ACCESO A DATOS

UNIDAD 2. BASES DE DATOS RELACIONALES

ÍNDICE DE CONTENIDOS

INTRODUCCIÓN

Las bases de datos relacionales son una de las tecnologías más utilizadas en el ámbito de la gestión de datos, ofreciendo una forma organizada y estructurada de almacenar información. Se fundamentan en el modelo relacional, que organiza los datos en tablas compuestas por filas y columnas. **Cada tabla representa una entidad**, donde cada fila o registro almacena información sobre un único elemento de esa entidad, y cada columna contiene un atributo o característica de los registros.

Una de las características más destacadas de las bases de datos relacionales es la posibilidad de establecer relaciones entre diferentes tablas. Estas relaciones se crean mediante el uso de claves. La clave primaria es un campo que se establece como identificador único para los registros dentro de una tabla, asegurando que no se repita ningún valor en esta columna. A su vez, las claves foráneas son campos que se utilizan para vincular dos tablas distintas, estableciendo una relación a través de la clave primaria de una de ellas. **Este tipo de organización es crucial para mantener la integridad y consistencia de los datos**. Las relaciones entre tablas permiten que la base de datos escale de manera efectiva, soportando la adición de nuevos registros y entidades sin perder la calidad de la información.

El lenguaje de definición de datos (DDL) se usa para definir la estructura de la base de datos. Con el DDL se pueden crear y modificar tablas, así como establecer las propiedades de los campos, como tipos de datos y restricciones.

Además de la creación de tablas, el DDL también incluye comandos para alterar la estructura existente de la base de datos, como añadir nuevas columnas o eliminar aquellas que ya no son necesarias. Estos procesos de modificación son importantes a medida que cambian las necesidades de datos de una organización.

El lenguaje de manipulación de datos (DML) permite a los usuarios realizar operaciones sobre los datos almacenados. Entre sus funciones está insertar nuevos registros, actualizar la información existente y eliminar datos obsoletos.

Utilizar el DML correctamente ayuda a mantener actualizada la información en la base de datos y para responder a las necesidades cambiantes de la organización. La correcta estructuración de las órdenes DML permite que las consultas sean más eficientes y optimizadas, aumentando así la rapidez y efectividad con la que se pueden obtener los datos deseados.

La correcta gestión de estas claves es especialmente importante en bases de datos que deben escalar. A medida que se incrementa el volumen de datos y se añaden nuevas entidades, una estructura de bases de datos bien diseñada permite realizar estas operaciones sin perder la integridad referencial. Esto significa que, sin importar cuántos datos se añadan o modifiquen, las relaciones entre las tablas se mantendrán correctas, lo que es esencial para cualquier aplicación que necesite gestionar una gran cantidad de información a lo largo del tiempo.

1. INTRODUCCIÓN A LAS BASES DE DATOS RELACIONALES

Las bases de datos relacionales son sistemas diseñados para almacenar, gestionar y recuperar información de forma estructurada y eficiente. Este tipo de bases de datos se basa en el modelo relacional, introducido por Edgar F. Codd en la década de 1970, y organiza los datos en tablas. Cada tabla, compuesta por filas y columnas, representa una entidad y sus atributos, lo que facilita la comprensión y el acceso a la información.

Cada tabla dentro de una base de datos relacional se identifica por una clave primaria, que asegura la singularidad de cada registro. Esta clave permite establecer relaciones con otras tablas, creando así un conjunto de datos interconectados que reflejan relaciones del mundo real. Estas relaciones pueden ser de uno a uno, uno a muchos o muchos a muchos, según cómo se conectan los datos.

El Lenguaje de Consulta Estructurado (SQL) es utilizado para interactuar con bases de datos relacionales. SQL permite realizar diversas operaciones, como insertar, actualizar, eliminar y consultar datos. La capacidad para manejar grandes volúmenes de información y la adaptabilidad a diferentes tipos de datos contribuyen a su popularidad en entornos empresariales y diferentes aplicaciones.

Las bases de datos relacionales encuentran aplicación en diversos sectores, como el comercio electrónico y la gestión de recursos humanos, destacándose por su habilidad para mantener la coherencia y la integridad de los datos en sistemas complejos.

1.1. CONCEPTOS BÁSICOS Y DEFINICIONES

Las bases de datos relacionales organizan la información en una estructura tabular. Estas estructuras, llamadas tablas, son fundamentales para el almacenamiento y la gestión de datos. Las tablas permiten una relación eficiente entre datos y simplifican las operaciones de consulta y modificación.

