UF2175 -
Diseño de Bases de Datos Relacionales

Unidad 3: El modelo relacional

By: Sergi Faura Alsina

SOC Servei d'Ocupació de Catalunya
Generalitat de Catalunya
MINISTERIO DE EDUCACIÓN, FORMACIÓN PROFESIONAL Y DEPORTES
MINISTERIO DE TRABAJO Y ECONOMÍA SOCIAL
SEPE

Índice

3. El modelo relacional

El **modelo relacional** es uno de los pilares fundamentales de la teoría de bases de datos y ha sido la base para el desarrollo de sistemas de gestión de bases de datos (DBMS) durante las últimas décadas. Introducido por el matemático Edgar F. Codd en 1970, este modelo revolucionó la manera en que se almacenan, gestionan y manipulan los datos en las organizaciones. Antes de su aparición, los sistemas de bases de datos utilizaban estructuras jerárquicas y de red, que, aunque eficaces en su contexto, carecían de la flexibilidad y la simplicidad que el modelo relacional proporcionó.

El modelo relacional se basa en conceptos matemáticos bien definidos, utilizando principalmente la teoría de conjuntos y la lógica de primer orden. En su núcleo, organiza los datos en **relaciones** (también conocidas como tablas), donde cada relación está compuesta por filas y columnas. Las **filas** representan instancias o registros de datos individuales, mientras que las **columnas** representan los atributos o campos que describen esos datos. Esta estructura tabular no solo facilita la comprensión y la manipulación de los datos, sino que también permite una mayor independencia de los datos respecto a las aplicaciones que los utilizan.

Una de las características más destacadas del modelo relacional es su capacidad para asegurar la **integridad** y **consistencia** de los datos a través de reglas bien definidas, como las **restricciones de integridad** y las **claves**. Las claves, incluyendo las claves primarias, candidatas y foráneas, juegan un papel crucial en la identificación única de registros y en el establecimiento de relaciones entre diferentes tablas. Estas relaciones permiten la normalización de los datos, un proceso mediante el cual se eliminan las redundancias y se evitan las anomalías de actualización, asegurando así que la base de datos permanezca coherente y eficiente.

La **teoría de la normalización** es otro componente esencial del modelo relacional. Este proceso se lleva a cabo en varias etapas, conocidas como formas normales, que van desde la Primera Forma Normal (1FN) hasta formas más avanzadas como la Cuarta (4FN) y Quinta Forma Normal (5FN). Cada una de estas formas normales aborda un tipo específico de redundancia o dependencia en los datos, y su correcta aplicación es fundamental para diseñar bases de datos relacionales optimizadas.

Además de la normalización, el modelo relacional también introduce el concepto de **desnormalización**, que, aunque parece contradecir el objetivo de la normalización, se utiliza en situaciones específicas para mejorar el rendimiento de la base de datos en ciertos escenarios de consulta.

El modelo relacional también define un conjunto de **operaciones algebraicas** que permiten manipular los datos de manera eficiente. Estas operaciones forman el núcleo de los lenguajes de consulta relacional, como el SQL (Structured Query Language), que se ha convertido en el estándar para la gestión y manipulación de bases de datos relacionales.

En resumen, el modelo relacional no solo es una herramienta poderosa para la organización y manipulación de datos, sino que también establece un marco teórico robusto que garantiza la integridad, consistencia y optimización de la información almacenada. Su adopción masiva en la industria de las bases de datos es un testimonio de su eficacia y versatilidad, y sigue siendo una referencia clave en el diseño y desarrollo de sistemas de bases de datos modernos. A lo largo de este tema, exploraremos en profundidad la evolución del modelo relacional, su estructura, las claves que lo sustentan, las restricciones de integridad que aseguran la validez de los datos, y la teoría de la normalización que permite mantener los datos organizados y libres de redundancias.

3.1. Evolución del modelo relacional.

El **modelo relacional** ha sido la base dominante para la organización y gestión de bases de datos desde su introducción en 1970. Desarrollado por Edgar F. Codd, un investigador de IBM, este modelo revolucionó la forma en que se almacenan y manipulan los datos, ofreciendo una estructura más flexible, eficiente y fácil de entender en comparación con los modelos de bases de datos jerárquicos y de red que prevalecían en ese momento.

Contexto Previo al Modelo Relacional

Antes de la aparición del modelo relacional, las bases de datos se organizaban principalmente utilizando modelos jerárquicos y de red. El **modelo jerárquico** organizaba los datos en una estructura de árbol, donde cada registro tenía un único "padre" y múltiples "hijos". Aunque este enfoque era eficiente para ciertas aplicaciones específicas, tenía limitaciones significativas en términos de flexibilidad y complejidad de las relaciones que podían representarse.

El **modelo de red**, también conocido como el modelo CODASYL, permitía relaciones más complejas al permitir que un registro tuviera múltiples padres. Sin embargo, la complejidad de este modelo hacía que las bases de datos fueran difíciles de mantener y actualizar, y requería un conocimiento profundo de la estructura de datos por parte de los programadores.

