos y clientes como la siguiente, `Cliente_Nombre` depende transitivamente de `Pedido_ID` a través de `Cliente_ID`:

Pedido_ID	Cliente_ID	Cliente_Nombre	Dirección	Producto_ID	Cantidad
1	1	Juan Pérez	Calle Sol, 15	1	2
2	1	Juan Pérez	Calle Sol, 15	2	1

- Pedido_ID → Cliente_ID → Cliente_Nombre

En este caso, el nombre del cliente no depende directamente del identificador del pedido, sino que lo hace a través del identificador del cliente. Esto introduce redundancia, ya que los datos del cliente se repetirán en cada pedido que realice.

El objetivo principal de eliminar dependencias transitivas es evitar redundancias y anomalías en los datos. Si los datos dependen de una relación indirecta, pueden repetirse innecesariamente en múltiples filas, lo que incrementa el riesgo de inconsistencias al actualizar o eliminar información. Por ejemplo, si cambia el nombre de un cliente, habría que modificarlo en todas las filas donde aparezca, lo que es propenso a errores.

Esta dependencia está ligada al cumplimiento de la Tercera Forma Normal (3FN), que garantiza que todos los atributos no clave dependan únicamente de la clave primaria, eliminando dependencias transitivas y mejorando la consistencia y eficiencia de la base de datos.

4.3. Formas normales

La normalización es un proceso sistemático que consiste en aplicar una serie de pasos secuenciales, cada uno de los cuales se corresponde con una forma normal (FN). Este proceso tiene como objetivo optimizar la estructura de las tablas en una base de datos, eliminando redundancias, minimizando riesgos de inconsistencias y garantizando la integridad de los datos. Conforme avanzamos en las formas normales, las tablas adoptan una estructura más clara, eficiente y fácil de mantener.

El concepto de formas normales fue introducido por Edgar F. Codd en 1972, quien propuso las tres

primeras formas normales (1FN, 2FN y 3FN). Estas constituyen la base del diseño relacional. En 1974, se añadió la Forma Normal de Boyce-Codd (BCNF), que perfecciona algunos casos no resueltos por la 3FN. Posteriormente, entre 1977 y 1979, Ronald Fagin amplió la teoría con la Cuarta (4FN) y la Quinta Forma Normal (5FN), dirigidas a situaciones más complejas en bases de datos especializadas.

En este curso, nos centraremos en las tres primeras formas normales (1FN, 2FN y 3FN), ya que cubren la mayoría de casos prácticos y constituyen la base de una base de datos correctamente diseñada. Aunque existen formas normales superiores como la BCNF, 4FN y 5FN, estas se aplican en contextos avanzados y no suelen ser necesarias en la mayoría de aplicaciones comunes.

Es importante entender que todas las formas normales se construyen de manera jerárquica. **Una tabla que cumple con la Segunda Forma Normal (2FN) también cumple automáticamente con la Primera Forma Normal (1FN), y lo mismo ocurre al pasar de la 2FN a la Tercera Forma Normal (3FN)**. Este proceso de anidación asegura que cada forma normal refuerza y perfecciona las propiedades logradas en las etapas anteriores.

Al completar cada etapa de la normalización, las tablas se convierten en componentes más especializados y eficientes dentro del sistema. Este enfoque no solo mejora el rendimiento y la consistencia de la base de datos, sino que también facilita su escalabilidad y mantenimiento a largo plazo.

4.3.1. Primera Forma Normal (1FN)

La Primera Forma Normal (1FN) establece los cimientos para un diseño estructurado y eficiente de bases de datos. Para que una tabla cumpla con esta forma normal, todos sus **campos deben ser atómicos**, lo que significa que cada celda debe contener un único valor indivisible. Además, **no deben existir grupos de campos repetitivos**, lo que implica que los datos relacionados deben ser organizados en tablas separadas y que cada fila sea única.

Un ejemplo de una tabla que no cumple con la 1FN podría ser una tabla que almacena información sobre clientes y sus números de teléfono, pero en la que algunos clientes tienen múltiples números almacenados en un solo campo o concatenados. Consideremos la siguiente tabla:

ID Cliente	Nombre	Apellido	Teléfono
101	Laura	González	600-123-456, 600-789-101
202	Pedro	Martínez	700-654-321
303	Carmen	López	800-987-654, 800-321-876

En este ejemplo, la columna `Teléfono` contiene múltiples valores para algunos clientes. Esto viola la regla de atomicidad de la 1FN, ya que los valores no son indivisibles. Además, esta estructura complica las operaciones de búsqueda y actualización, dado que no es posible manejar eficientemente los números de teléfono sin procesarlos manualmente. Para cumplir con la 1FN, el primer paso es hacer que todos sus campos sean atómicos, para ello:

ID Cliente	Nombre	Apellido	Teléfono
101	Laura	González	600-123-456
101	Laura	González	600-789-101
202	Pedro	Martínez	700-654-321
303	Carmen	López	800-987-654
303	Carmen	López	800-321-876