Una tabla se compone de campos (columnas) y registros (filas). **Cada columna representa un atributo específico de los datos, y cada fila contiene un conjunto de valores que corresponden a un registro único**. Por ejemplo, en una tabla de “Clientes”, las columnas pueden ser “ClienteID”, “Nombre”, “Apellido”, “Teléfono” y “Email”. Cada fila en esta tabla representa un cliente concreto con su información correspondiente. Esta organización permite a los usuarios realizar consultas complejas sobre la información de manera sencilla.

La **clave primaria** es importante para identificar de forma única cada registro en la tabla. Esta debe ser un valor único para cada fila. Si utilizamos “ClienteID” como clave primaria en la tabla de “Clientes”, se garantiza que no haya duplicados. Este procedimiento evita problemas de repetición y asegura la integridad de los datos.

Las **claves foráneas** también permiten establecer relaciones entre diferentes tablas. La clave foránea se utiliza para vincular una tabla con otra, facilitando la normalización de los datos. Por

ejemplo, en un sistema de gestión de pedidos, la tabla “Pedidos” puede incluir una columna “ClienteID” que actúa como clave foránea, referenciando la clave primaria en la tabla “Clientes”. Esto significa que cada pedido puede estar asociado a un cliente específico. Al consultar la base de datos para obtener información sobre pedidos, se puede unir la tabla de “Pedidos” con la tabla de “Clientes” para obtener una visión más completa.

> La normalización se refiere al proceso de estructurar los datos de manera que se minimice la repetición y la dependencia.

Existen diversas etapas de normalización que definen las reglas para lograr un diseño de datos organizado. Por ejemplo, al **aplicar la primera etapa de normalización, es esencial eliminar grupos repetitivos en una tabla**. Si inicialmente tenemos una tabla de “Estudiantes” que incluye una columna para las materias donde se permiten múltiples inscripciones, no estaría cumpliendo con la primera etapa y sería necesario modificarla para que cada inscripción sea una fila única.

La segunda etapa de normalización requiere que todos los atributos de la tabla dependan exclusivamente de la clave primaria. Por ejemplo, si en una tabla de “Facturas” se almacenan tanto detalles de las facturas como información del cliente, sería conveniente dividir esa información en dos tablas separadas. Esto resultaría en una tabla “Clientes” y otra “Facturas”, donde las facturas sólo contendrían la clave foránea del cliente, manteniendo la información organizada y evitando dependencias indeseadas.

La tercera etapa de normalización busca eliminar dependencias transitivas en la tabla. **Esta etapa establece que, si un atributo no depende directamente de la clave primaria, debe ser trasladado a otra tabla**. *Por ejemplo, en una tabla de “Productos” que contenga información sobre el proveedor junto con los detalles del producto, si el nombre del proveedor no es un atributo que dependa del producto, entonces debería ser llevado a una tabla separada de “Proveedores”.*

El lenguaje SQL (Structured Query Language) permite gestionar y consultar datos de bases de datos relacionales. Esta herramienta proporciona una sintaxis clara para realizar diversas operaciones. El comando `INSERT` se utiliza para añadir nuevos registros a una tabla. Si se quiere insertar un nuevo cliente en la tabla “Clientes”, se podría utilizar el siguiente comando:

```
INSERT INTO Clientes (ClienteID, Nombre, Apellido, Teléfono, Email)
       VALUES (1, 'Juan', 'Pérez', '123456789', 'juan.perez@example.com');
```

También es común actualizar registros utilizando el comando `UPDATE`. Si se necesita modificar el número de teléfono de un cliente, el comando podría ser:

```
UPDATE Clientes SET Teléfono = '987654321' WHERE ClienteID = 1;
```

Las operaciones de consulta se realizan a través del comando `SELECT`, que permite filtrar resultados, agregar condiciones y unir tablas. Por ejemplo, para obtener la lista de todos los clientes con pedidos, se podría usar la siguiente consulta:

```
SELECT c.Nombre, c.Apellido, p.FechaPedido
FROM Clientes c
JOIN Pedidos p ON c.ClienteID = p.ClienteID;
```

Las transacciones representan un conjunto de operaciones que se ejecutan como una única unidad. **Deben cumplir con las propiedades ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad)**. La atomicidad asegura que todas las operaciones de una transacción se realicen con éxito; si una falla, todas se revierten. Por ejemplo, en un proceso de transferencia de dinero entre cuentas, es importante que el monto se reste de una cuenta y se añada a la otra simultáneamente. Si alguna de estas operaciones falla, la transacción completa no se debe ejecutar, evitando inconsistencias en la base de datos.

El aislamiento asegura que las transacciones concurrentes no afecten entre sí y, por último, la durabilidad garantiza que, una vez que se ha confirmado una transacción, sus cambios persisten en el sistema incluso si ocurren fallos. Estas propiedades son relevantes en sistemas que manejan información sensible, como el bancario.