La Revolución del Modelo Relacional

Edgar F. Codd propuso el modelo relacional como una solución a las limitaciones de los modelos jerárquicos y de red. En su documento seminal titulado "A Relational Model of Data for Large Shared Data Banks" (1970), Codd describió un enfoque en el que los datos se organizaban en **tablas** (o relaciones) que podían ser manipuladas utilizando operaciones matemáticas basadas en la teoría de conjuntos y la lógica de primer orden.

El modelo relacional introdujo varios conceptos clave que transformaron la gestión de bases de datos:

1. **Estructura Tabular**: Los datos se organizan en tablas, donde cada tabla representa una relación y está compuesta por filas (tuplas) y columnas (atributos). Esta estructura simplifica la representación de los datos y permite que las tablas se relacionen entre sí mediante claves.
2. **Independencia de los Datos**: El modelo relacional promueve la independencia lógica y física de los datos, lo que significa que las aplicaciones pueden acceder a los datos sin necesidad de conocer la estructura física de almacenamiento. Esto facilita el desarrollo de aplicaciones y la migración de datos.
3. **Lenguaje de Consulta Declarativo**: Codd sugirió el uso de un lenguaje de consulta declarativo, que eventualmente llevó al desarrollo de SQL (Structured Query Language). Este enfoque permite a los usuarios especificar qué datos quieren recuperar sin necesidad de definir cómo deben ser recuperados, simplificando el proceso de consulta y manipulación de datos.
4. **Integridad y Consistencia de los Datos**: El modelo relacional incluye mecanismos para asegurar la integridad referencial y la consistencia de los datos mediante el uso de claves primarias, claves foráneas y restricciones de integridad.

Adopción y Desarrollo del Modelo Relacional

Después de la publicación del trabajo de Codd, IBM comenzó a desarrollar sistemas de bases de datos basados en el modelo relacional. Uno de los primeros productos comerciales fue el sistema de bases de datos **System R**, que sirvió como prototipo para la implementación de SQL. Durante los años 1980, varios sistemas de bases de datos relacionales se lanzaron al mercado, incluyendo **Oracle**, **DB2** de IBM y **Ingres**, entre otros.

La estandarización del lenguaje SQL por parte de ANSI y ISO en los años 1980 y 1990 consolidó aún más la posición del modelo relacional como el estándar de facto para la gestión de bases de datos. SQL proporcionó una interfaz universal para interactuar con bases de datos relacionales, lo que facilitó la adopción de estas tecnologías en una amplia gama de aplicaciones.

Avances y Adaptaciones Recientes

A lo largo de las décadas, el modelo relacional ha evolucionado para adaptarse a nuevas demandas y tecnologías emergentes. Aunque el modelo relacional sigue siendo dominante, han surgido nuevos enfoques y tecnologías en respuesta a los desafíos de manejar grandes volúmenes de datos, la necesidad de escalabilidad y la diversidad de tipos de datos.

- **Optimización y Rendimiento**: Los sistemas de bases de datos relacionales han incorporado mejoras en el rendimiento, como índices avanzados, técnicas de particionamiento y almacenamiento en memoria, para manejar volúmenes crecientes de datos y mejorar la eficiencia de las consultas.
- **Bases de Datos Distribuidas**: Con el crecimiento de las aplicaciones globales y la necesidad de alta disponibilidad, se han desarrollado bases de datos relacionales distribuidas que permiten el almacenamiento y la gestión de datos en múltiples ubicaciones geográficas.

- **Integración con NoSQL**: A medida que los sistemas NoSQL ganaron popularidad debido a su flexibilidad y capacidad para manejar datos no estructurados, muchos sistemas relacionales han incorporado características para integrar y manejar datos NoSQL dentro de una infraestructura relacional.

Perspectivas Futuras

El modelo relacional ha demostrado ser notablemente resistente y adaptable a lo largo del tiempo. Aunque enfrenta competencia de modelos de bases de datos NoSQL y otros enfoques especializados, sigue siendo la opción preferida para muchas aplicaciones empresariales críticas, gracias a su robustez, fiabilidad y capacidad para garantizar la integridad de los datos.

En el futuro, es probable que el modelo relacional continúe evolucionando, incorporando más capacidades de automatización, soporte para datos más diversos y mejores herramientas para la gestión de grandes volúmenes de datos en entornos distribuidos. Además, el uso de inteligencia artificial y aprendizaje automático para optimizar el rendimiento de las bases de datos relacionales es una tendencia emergente que podría llevar a nuevas innovaciones en este campo.

Conclusión

La evolución del modelo relacional representa una historia de innovación constante y adaptación. Desde su introducción por Edgar F. Codd en 1970, ha transformado la manera en que las organizaciones manejan y utilizan los datos. A pesar de los cambios en la tecnología y las nuevas demandas del mercado, el modelo relacional sigue siendo una piedra angular en el campo de la gestión de bases de datos, proporcionando una base sólida para el desarrollo de aplicaciones confiables y eficientes.