El siguiente paso es eliminar las redundancias de grupos de campos que aparecen al examinar estos registros. Y es que podemos darnos cuenta que pueden existir un grupo de campos repetidos para `Teléfono`, que serían los campos `ID Cliente`, `Nombre` y `Apellido`. Por lo tanto la solución es separar la tabla en dos, una que relaciona al teléfono con el cliente y otra que tiene los datos principales del cliente, quedando de la siguiente forma:

ID Cliente	Nombre	Apellido
101	Laura	González
202	Pedro	Martínez
303	Carmen	López

La segunda tabla, llamada `Teléfono del cliente`, se utiliza para almacenar los números de teléfono relacionados con cada cliente. Esto se logra vinculando cada número de teléfono con el cliente correspondiente mediante la clave primaria `ID Cliente`+`Teléfono`:

ID Cliente	Teléfono
101	600-123-456
101	600-789-101
202	700-654-321
303	800-987-654
303	800-321-876

Con esta reorganización, cada número de teléfono se almacena de manera atómica y en una fila independiente, eliminando las listas concatenadas y respetando la regla de no repetición de grupos de campos. Esta estructura permite manejar los datos de forma más eficiente, facilitando las consultas, actualizaciones y eliminaciones sin afectar la integridad de la base de datos.

Cumplir con la Primera Forma Normal no solo asegura que los datos estén organizados de manera lógica y clara, sino que también elimina redundancias estructurales y prepara las tablas para las siguientes etapas de normalización. La separación en dos tablas, como en este ejemplo, permite escalar fácilmente el modelo y adaptarlo a futuras necesidades, garantizando un diseño funcional y bien estructurado.

4.3.2. Segunda Forma Normal (2FN)

La Segunda Forma Normal (2FN) es un paso fundamental en el proceso de normalización y tiene como objetivo eliminar las dependencias parciales en tablas que ya cumplen con la Primera Forma Normal (1FN). Para que una tabla esté en 2FN, todos los atributos no clave deben depender completamente de toda la clave primaria y no solo de una parte de ella. Esto es especialmente

relevante en tablas donde la clave primaria está compuesta por más de un atributo.

Consideremos el siguiente ejemplo de una tabla que registra información sobre empleados, sus habilidades y su lugar actual de trabajo. En este caso, la clave primaria está compuesta por los atributos `Empleado` y `Habilidad`:

Empleado	**Habilidad**	**Lugar actual de trabajo**
López	Carpintería	Calle Central, 42
López	Pintura	Calle Central, 42
López	Soldadura	Calle Central, 42
Martínez	Fontanería	Avenida Norte, 18
Gómez	Electrónica	Avenida Norte, 18
Gómez	Programación	Avenida Norte, 18
Torres	Fontanería	Avenida Norte, 18

En esta tabla, el atributo `Lugar actual de trabajo` depende únicamente del atributo `Empleado`, no de toda la clave primaria compuesta (`Empleado`, `Habilidad`). Esto genera una dependencia parcial, lo que introduce redundancia, ya que el lugar de trabajo del empleado se repite en cada fila asociada a sus habilidades.

Para solucionar este problema y cumplir con la 2FN, debemos dividir la tabla en dos. Una tabla almacenará la relación entre empleados y sus lugares de trabajo, y otra tabla gestionará la relación entre empleados y sus habilidades. Esta separación elimina las dependencias parciales, ya que cada atributo no clave dependerá completamente de la clave primaria de su tabla respectiva.

Tabla: Empleados

Empleado	**Lugar actual de trabajo**
López	Calle Central, 42
Martínez	Avenida Norte, 18
Gómez	Avenida Norte, 18
Torres	Avenida Norte, 18

Tabla: Habilidades de los empleados

Empleado	**Habilidad**
López	Carpintería
López	Pintura
López	Soldadura
Martínez	Fontanería
Gómez	Electrónica
Gómez	Programación
Torres	Fontanería

Con esta reorganización, los datos del lugar de trabajo se almacenan una sola vez en la tabla `Empleados`, eliminando la redundancia asociada. La tabla `Habilidades de los empleados` mantiene la relación única entre cada empleado y sus habilidades, asegurando que cada atributo no clave dependa completamente de toda la clave primaria.

Al cumplir con la 2FN, se logra una estructura más eficiente y fácil de mantener. Este paso reduce significativamente las redundancias y previene anomalías durante las actualizaciones o eliminaciones. Además, establece una base sólida para avanzar hacia formas normales más avanzadas, como la Tercera Forma Normal (3FN), que se enfocará en eliminar las dependencias transitivas restantes.

4.3.3. Tercera Forma Normal (3FN)

La Tercera Forma Normal (3FN) tiene como objetivo eliminar las dependencias transitivas en tablas que ya cumplen con la Segunda Forma Normal (2FN). Para que una tabla esté en 3FN, todos los atributos no clave deben depender únicamente de la clave primaria y no de otros atributos no clave. Esto asegura que los datos estén completamente normalizados, evitando redundancias y anomalías durante las actualizaciones o eliminaciones.