Un caso práctico puede ser el diseño de un sistema de gestión para reservas de un hotel. En este sistema, se pueden crear tablas para “Clientes”, “Habitaciones” y “Reservas”. La tabla “Clientes” contendría información personal, la tabla “Habitaciones” incluiría detalles sobre los tipos de habitaciones, precios y disponibilidad, y la tabla “Reservas” asociaría a los clientes con las habitaciones reservadas mediante claves foráneas. Este enfoque permite realizar consultas efectivas para verificar la ocupación, gestionar las reservas y obtener información sobre los clientes de forma estructurada.

La creación de índices mejora el rendimiento de las consultas al permitir un acceso más rápido a los datos. Al crear un índice sobre columnas que se consultan frecuentemente, como “Nombre” en la tabla de “Clientes”, se reduce el tiempo requerido para ejecutar consultas que busquen por nombre. **Sin embargo, es necesario manejar con cuidado la creación de índices, ya que cada modificación de datos en la tabla también requiere la actualización del índice, lo que puede afectar el rendimiento en operaciones de inserción o modificación**.

La gestión adecuada de bases de datos relacionales es crucial en el entorno empresarial moderno. Las organizaciones dependen de estas estructuras para tomar decisiones informadas a partir de los datos extraídos de sus bases de datos. Con el auge de la analítica de datos y las tecnologías de *big data*, la comprensión de los conceptos y definiciones de bases de datos relacionales se torna cada vez más relevante en la formación de quienes trabajan en el desarrollo de aplicaciones y en la gestión de datos. La capacidad de manejar grandes volúmenes de información de manera efectiva y realizar análisis precisos se convierte en un activo valioso para cualquier organización.

1.2. COMPONENTES DE UNA BASE DE DATOS RELACIONAL

Las bases de datos relacionales constituyen una parte central en la gestión de información en diversos sectores. Para entender su funcionamiento y aplicabilidad, es necesario examinar en detalle cada uno de sus componentes y cómo se relacionan entre sí.

Tablas

Las tablas son estructuras que organizan los datos (y que los persisten). Están compuestas de filas y columnas, donde cada fila representa un registro único y **cada columna corresponde a un atributo del registro (columna = campo = atributo)**. Por ejemplo, se puede tener una tabla llamada "Productos" que contenga columnas como "ID_Producto", "Nombre_Producto", "Precio" y "Cantidad_Disponible". Cada fila reflejaría un producto específico junto con su información asociada.

Las tablas permiten almacenar datos mediante un diseño que reduce la redundancia y mejora la integridad. En la práctica, una tienda en línea podría incluir diversas tablas, como "Clientes", "Productos", "Pedidos" y "Detalles_Pedidos", cada una diseñada para interactuar entre sí sin duplicar información innecesariamente.

Relaciones

Las relaciones en una base de datos relacional definen la conexión entre las distintas tablas. Este aspecto es importante para mantener la consistencia de los datos. Existen diferentes tipos de relaciones:

- **Relación Uno a Uno**: Aquí, un registro de una tabla está vinculado a un registro en otra tabla. Por ejemplo, en una base de datos de recursos humanos, puede existir una tabla "Empleados" y otra "Direcciones", donde cada empleado tiene una sola dirección registrada.
- **Relación Uno a Muchos**: Este tipo de relación es común en el modelado de datos. Un registro en la tabla "A" puede estar asociado con múltiples registros en la tabla "B". Por ejemplo, en una base de datos que gestiona pedidos, un cliente en la tabla "Clientes" puede tener varios registros en la tabla "Pedidos". En este caso, el "ID_Cliente" en la tabla "Pedidos" actúa como una referencia que establece la conexión.
- **Relación Muchos a Muchos**: Aquí, múltiples registros en la tabla "A" pueden asociarse con múltiples registros en la tabla "B". Para gestionar esta relación, es necesario crear una tabla intermedia que contenga referencias de ambas tablas. Por ejemplo, en un sistema educativo, una tabla "Estudiantes" y otra "Cursos" pueden establecer una relación muchos a muchos mediante una tabla intermedia que registre "ID_Estudiante" e "ID_Curso".

Claves

Las claves son componentes que garantizan la integridad y la conexión entre los datos. Existen principalmente dos tipos:

- **Clave Primaria**: Este es un atributo o conjunto de atributos que identifica de forma única cada registro en una tabla. En la tabla "Productos", el "ID_Producto" podría ser la clave primaria, asegurando que no existan duplicados y permitiendo referencias claras desde otras tablas.
- **Clave Foránea**: Se utiliza como un campo en una tabla que hace referencia a la clave primaria de otra tabla, estableciendo conexiones entre los dos conjuntos de datos. En el ejemplo anterior, en la tabla "Detalles_Pedidos", el "ID_Producto" podría ser una referencia que vincule los detalles con la tabla "Productos". Esto permite realizar consultas que extraigan información sobre todas las instancias de un producto específico en pedidos.