3.2. Estructura del modelo relacional:

El modelo relacional se basa en una estructura formal para la organización y gestión de datos, utilizando conceptos matemáticos que permiten representar los datos y las relaciones entre ellos de una manera coherente y lógica. Esta estructura es clave para asegurar que los datos se almacenen de manera eficiente, se accedan de manera efectiva y se mantengan íntegros. A continuación, se describe en detalle la estructura del modelo relacional, comenzando con el concepto de relación y sus propiedades, seguido de los atributos y dominios, y finalizando con las tuplas, grado y cardinalidad, así como la relación entre estas estructuras y las tablas en una base de datos.

3.2.1. El concepto de relación. Propiedades de las relaciones.

En el modelo relacional, una **relación** es el concepto fundamental para la organización de los datos. Una relación puede ser entendida como una tabla en la cual los datos están organizados en filas y columnas. Cada fila en una tabla representa un registro único, mientras que cada columna corresponde a un atributo que describe alguna característica de los datos.

Una relación en términos matemáticos es un subconjunto del producto cartesiano de varios dominios. Si consideramos una relación `R` que involucra `n` atributos, podemos decir que `R` es un subconjunto del producto cartesiano de `n` dominios `D1, D2, ..., Dn`, es decir, `R ⊆ D1 x D2 x ... x Dn`.

Propiedades de las Relaciones:

- **Sin duplicados**: Cada fila (o tupla) en una relación debe ser única; no puede haber duplicados dentro de una tabla. Esto asegura que cada registro sea identificable de manera única, lo cual es fundamental para la integridad de los datos.
- **Orden no relevante**: En el modelo relacional, el orden de las tuplas dentro de una relación no tiene relevancia. Es decir, los datos no dependen del orden en el que se almacenan, y la recuperación de datos no se ve afectada por el orden de las filas o columnas.
- **Atributos homogéneos**: Cada columna dentro de una relación debe contener valores del mismo tipo de datos o dominio. Por ejemplo, si una columna almacena fechas, todos los valores en esa columna deben ser fechas válidas dentro del dominio definido.

3.2.1. Atributos y dominio de los atributos.

Los **atributos** son las columnas de una relación y representan las diferentes propiedades o características de las entidades que se modelan en la base de datos. Cada atributo en una relación tiene un nombre único y está asociado con un **dominio**, que define el conjunto de valores permitidos para ese atributo.

Dominio:

- Un **dominio** es un conjunto de valores válidos para un atributo particular. El dominio define el tipo de dato que puede ser almacenado en un atributo, como números enteros, cadenas de texto, fechas, etc. Además, un dominio puede incluir restricciones adicionales, como un rango de valores permitidos.
- **Ejemplo de dominio**: Si el atributo `Edad` en una tabla `Empleado` tiene un dominio definido como números enteros entre 18 y 65, entonces el dominio restringe que los valores almacenados en la columna `Edad` deben estar dentro de este rango.

Atributos:

- Los atributos son los elementos que describen las propiedades de las entidades dentro de una relación. Cada atributo debe estar claramente definido con su nombre y dominio. Además, los atributos juegan un papel crucial en la identificación de las relaciones y en la imposición de restricciones de integridad (como las claves primarias y foráneas).
- **Ejemplo de atributo**: En una relación `Estudiante`, los atributos pueden incluir `ID_Estudiante`, `Nombre`, `Fecha de Nacimiento`, y `Carrera`. Cada uno de estos atributos describe una característica específica de los estudiantes.

3.2.1. Tupla, grado y cardinalidad.

Una **tupla** es una fila o registro individual en una relación. Cada tupla es una instancia específica de los datos que están organizados en la tabla y corresponde a una entidad particular. Las tuplas contienen valores para cada uno de los atributos definidos en la relación.

Tupla:

- Una tupla es una lista ordenada de valores, donde cada valor pertenece a un atributo específico de la relación. Las tuplas representan una entidad o un objeto en el mundo real, capturando todos los detalles relevantes que se han modelado en la tabla.
- **Ejemplo de tupla**: Consideremos una relación `Empleado` con los atributos `ID_Empleado`, `Nombre`, `Cargo`, y `Salario`. Una tupla de esta relación podría ser `(101, 'Juan Pérez', 'Gerente', 50000)`.

Grado:

- El **grado** de una relación es el número de atributos que contiene. Es decir, el grado es el número de columnas o campos en la tabla.
- **Ejemplo de grado**: Si una relación `Producto` tiene los atributos `ID_Producto`, `Nombre`, `Precio`, y `Stock`, el grado de esta relación sería 4, ya que contiene cuatro atributos.

Cardinalidad:

- La **cardinalidad** de una relación es el número de tuplas que contiene. Es decir, la cardinalidad es el número de filas o registros en la tabla.
- **Ejemplo de cardinalidad**: Si la tabla `Empleado` contiene 200 registros, entonces la cardinalidad de la relación `Empleado` es 200.