Consideremos el siguiente ejemplo de una tabla que registra información sobre los ganadores de un torneo, su año y su fecha de nacimiento. En este caso, la clave primaria está compuesta por los atributos `Torneo` y `Año`:

Torneo	**Año**	**Ganador**	**Fecha de nacimiento**
Summer Challenge	2020	Luis Fernández	15 de enero de 1985
Winter Showdown	2021	Clara Martínez	8 de abril de 1990
Autumn Clash	2021	Luis Fernández	15 de enero de 1985
Spring Invitational	2022	Diego García	20 de junio de 1988

En esta tabla, el atributo `Fecha de nacimiento` depende del atributo no clave `Ganador`, no de la clave primaria compuesta (`Torneo`, `Año`). Esta dependencia transitiva introduce redundancia, ya que la fecha de nacimiento del ganador se repite para cada torneo que ha ganado.

Para cumplir con la 3FN, debemos reorganizar los datos en dos tablas separadas. Una tabla contendrá la relación entre los torneos y los ganadores, y otra almacenará las fechas de nacimiento relacionadas con los ganadores. Esto elimina las dependencias transitivas y asegura que todos los atributos no clave dependan únicamente de la clave primaria.

Tabla: Ganadores del torneo

Torneo	Año	Ganador
Summer Challenge	2020	Luis Fernández
Winter Showdown	2021	Clara Martínez
Autumn Clash	2021	Luis Fernández
Spring Invitational	2022	Diego García

Tabla: Fechas de nacimiento de los ganadores

Ganador	Fecha de nacimiento
Luis Fernández	15 de enero de 1985
Clara Martínez	8 de abril de 1990
Diego García	20 de junio de 1988

Con esta reorganización, eliminamos la redundancia en la columna `Fecha de nacimiento`, que ahora se almacena una sola vez en la tabla `Fechas de nacimiento de los ganadores`. La tabla `Ganadores del torneo` se centra exclusivamente en la relación entre los torneos y sus respectivos ganadores, asegurando que todos los atributos no clave dependan únicamente de la clave primaria (`Torneo`, `Año`).

Al cumplir con la 3FN, se logra una estructura de base de datos completamente normalizada, donde las dependencias transitivas han sido eliminadas. Esto mejora significativamente la eficiencia y consistencia del sistema, permitiendo manejar datos más complejos sin redundancia innecesaria y reduciendo el riesgo de errores durante las operaciones de actualización y eliminación.

4.3.4. Forma Normal de Boyce-Codd (BCNF)

La Forma Normal de Boyce-Codd (BCNF) es una extensión de la Tercera Forma Normal (3FN) que aborda ciertas limitaciones en situaciones específicas. Una tabla está en BCNF si cumple con la 3FN y, además, para cada dependencia funcional no trivial de la forma `A → B`, el atributo `A` es una superclave. Esto significa que no puede existir ninguna dependencia funcional donde un atributo no clave determine otro atributo.

La BCNF es particularmente útil en tablas con claves compuestas y relaciones más complejas, donde las dependencias funcionales pueden violar la integridad de los datos incluso si se cumple la 3FN. Aunque la BCNF se considera una mejora, su implementación puede requerir dividir tablas en estructuras aún más pequeñas, lo que a veces puede afectar el rendimiento. En la práctica, se aplica principalmente en sistemas avanzados que manejan datos altamente interrelacionados.

4.3.5. Cuarta Forma Normal (4FN)

La Cuarta Forma Normal (4FN) se centra en eliminar dependencias multivaluadas en una tabla. Una tabla está en 4FN si está en BCNF y no contiene dependencias multivaluadas, a menos que estas sean implícitas debido a la clave primaria. Las dependencias multivaluadas ocurren cuando un atributo está relacionado con múltiples valores independientes de otro atributo en la misma tabla, lo que genera redundancias.

Para resolver este problema, las tablas con dependencias multivaluadas deben dividirse en tablas separadas, asegurando que cada tabla maneje una relación única entre los atributos. La 4FN se aplica principalmente en sistemas con estructuras de datos complejas que requieren un diseño altamente normalizado para evitar inconsistencias.

4.3.6. Quinta Forma Normal (5FN o Proyección-Join)

La Quinta Forma Normal (5FN), también conocida como Forma Normal de Proyección-Join, se enfoca en eliminar dependencias de unión. Una tabla está en 5FN si está en 4FN y todos los datos pueden representarse como proyecciones de la tabla original sin pérdida de información al unirlas. Esto significa que la tabla no debe contener dependencias complejas que puedan generar redundancia o inconsistencia al descomponerla en tablas más pequeñas.

La 5FN es necesaria en casos muy avanzados, donde las relaciones entre los datos son tan complejas que la eliminación completa de redundancias requiere un nivel adicional de descomposición. Aunque rara vez se aplica en la práctica, la 5FN garantiza un diseño relacional absolutamente optimizado y sin duplicaciones innecesarias.