Índices

Los índices son estructuras que mejoran la velocidad de las operaciones de búsqueda en las tablas. **Funcionan como punteros a los registros de la tabla**, facilitando un acceso más rápido a los datos solicitados. Por ejemplo, en una base de datos de clientes con millones de registros, crear un índice en la columna "Email" aceleraría la recuperación de información específica sobre un cliente.

En sistemas donde la eficiencia es prioritaria, el uso de índices puede reducir significativamente los tiempos de respuesta para consultas complejas, aunque este uso puede incrementar el tiempo necesario para insertar o actualizar datos, ya que los índices deben actualizarse cada vez que se modifica una tabla.

Vistas

Las vistas son consultas que se almacenan y permiten a los usuarios acceder a datos desde una o más tablas como si fueran una tabla única. Esto resulta útil para simplificar el acceso a datos complejos y para asegurar que los usuarios no necesiten conocer la estructura subyacente. Por ejemplo, una vista que reúna información relevante de las tablas "Clientes" y "Pedidos" podría mostrar solo los clientes activos con sus últimos pedidos.

Las vistas también pueden proporcionar un control de acceso a los datos, permitiendo restringir la visualización de información sensible y mostrando solo ciertos datos a los usuarios según sus permisos.

Procedimientos almacenados y funciones

Los procedimientos almacenados son conjuntos de instrucciones SQL que se almacenan en el servidor y pueden ejecutarse cuando sea necesario. Su principal ventaja radica en la reutilización del código y la optimización de operaciones repetitivas. Por ejemplo, un procedimiento almacenado podría calcular automáticamente el total de la facturación de un cliente a partir de la tabla "Pedidos"

y la tabla "Detalles_Pedidos". Cada vez que se necesite esta información, se puede ejecutar el procedimiento sin reescribir la consulta.

Las funciones son similares a los procedimientos almacenados, pero están diseñadas para devolver un valor y pueden utilizarse dentro de otras consultas. Por ejemplo, una función que calcule el importe total de un pedido podría facilitar las operaciones relacionadas con el manejo de pedidos en tiempo real.

Gestión de transacciones

La gestión de transacciones es relevante para mantener la integridad de los datos en situaciones donde múltiples operaciones dependen entre sí. Aplicando las propiedades de Atomicidad, Consistencia, Aislamiento y Durabilidad (ACID), se asegura que una serie de operaciones se completen exitosamente o se cancelen completamente en caso de un error. Por ejemplo, al procesar un pedido, es necesario restar el monto correspondiente del inventario y sumar el importe a las ventas. Ambas operaciones deben completarse; si una falla, la transacción completa debe revertirse para preservar la coherencia.

Seguridad

La seguridad en bases de datos relacionales incluye la implementación de mecanismos de autenticación y autorización que controlan el acceso a los datos. En un sistema de gestión de recursos humanos, el acceso a la tabla "Salarios" debería restringirse a personal del departamento de finanzas. Establecer roles de usuario y permisos específicos es importante para proteger la información sensible y cumplir con regulaciones sobre la protección de datos.

Cada uno de estos componentes desempeña un rol significativo en el funcionamiento de bases de datos relacionales, garantizando que los datos se gestionen de manera eficiente, segura y accesible en entornos profesionales. La correcta implementación y uso de estas estructuras permite optimizar los procesos de consulta y administración de datos, adaptándose a las variadas necesidades de diferentes organizaciones.

1.3. MODELADO DE DATOS RELACIONAL

El modelado de datos relacional se centra en cómo se organiza la información en una base de datos y sigue un enfoque sistemático que abarca diversas etapas de conceptualización y diseño. Este proceso implica la creación de un modelo entidad-relación (ER), que luego se transforma en un modelo relacional apto para su uso en sistemas de gestión de bases de datos.

Entre los modelos de datos, el modelo entidad-relación es ampliamente utilizado. Ayuda a identificar entidades relevantes en un sistema, sus atributos y las relaciones que existen entre ellas.

> Las entidades representan objetos o conceptos del mundo real que tienen relevancia para el sistema, mientras que los atributos (campos/columnas) son las características que describen a esas entidades.

Por ejemplo, en un sistema educativo, se pueden señalar entidades como “Estudiantes”, “Profesores”, “Asignaturas” y “Cursos”, cada una con atributos como “ID Estudiante”, “Nombre”, “Apellido”, “ID Profesor” y “Descripción de Asignatura”.