3.2.1. Relaciones y tablas.

En el contexto del modelo relacional, el término **relación** se usa de manera intercambiable con el término **tabla**. Cada relación en el modelo relacional se implementa físicamente como una tabla en la base de datos. Estas tablas son la estructura principal a través de la cual se almacenan los datos, y están organizadas en filas (tuplas) y columnas (atributos).

Relaciones como Tablas:

- Una **tabla** en una base de datos relacional es la representación física de una relación. Cada tabla almacena datos sobre un tipo específico de entidad o concepto, como `Clientes`, `Pedidos`, o `Productos`. Las tablas se pueden conectar entre sí mediante relaciones, utilizando claves foráneas para vincular datos en diferentes tablas.
- **Ejemplo**: Consideremos una base de datos de una tienda en línea. Se pueden tener tablas como `Clientes`, `Productos`, `Pedidos`, y `Detalles de Pedido`. Cada una de estas tablas es una relación que almacena información sobre un aspecto diferente del negocio. Las tablas pueden estar relacionadas entre sí, por ejemplo, `Pedidos` puede estar relacionado con `Clientes` a través de una clave foránea que conecta el `ID_Cliente` en la tabla `Pedidos` con el `ID_Cliente` en la tabla `Clientes`.

Conclusión

La **estructura del modelo relacional** se basa en conceptos matemáticos y lógicos que permiten organizar los datos de manera eficiente y consistente. Al entender el concepto de relación, junto con las propiedades de las relaciones, los atributos y dominios, las tuplas, el grado, la cardinalidad y la relación entre tablas, los diseñadores de bases de datos pueden crear sistemas que no solo almacenen datos de manera segura, sino que también los hagan accesibles y útiles para las aplicaciones que dependen de ellos. Estas estructuras forman la base sobre la cual se construyen las bases de datos relacionales, permitiendo a las organizaciones manejar grandes volúmenes de datos con precisión y eficacia.

3.3. Claves en el modelo relacional:

En el modelo relacional, las **claves** son fundamentales para garantizar la integridad y la coherencia de los datos. Las claves son atributos o conjuntos de atributos que se utilizan para identificar de manera única las tuplas (filas) en una relación (tabla). Estas claves no solo permiten identificar registros de forma única, sino que también son esenciales para establecer y mantener relaciones entre diferentes tablas dentro de la base de datos. A continuación, se describen los diferentes tipos de claves en el modelo relacional, incluyendo claves candidatas, claves primarias, claves alternativas y claves ajenas.

3.3.1. Claves candidatas.

Una **clave candidata** es un atributo o un conjunto de atributos que puede identificar de manera única a cada tupla en una relación. En otras palabras, es una clave que tiene el potencial de ser elegida como la clave primaria, pero que aún no ha sido designada como tal. Una tabla puede tener varias claves candidatas, pero solo una de ellas se seleccionará como la clave primaria.

Características de las Claves Candidatas:

- **Unicidad**: Cada valor de una clave candidata debe ser único dentro de la relación, lo que garantiza que ninguna tupla tenga el mismo valor en los atributos de la clave candidata.
- **No nulidad**: Ningún valor de la clave candidata puede ser nulo, ya que una clave candidata debe ser capaz de identificar cualquier tupla dentro de la relación.

Ejemplo de Clave Candidata:

- En una tabla `Empleado`, los atributos `ID_Empleado` y `Número de Seguridad Social` podrían ser ambos claves candidatas porque ambos pueden identificar de manera única a cada empleado. Sin embargo, solo uno de estos atributos será seleccionado como la clave primaria.

3.3.2. Claves primarias.

La **clave primaria** es la clave candidata que se selecciona para identificar de manera única a cada tupla en una relación. Es el identificador principal para los registros en una tabla, y no puede haber dos registros en la tabla con el mismo valor de clave primaria. Además, una clave primaria no puede contener valores nulos.

Características de la Clave Primaria:

- **Unicidad**: Al igual que las claves candidatas, la clave primaria debe ser única en toda la tabla.
- **No nulidad**: La clave primaria no puede tener un valor nulo, ya que debe ser capaz de identificar de manera única a cada tupla.
- **Estabilidad**: Idealmente, una clave primaria no debería cambiar con frecuencia, ya que esto afectaría la integridad de la base de datos, especialmente si otras tablas dependen de esta clave como clave foránea.

Ejemplo de Clave Primaria:

- En una tabla `Cliente`, el atributo `ID_Cliente` podría ser la clave primaria, asegurando que cada cliente tenga un identificador único que no se repita en ningún otro registro de la tabla.

Clave Primaria Compuesta

En algunos casos, una única columna puede no ser suficiente para identificar de manera única cada registro en una tabla. En estas situaciones, se utiliza una **clave primaria compuesta**, que es una combinación de dos o más columnas que, juntas, forman la clave primaria.

Consideraciones al Elegir una Clave Primaria

Al elegir una clave primaria, es importante considerar varios factores:

1. **Unicidad**: La columna seleccionada debe garantizar la unicidad de los registros.
2. **No Nulidad**: La clave primaria no debe permitir valores nulos.
3. **Estabilidad**: Idealmente, la clave primaria debe ser un valor que no cambie con el tiempo. Un valor que cambia frecuentemente puede complicar la gestión de la base de datos.
4. **Simplicidad**: Si es posible, es mejor elegir una clave primaria simple (una sola columna) en lugar de una clave compuesta, ya que las claves simples suelen ser más fáciles de gestionar y optimizar.

3.3.3. Claves alternativas.

Las **claves alternativas** son las claves candidatas que no se seleccionaron como clave primaria. Aunque no se usan para identificar de manera principal a las tuplas, estas claves aún conservan las propiedades de unicidad y no nulidad. Las claves alternativas pueden ser utilizadas en otras operaciones dentro de la base de datos, como consultas que requieran identificar registros a través de un atributo diferente de la clave primaria.

Características de las Claves Alternativas:

- **Unicidad**: Al igual que la clave primaria, una clave alternativa debe contener valores únicos dentro de la tabla.
- **No nulidad**: Una clave alternativa tampoco puede contener valores nulos.

Ejemplo de Clave Alternativa:

- Siguiendo con el ejemplo anterior, si `ID_Cliente` es la clave primaria en la tabla `Cliente`, entonces `Número de Seguridad Social` podría ser una clave alternativa, ya que también identifica de manera única a cada cliente.

3.3.4. Claves ajenas.

Una **clave ajena** (o **clave foránea**) es un atributo o conjunto de atributos en una tabla que se utiliza para establecer y reforzar una relación entre dos tablas. La clave ajena en una tabla apunta a la clave primaria de otra tabla, creando una relación entre las dos tablas. Las claves ajenas son fundamentales para mantener la **integridad referencial** en la base de datos, asegurando que las relaciones entre tablas permanezcan consistentes.

Características de las Claves Ajenas:

- **Relación entre tablas**: Las claves ajenas permiten que las tablas se relacionen entre sí. Esto es especialmente útil para establecer relaciones de uno a muchos o muchos a muchos entre entidades diferentes.
- **Integridad referencial**: Una clave ajena debe tener un valor que coincida con un valor existente en la clave primaria de la tabla a la que hace referencia. Si no se encuentra un valor coincidente, se produce una violación de la integridad referencial.

Ejemplo de Clave Ajena:

- En una base de datos de una tienda en línea, se podría tener una tabla `Pedidos` y una tabla `Clientes`. La tabla `Pedidos` podría contener un atributo `ID_Cliente` que actúa como clave ajena, refiriéndose a `ID_Cliente` en la tabla `Clientes`. Esto establece una relación entre los pedidos y los clientes, garantizando que cada pedido esté asociado a un cliente válido.

Comportamientos Asociados a Claves Ajenas

Cuando se define una clave ajena, es posible especificar cómo debe comportarse la base de datos ante operaciones como la eliminación o actualización de registros en la tabla referenciada:

1. **ON DELETE CASCADE**: Si se elimina un registro en la tabla referenciada, todos los registros relacionados en la tabla con la clave ajena también se eliminan automáticamente.
2. **ON DELETE SET NULL**: Si se elimina un registro en la tabla referenciada, los valores de la clave ajena en la tabla relacionada se establecen en `NULL`, manteniendo el registro pero eliminando la referencia.
3. **ON DELETE RESTRICT**: Impide la eliminación de un registro en la tabla referenciada si hay registros relacionados en la tabla con la clave ajena, evitando la ruptura de la integridad referencial.
4. **ON UPDATE CASCADE**: Si se actualiza el valor de la clave primaria en la tabla referenciada, los valores correspondientes en la tabla con la clave ajena se actualizan automáticamente para reflejar el nuevo valor.
5. **ON UPDATE SET NULL**: Si se actualiza el valor de la clave primaria en la tabla referenciada, los valores de la clave ajena en la tabla relacionada se establecen en `NULL`, eliminando la referencia al valor actualizado.
6. **ON UPDATE RESTRICT**: Impide la actualización del valor de la clave primaria en la tabla referenciada si hay registros relacionados en la tabla con la clave ajena, manteniendo la integridad referencial.
7. **ON DELETE NO ACTION / ON UPDATE NO ACTION**: No realiza ninguna acción automática. Si se intenta eliminar o actualizar un registro en la tabla referenciada que está siendo referenciado por una clave ajena, la operación será rechazada y se lanzará un error, asegurando la integridad referencial.

Estos comportamientos permiten controlar cómo se manejan las relaciones entre tablas cuando se realizan modificaciones en los datos, asegurando la integridad y coherencia de la base de datos tanto en eliminaciones como en actualizaciones.