Durante el modelado, se especifican los tipos de relaciones que existen entre las entidades. Estas pueden ser uno a uno, uno a muchos y muchos a muchos. Por ejemplo, en el sistema educativo mencionado, puede haber una relación uno a muchos entre “Profesores” y “Asignaturas”, dado que un profesor puede impartir varias asignaturas, y cada asignatura tiene asignado un único profesor. Sin embargo, la interacción entre “Estudiantes” y “Asignaturas” puede ser muchos a muchos, ya que un estudiante puede inscribirse en diversas asignaturas, y cada asignatura puede tener múltiples estudiantes. **Para gestionar las relaciones muchos a muchos, se crea una tabla intermedia, conocida como tabla de unión**; en este caso, podría existir una tabla llamada “Inscripciones”, que contenga referencias a las claves primarias de “Estudiantes” y “Asignaturas”.

La siguiente fase en el modelado consiste en elaborar el diagrama ER, que representa gráficamente las entidades, sus atributos y las relaciones entre ellas. Este diagrama actúa como guía visual que facilita la comprensión del sistema. Por ejemplo, en una base de datos para una tienda de comercio electrónico, el diagrama podría incluir entidades como “Clientes”, “Productos”, “Pedidos” y “Pagos”, mostrando cómo estas entidades se vinculan entre sí, como que cada cliente puede realizar múltiples pedidos y cada pedido puede contener varios productos.

Cuando el diagrama ER está elaborado, se traducirá en un modelo relacional. Esto implica definir tablas para cada entidad y especificar las claves primarias y foráneas necesarias. **Las claves primarias son atributos que identifican de forma única cada entrada en una tabla**. En el caso de la tienda de comercio electrónico, la tabla “Productos” podría tener una clave primaria denominada “ID Producto”. La relación entre “Clientes” y “Pedidos” conlleva que la tabla de “Pedidos” tendrá una clave foránea llamada “ID Cliente”, que referenciará la tabla “Clientes”.

Para garantizar que la estructura de la base de datos esté optimizada, se aplican las reglas de normalización. Este proceso busca eliminar redundancias y mejorar la integridad de los datos dividiendo la información en diferentes tablas relacionadas.

- **La primera forma normal (1NF)** implica que cada campo en la tabla debe contener un valor atómico. Si en la tabla de “Pedidos” se guardaran varios productos como una lista en un solo campo, esto rompería la 1NF. Cada producto debería registrarse en filas separadas, posiblemente en una tabla denominada “Detalles de Pedido” que contenga referencias a los pedidos y los productos específicos.

- **La segunda forma normal (2NF)** exige que todos los atributos que no son clave dependan completamente de la clave primaria. Si en una tabla de "Pedidos" se incluye un campo que almacena el nombre del cliente sin una referencia a su clave primaria, esto podría generar dependencia no deseada. Lo correcto en este caso sería crear una relación con la tabla "Clientes", donde cada pedido pueda acceder a la información del cliente mediante la clave foránea correspondiente.
- **La tercera forma normal (3NF)** se centra en eliminar dependencias transitivas. Esto significa que, si un atributo depende de otro atributo que no es la clave primaria, debe separarse en una nueva tabla. Por ejemplo, si en la tabla "Productos" se almacena también la categoría del producto, lo adecuado sería crear una tabla "Categorías" y vincularla desde "Productos". Esto permite que la información sobre categorías esté centralizada y evita cambios repetitivos en múltiples filas.

En la práctica, muchas organizaciones optan por evitar un exceso de normalización cuando las operaciones de lectura son más frecuentes que las de escritura[1][i]. En estas situaciones, es posible elegir una estrategia de desnormalización. Por ejemplo, en un sistema donde se requiere análisis de datos con alta frecuencia de consulta, se podrían almacenar datos redundantes para evitar uniones de múltiples tablas, que pueden ralentizar el rendimiento en momentos de alta demanda.

Los ejemplos de aplicación del modelado de datos relacional abarcan diversas industrias. En el ámbito de la salud, el modelado ayuda a gestionar registros médicos, identificando tablas para "Pacientes", "Médicos", "Citas" y "Tratamientos". Las relaciones entre estas entidades permiten acceder rápidamente al historial médico de un paciente y a los tratamientos que ha recibido.

En el sector financiero, se pueden modelar entidades tales como "Cuentas", "Transacciones" y "Tarjetas de Crédito". En este caso, cada cuenta puede tener múltiples transacciones y cada transacción puede involucrar una tarjeta de crédito específica. La estructura relacional permite realizar análisis sobre gastos y patrones de comportamiento financiero de forma eficaz.

Además, el manejo adecuado del modelado de datos relacional permite la integración con tecnologías y herramientas de análisis, mejorando la capacidad de las organizaciones para tomar decisiones basadas en datos. Los sistemas de gestión de bases de datos como Oracle Database, Microsoft SQL Server o MySQL ofrecen entornos que permiten implementar estas prácticas de modelado y proporcionan funcionalidades robustas para la gestión y consulta de datos.