Conclusión

Las claves en el modelo relacional son fundamentales para garantizar la integridad, la coherencia y la eficacia de una base de datos. Cada tipo de clave —candidata, primaria, alternativa y ajena— desempeña un papel específico en la organización y la gestión de los datos. Las claves candidatas ofrecen opciones para la identificación única de registros, de las cuales se elige una clave primaria para identificar de manera principal a cada registro. Las claves alternativas, aunque no seleccionadas como primarias, siguen siendo cruciales para operaciones específicas, mientras que las claves ajenas aseguran la integridad referencial al conectar y relacionar diferentes tablas dentro del sistema de bases de datos. Comprender y utilizar correctamente estos tipos de claves es esencial para el diseño eficiente y funcional de una base de datos relacional.

3.5. Restricciones de integridad:

Las **restricciones de integridad** son reglas fundamentales en el modelo relacional que garantizan la exactitud, consistencia y validez de los datos almacenados en una base de datos. Estas restricciones aseguran que los datos se mantengan coherentes con las reglas del sistema y las expectativas del negocio, evitando la entrada de datos inválidos o inconsistentes. A continuación, se detallan tres tipos cruciales de restricciones de integridad: el manejo de valores nulos, la integridad de las entidades y la integridad referencial.

3.5.1. Valor “Null” en el modelo.

En el modelo relacional, el valor **NULL** representa la ausencia de un valor en un atributo específico de una tupla. Este valor no es lo mismo que cero, una cadena vacía o cualquier otro valor definido; es simplemente la indicación de que el valor es desconocido o no aplicable.

Importancia del Valor NULL:

1. **Representación de Datos Incompletos o Desconocidos**: El valor NULL se utiliza cuando no se dispone de un valor específico para un atributo en un registro. Esto es común en situaciones donde la información aún no se ha recopilado o no es relevante para un registro en particular.
2. **Evitar Asunciones Incorrectas**: Al usar NULL, se evita la confusión que podría surgir al asignar valores como cero o una cadena vacía para representar la falta de datos. Esto ayuda a mantener la precisión en la interpretación de los datos.

Implicaciones del Uso de NULL:

1. **Operaciones con NULL**: Las operaciones aritméticas o de comparación con valores NULL suelen devolver NULL. Por ejemplo, sumar cualquier número a NULL da como resultado NULL, lo que refleja la incertidumbre de la operación.
2. **Condiciones WHERE y NULL**: Las consultas que utilizan la cláusula `WHERE` deben manejar correctamente los valores NULL. Por ejemplo, para seleccionar filas donde un atributo es NULL, se debe utilizar la condición `IS NULL` en lugar de una comparación estándar. **Ejemplo:** SELECT * FROM Empleados WHERE Fecha_Salida IS NULL;

3.5.2. Integridad de las entidades.

La **integridad de las entidades** es una restricción fundamental que asegura que cada tupla (fila) en una tabla pueda ser identificada de manera única. Esto se logra principalmente mediante la definición de una clave primaria para cada tabla en la base de datos.

Principios de la Integridad de las Entidades:

1. **Clave Primaria No Nula**: La restricción de integridad de las entidades requiere que la clave primaria de una tabla no contenga valores NULL. Esto garantiza que cada registro en la tabla tenga un identificador único y válido.
2. **Unicidad de la Clave Primaria**: La clave primaria debe ser única para cada registro en la tabla, evitando duplicados y asegurando que cada entidad esté representada de manera única en la base de datos.

3.5.3. Integridad referencial.

La **integridad referencial** es una restricción que garantiza que las relaciones entre tablas en una base de datos se mantengan coherentes. Esta restricción se asegura de que las claves ajenas (foreign keys) en una tabla hagan referencia a claves primarias válidas en otra tabla, evitando referencias rotas o inconsistentes.

Principios de la Integridad Referencial:

1. **Referencias Válidas**: Una clave ajena en una tabla debe referirse a un valor de clave primaria existente en la tabla relacionada. Esto evita que se inserten registros con referencias a entidades inexistentes.
2. **Acciones Definidas para Modificaciones**: Cuando se modifica o elimina un registro en la tabla referenciada (aquella que contiene la clave primaria), se pueden definir acciones específicas para manejar la clave ajena en la tabla relacionada. Estas acciones incluyen CASCADE, SET NULL, RESTRICT y NO ACTION, entre otras.

Conclusión

Las **restricciones de integridad** son esenciales para mantener la precisión, coherencia y validez de los datos en una base de datos relacional. El manejo adecuado de valores NULL, la garantía de integridad de las entidades mediante claves primarias, y la preservación de la integridad referencial a través de claves ajenas, aseguran que los datos almacenados se mantengan consistentes y reflejen correctamente las relaciones del mundo real que representan. Estas restricciones son fundamentales para el funcionamiento correcto y fiable de cualquier sistema de bases de datos.

3.6. Teoría de la normalización:

La **normalización** es un proceso crítico en el diseño de bases de datos relacionales, cuyo objetivo es organizar los datos de manera que se minimicen las redundancias y se eliminen las anomalías en la inserción, actualización y eliminación de datos. Este proceso se lleva a cabo a través de una serie de etapas conocidas como **formas normales**. Cada forma normal aborda problemas específicos en la estructura de la base de datos, mejorando la consistencia y eficiencia del almacenamiento de datos. A continuación, se detallan las formas normales desde la Primera Forma Normal (1FN) hasta las más avanzadas, como la Cuarta Forma Normal (4FN) y la Quinta Forma Normal (5FN), junto con ejemplos ilustrativos.