El modelado de datos relacional establece un sistema organizativo para la información, actuando como un cimiento que soporta el análisis y optimización de datos para el crecimiento y la efectividad operativa de las organizaciones en una amplia variedad de sectores.

[1] La experiencia aconseja que, independientemente de que una BBDD esté normalizada o no, los datos críticos deberían estar repartidos entre varias tablas, por motivos de seguridad.

2. LENGUAJE DE DEFINICIÓN DE DATOS (DDL)

El Lenguaje de Definición de Datos (DDL) forma parte de un sistema de gestión de bases de datos y **se utiliza para definir y modificar la estructura de la base de datos**. A través de DDL, es posible crear, alterar y eliminar objetos dentro de la base de datos, como tablas y esquemas. Este lenguaje incluye comandos que permiten describir la forma y las características de los datos almacenados.

Los comandos más comunes de DDL son 'CREATE', 'ALTER' y 'DROP'. El comando CREATE permite la creación de nuevas tablas y estructuras dentro de la base de datos, definiendo los tipos de datos y restricciones para cada columna. ALTER se utiliza para modificar la estructura de objetos existentes, lo que permite, por ejemplo, agregar o eliminar columnas en una tabla. Por otro lado, el comando DROP se utiliza para eliminar tablas o bases de datos **completas**, asegurando que todos los datos asociados se borren **permanentemente**.

Una característica importante de DDL es que sus operaciones son generalmente transaccionales; si ocurre un error durante la ejecución de un comando DDL, es posible revertir la transacción para evitar la creación de estructuras incompletas o inconsistentes. Además, los comandos DDL son frecuentemente utilizados por administradores de bases de datos para establecer políticas de seguridad y definir permisos de acceso, contribuyendo a la integridad y seguridad de la información gestionada.

El uso apropiado de DDL es necesario, ya que cualquier modificación o eliminación incorrecta de estructuras de bases de datos puede afectar el rendimiento y la disponibilidad de la información.

2.1. CREACIÓN DE BASES DE DATOS Y TABLAS

La creación de bases de datos y tablas implica procesos que son fundamentales en el desarrollo de sistemas de gestión de datos en bases de datos relacionales. A través del Lenguaje de Definición de Datos (DDL), se pueden establecer las estructuras necesarias para almacenar información. Este proceso asegura que los datos estén organizados, sean accesibles y mantengan la integridad.

El primer paso consiste en crear la base de datos, que establece el entorno necesario para almacenar tablas y otros objetos. Mediante la instrucción `CREATE DATABASE`, se puede definir una nueva base de datos. Por ejemplo, para un sistema de gestión de ventas, si se desea establecer una base de datos llamada `Tienda`, se realizaría de la siguiente manera:

```
CREATE DATABASE Tienda;
```

Posteriormente, es necesario crear las tablas que almacenarán los datos. Las tablas se organizan en filas y columnas. Cada columna está definida por un nombre y un tipo de dato que determina el tipo de información que se puede almacenar, como texto, números o fechas.

La sintaxis básica para la instrucción `CREATE TABLE` comienza con el nombre de la tabla y la lista de columnas junto con sus tipos de datos. Por ejemplo, para crear una tabla denominada

`Productos`, que contenga información sobre los artículos disponibles, se podría utilizar el siguiente comando:

```
CREATE TABLE Productos (     id INT PRIMARY KEY AUTO_INCREMENT,     nombre VARCHAR(100)
NOT NULL,     precio DECIMAL(10, 2) NOT NULL,     stock INT DEFAULT 0,     fecha_ingreso DATE );
```

En este caso, se establece un campo `id` que es un número entero que se autoincrementa, garantizando que cada producto tenga un identificador único. El campo `nombre` permite almacenar una cadena de texto de hasta 100 caracteres. El campo `precio` es un número decimal que puede contener valores con hasta dos decimales, `stock` representa la cantidad disponible y `fecha_ingreso` almacena la fecha en la que el producto fue agregado a la base de datos.

Es relevante definir adecuadamente las restricciones en las columnas. Por ejemplo, `NOT NULL` indica que ciertos campos son obligatorios, mientras que `DEFAULT` permite asignar un valor predeterminado en caso de que no se proporcione uno al insertar un registro. Esto es útil para asegurarse de que la cantidad de productos no sea negativa al establecer su valor predeterminado en cero.