3.6.1. El proceso de normalización. Tipos de dependencias funcionales.

El **proceso de normalización** implica la descomposición de tablas grandes y complejas en tablas más pequeñas y manejables, sin perder información. El objetivo es eliminar la redundancia de datos y asegurar que las dependencias entre los atributos se mantengan correctamente, mejorando así la integridad y consistencia de la base de datos.

Las **dependencias funcionales** son un concepto clave en la normalización. Una dependencia funcional ocurre cuando un atributo depende de otro atributo o conjunto de atributos. Por ejemplo, si en una tabla `Empleado`, el `ID_Empleado` determina el `Nombre`, entonces `Nombre` depende funcionalmente de `ID_Empleado`.

Tipos de Dependencias Funcionales:

- **Dependencia Funcional Completa**: Un atributo depende completamente de la clave primaria, sin depender de solo una parte de ella.
- **Dependencia Funcional Parcial**: Un atributo depende solo de una parte de una clave primaria compuesta, lo cual se corrige en la Segunda Forma Normal (2FN).
- **Dependencia Transitiva**: Un atributo depende de otro atributo que, a su vez, depende de la clave primaria, y se elimina en la Tercera Forma Normal (3FN).

3.6.2. Primera forma normal (1FN).

La **Primera Forma Normal (1FN)** es el nivel básico de normalización que una tabla debe cumplir. Para que una tabla esté en 1FN, debe cumplir con los siguientes requisitos:

1. **Atributos Atómicos**: Todos los atributos deben contener valores atómicos, es decir, indivisibles. No se permiten grupos repetitivos o atributos que contengan listas de valores.

2. **No Se Permiten Valores Repetidos**: Cada celda en la tabla debe contener un único valor. No se permiten múltiples valores en una sola celda.
3. **Existencia de una Clave Primaria**: Debe existir una clave primaria que identifique de manera única cada registro en la tabla.

Ejemplo:

Supongamos una tabla `Pedidos` con las siguientes columnas: `ID_Pedido`, `Cliente`, `Productos`.

ID_Pedido	Cliente	Productos
001	Juan	Laptop, Teclado
002	Ana	Monitor, Mouse

Esta tabla no está en 1FN porque la columna `Productos` contiene múltiples valores. Para normalizarla en 1FN, descomponemos los productos en filas separadas:

ID_Pedido	Cliente	Producto
001	Juan	Laptop
001	Juan	Teclado
002	Ana	Monitor
002	Ana	Mouse

3.6.3. Segunda forma normal (2FN).

Una tabla está en **Segunda Forma Normal (2FN)** si cumple con los siguientes criterios:

1. **Cumple con 1FN**: La tabla ya debe estar en 1FN.
2. **No Hay Dependencias Parciales**: Todos los atributos no clave deben depender completamente de la clave primaria. Es decir, no debe haber atributos que dependan solo de una parte de una clave primaria compuesta.

Ejemplo:

Consideremos una tabla `Pedidos` con las columnas `ID_Pedido`, `Producto`, `Precio_Unitario`, `Cantidad`, `Nombre_Cliente`.

ID_Pedido	Producto	Precio_Unitario	Cantidad	Nombre_Cliente
001	Laptop	1000	2	Juan
002	Teclado	50	1	Ana

En esta tabla, `Nombre_Cliente` depende solo de `ID_Pedido`, no de toda la clave compuesta (si consideramos que `ID_Pedido` y `Producto` forman la clave compuesta). Para llevar esta tabla a 2FN, debemos separar la información relacionada con el cliente en otra tabla.

Tabla Pedidos:

ID_Pedido	Producto	Precio_Unitario	Cantidad
001	Laptop	1000	2
002	Teclado	50	1

Tabla Clientes:

ID_Pedido	Nombre_Cliente
001	Juan
002	Ana

3.6.4. Tercera forma normal (3FN).

Una tabla está en **Tercera Forma Normal (3FN)** si cumple con los siguientes criterios:

1. **Cumple con 2FN**: La tabla ya debe estar en 2FN.
2. **No Hay Dependencias Transitivas**: Los atributos no clave deben depender directamente de la clave primaria y no de otros atributos no clave.

Ejemplo:

Consideremos una tabla `Pedidos` con las columnas `ID_Pedido`, `ID_Cliente`, `Nombre_Cliente`, `Ciudad_Cliente`.

ID_Pedido	ID_Cliente	Nombre_Cliente	Ciudad_Cliente
001	100	Juan	Madrid
002	101	Ana	Barcelona

Aquí, `Ciudad_Cliente` depende de `Nombre_Cliente`, que a su vez depende de `ID_Cliente`. Esto es una dependencia transitiva, que se debe eliminar para cumplir con 3FN.