En sistemas con múltiples tablas, es común establecer relaciones entre ellas. Esto se define mediante claves foráneas, que son columnas que establecen un vínculo con la clave primaria de otra tabla. Por ejemplo, para registrar las ventas de productos, se podría crear una tabla `Ventas` que se vincule con la tabla `Productos`:

```
CREATE TABLE Ventas (     id INT PRIMARY KEY AUTO_INCREMENT,     producto_id INT,     cantidad
INT,     fecha_venta DATE,     FOREIGN KEY (producto_id) REFERENCES Productos(id) );
```

En este caso, `producto_id` es una clave foránea que hace referencia al campo `id` de la tabla `Productos`. Así, se puede rastrear qué producto se ha vendido en cada transacción, manteniendo la integridad entre las tablas.

Un ejemplo práctico de estas tablas se observa en un sistema que gestiona inventarios y ventas de una tienda. Si un empleado desea conocer las ventas de un producto específico, podría ejecutarse la siguiente consulta para obtener información detallada de ambas tablas:

```
SELECT Productos.nombre, Ventas.cantidad, Ventas.fecha_venta
FROM Ventas
JOIN Productos ON Ventas.producto_id = Productos.id
WHERE Productos.nombre = 'Producto Ejemplo';
```

Esta instrucción aporta el nombre del producto, la cantidad vendida y la fecha de venta relacionada con el producto especificado. A través de este tipo de consultas, se accede a información que asiste en la toma de decisiones.

Además de la creación, a menudo se necesita modificar la estructura de las tablas mediante la instrucción `ALTER TABLE`. Esta instrucción permite funciones como agregar columnas adicionales, cambiar tipos de datos o eliminar columnas que ya no se requieren. Por ejemplo:

```
ALTER TABLE Productos ADD COLUMN descripcion TEXT;
```

Este comando añadiría una columna `descripcion` a la tabla `Productos`, permitiendo agregar información adicional sobre cada artículo.

Si se necesita modificar el tipo de dato de una columna, como cambiar el tipo de `precio` para aceptar más decimales, se usaría:

```
ALTER TABLE Productos MODIFY COLUMN precio DECIMAL(10, 4);
```

También es posible eliminar columnas que no son necesarias. Para eliminar el campo `fecha_ingreso`, el comando sería:

```
ALTER TABLE Productos DROP COLUMN fecha_ingreso;
```

Cuando las tablas ya no son requeridas, se puede proceder a su eliminación mediante la instrucción `DROP TABLE`. Por ejemplo, para borrar la tabla `Ventas`:

```
DROP TABLE Ventas;
```

Además de la creación y modificación, el DDL permite establecer índices en las tablas, lo que mejora la velocidad de las consultas al facilitar que el motor de la base de datos acceda a los registros de manera más rápida. La instrucción para crear un índice es:

```
CREATE INDEX idx_nombre ON Productos(nombre);
```

Este índice mejora la eficiencia de las consultas que realizan búsquedas basadas en el nombre de los productos.

El DDL se presenta como una herramienta útil para el manejo de bases de datos relacionales. Proporciona a los desarrolladores las capacidades necesarias para crear, modificar y organizar datos de manera efectiva, asegurando que cada operación en la base de datos mantenga la integridad y facilita el acceso a la información. A medida que se presentan diversas situaciones de uso práctico, queda claro el importe de comprender y aplicar correctamente estos conceptos en cualquier sistema que requiera un manejo efectivo de datos.

2.2. MODIFICACIÓN DE ESTRUCTURAS DE TABLAS

La modificación de estructuras de tablas en bases de datos relacionales permite realizar ajustes en el diseño para adaptarse a la evolución de los datos y a las necesidades de la organización. Estas

operaciones se llevan a cabo a través del Lenguaje de Definición de Datos (DDL), utilizando la instrucción `ALTER TABLE`.

Adición de columnas

Agregar columnas es común cuando es necesario almacenar información nueva. Por ejemplo, en un sistema de gestión de reservas de viajes, una tabla llamada `Clientes` puede contener las columnas `ID`, `Nombre` y `Email`. Si se decide incluir el país de residencia del cliente, se puede añadir una columna `Pais` de la siguiente manera:

```
ALTER TABLE Clientes ADD COLUMN Pais VARCHAR(50);
```

Con esta modificación, se permite almacenar información adicional sobre el cliente, útil para personalizar ofertas.

Este proceso también encuentra aplicación en sectores específicos. En un sistema de gestión de inventario, si se quiere llevar un registro del proveedor de cada producto, es posible agregar una columna `ProveedorID` a la tabla `Productos` con:

```
ALTER TABLE Productos ADD COLUMN ProveedorID INT;
```

Este cambio fortalece la normalización de la base de datos al establecer relaciones con otras tablas.

Eliminación de columnas

Eliminar columnas es importante para mantener la base de datos libre de datos innecesarios. Por ejemplo, si en una tabla `Ordenes` hay una columna denominada `ReferenciaInterna` que ya no se utiliza, se puede eliminar así:

```
ALTER TABLE Ordenes DROP COLUMN ReferenciaInterna;
```

Este tipo de operación simplifica el modelo de datos y facilita su administración, al reducir la posibilidad de confusiones al manipular datos desactualizados.