Tabla Pedidos:

ID_Pedido	ID_Cliente
001	100
002	101

Tabla Clientes:

ID_Cliente	Nombre_Cliente	Ciudad_Cliente
100	Juan	Madrid
101	Ana	Barcelona

3.6.5. Otras formas normales (4FN, 5FN).

Las formas normales más avanzadas, como la **Cuarta Forma Normal (4FN)** y la **Quinta Forma Normal (5FN)**, abordan problemas más complejos que pueden surgir en el diseño de bases de datos con múltiples relaciones y dependencias.

Cuarta Forma Normal (4FN):

- Una tabla está en 4FN si, además de estar en 3FN, no tiene **dependencias multivalor**. Las dependencias multivalor ocurren cuando un atributo depende de más de un valor de otro atributo en la misma tabla, lo que puede llevar a redundancias y anomalías.

Ejemplo de 4FN:

Supongamos que una tabla `Proyectos` tiene los atributos `ID_Proyecto`, `Nombre_Empleado`, y `Habilidad`:

ID_Proyecto	Nombre_Empleado	Habilidad
001	Juan	Programación
001	Juan	Gestión
002	Ana	Diseño

Aquí, `Nombre_Empleado` tiene múltiples habilidades asociadas dentro del mismo proyecto, lo que crea una dependencia multivalor. Para normalizar en 4FN, debemos descomponer la tabla en dos:

Tabla Empleados_Proyectos:

ID_Proyecto	Nombre_Empleado
001	Juan
002	Ana

Tabla Empleados_Habilidades:

Nombre_Empleado	Habilidad
Juan	Programación
Juan	Gestión
Ana	Diseño

Quinta Forma Normal (5FN):

- Una tabla está en 5FN si está en 4FN y no contiene **dependencias de unión**. Las dependencias de unión se refieren a situaciones donde ciertos atributos de una tabla dependen de la combinación de otros atributos y no de los atributos individualmente. Esto ocurre en tablas con relaciones complejas y se resuelve descomponiendo la tabla en varias tablas más pequeñas sin perder la información de las combinaciones.

Ejemplo de 5FN:

Supongamos una tabla `Proyectos_Empleados_Clientes` con los atributos `ID_Proyecto`, `Nombre_Empleado`, y `ID_Cliente`:

ID_Proyecto	Nombre_Empleado	ID_Cliente
001	Juan	C001
001	Ana	C002
002	Juan	C001

Aquí, la combinación de `ID_Proyecto`, `Nombre_Empleado`, y `ID_Cliente` puede ser descompuesta en tres tablas separadas para eliminar dependencias de unión:

Tabla Proyectos_Empleados:

ID_Proyecto	Nombre_Empleado
001	Juan
001	Ana
002	Juan

Tabla Proyectos_Clientes:

ID_Proyecto	ID_Cliente
001	C001
001	C002
002	C001

Tabla Empleados_Clientes:

Nombre_Empleado	ID_Cliente
Juan	C001
Ana	C002

3.6.6. Desnormalización. Razones para la desnormalización.

Desnormalización es el proceso opuesto a la normalización. Implica la combinación de tablas que fueron divididas durante la normalización para mejorar el rendimiento de las consultas o simplificar la estructura de la base de datos en ciertos casos.

Razones para la Desnormalización:

1. **Mejora del Rendimiento**: En bases de datos muy grandes o con un alto volumen de transacciones, las consultas que involucran múltiples tablas normalizadas pueden ser lentas. La desnormalización puede reducir la cantidad de uniones (joins) necesarias, acelerando las consultas.
2. **Simplificación del Diseño**: A veces, las necesidades del negocio requieren un acceso rápido y sencillo a los datos, sin la complejidad que las tablas altamente normalizadas pueden introducir.
3. **Reducción de la Complejidad de Consultas**: En sistemas donde la prioridad es la velocidad de acceso a la información y la simplicidad en la gestión, la desnormalización puede ser útil para reducir la complejidad de las consultas SQL.

Ejemplo de Desnormalización:

Si tenemos una base de datos donde las tablas `Clientes` y `Pedidos` están separadas pero se consultan juntas con frecuencia, podríamos combinarlas en una sola tabla desnormalizada para evitar realizar joins constantemente.

Tabla Desnormalizada:

ID_Pedido	ID_Cliente	Nombre_Cliente	Producto	Cantidad
001	100	Juan	Laptop	2
002	101	Ana	Teclado	1

En este caso, aunque hay una redundancia de información (el `Nombre_Cliente` se repite), el acceso a los datos podría ser más rápido para ciertas consultas.

Conclusión

La **normalización** es un proceso esencial en el diseño de bases de datos que ayuda a minimizar la redundancia y a evitar anomalías en las operaciones de la base de datos. Sin embargo, en algunos casos, la **desnormalización** puede ser una estrategia válida para mejorar el rendimiento o simplificar el diseño. Comprender las diferentes formas normales y saber cuándo aplicarlas o cuándo desnormalizar es clave para crear una base de datos que sea tanto eficiente como eficaz en cumplir con los requisitos del negocio.