Modificación del tipo de datos de una columna

Cambiar el tipo de dato de una columna puede ser necesario. Por ejemplo, si se tiene una columna `Precio` en la tabla `Productos` definida como `FLOAT`, se puede modificar a `DECIMAL` para mejorar la precisión:

```
ALTER TABLE Productos MODIFY COLUMN Precio DECIMAL(10, 2);
```

Esto permite que las operaciones relacionadas con precios se realicen sin errores en los decimales.

Adición de restricciones

Las restricciones son importantes para asegurar la calidad de los datos. Por ejemplo, para evitar que se repitan las direcciones de correo electrónico en la tabla `Usuarios`, se puede establecer una restricción única:

```
ALTER TABLE Usuarios ADD CONSTRAINT UQ_Email UNIQUE (Email);
```

Esto garantiza que no haya registros duplicados, lo que mejora la integridad de la información.

Además, en situaciones donde la información personal es relevante, se puede requerir que ciertos campos, como `Telefono`, no permitan valores nulos con la siguiente instrucción:

```
ALTER TABLE Usuarios MODIFY COLUMN Telefono VARCHAR(15) NOT NULL;
```

Este ajuste exige que todos los registros tengan un número de teléfono asociado.

Creación de índices

Los índices se usan para mejorar el rendimiento en consultas. Si hay aplicaciones que realizan frecuentemente búsquedas filtradas por el nombre de un producto, es recomendable crear un índice en la columna correspondiente de esta forma:

```
CREATE INDEX idx_nombre_producto ON Productos (Nombre);
```

Esto optimiza la recuperación de datos en tablas con un gran volumen de registros.

La creación de índices también puede ser útil en sistemas como el de una biblioteca. Si se suelen buscar libros por `Autor`, tener un índice en la columna `Autor` permite que estas consultas se realicen con mayor eficacia.

Reorganización de columnas

La reorganización de columnas implica mover la posición de columnas dentro de una tabla. Aunque no afecta directamente a los datos, puede facilitar la lectura y el mantenimiento de la tabla. Esta acción puede variar según el sistema de gestión de bases de datos, debido a que no todos proporcionan una instrucción estándar para ello.

Gestión de cambios

Es importante implementar un sistema que permita seguir los cambios. Documentar modificaciones con fechas y motivos facilita el mantenimiento futuro de la base de datos.

También es relevante tener cuidado al realizar modificaciones. Por ejemplo, en un sistema de comercio electrónico, cambiar el tipo de dato de una columna que contiene información sobre pedidos puede afectar las transacciones en tiempo real. Se recomienda hacer estas modificaciones primero en entornos de desarrollo o pruebas antes de implementarlas en el entorno de producción.

Además, algunas modificaciones pueden requerir ajustes en los datos existentes, como al modificar el tipo de datos de una columna, por lo que un análisis previo de la información es recomendable para evitar pérdidas.

El procedimiento para modificar estructuras de tablas es importante para asegurar que la base de datos se mantenga eficiente y se adapte a las necesidades del negocio, logrando una estructura que facilite la entrada, análisis y recuperación de datos.

2.3. ELIMINACIÓN DE BASES DE DATOS Y TABLAS

La eliminación de bases de datos y tablas mediante el uso del Lenguaje de Definición de Datos (DDL) requiere un conocimiento preciso de los comandos SQL utilizados y de las implicaciones de su aplicación.

El comando `DROP DATABASE` se utiliza para eliminar una base de datos completa. Su uso es sencillo, pero requiere una cuidadosa consideración. La sintaxis básica es:

```
DROP DATABASE nombre_base_datos;
```

Este comando eliminará no solo la base de datos, sino también todas sus tablas, registros, vistas, procedimientos almacenados y cualquier otro objeto relacionado. Por ejemplo, si un desarrollador de software tiene una base de datos llamada `ventas`, ejecutar este comando resultará en la pérdida de toda la información sobre las transacciones, productos y clientes, sin posibilidad de recuperación a menos que exista un respaldo.

En la práctica, este escenario puede ocurrir durante una fase de desarrollo donde una base de datos de pruebas no se necesita más. Sin embargo, en un entorno productivo, eliminar una base de datos sin evaluar sus dependencias podría llevar a la pérdida de información crítica. Un caso de uso claro se presenta en empresas que gestionan pedidos, donde la base de datos de pedidos no debe eliminarse mientras los equipos de ventas y logística la estén utilizando.

El comando `DROP TABLE` elimina una tabla específica dentro de una base de datos. Su sintaxis es:

```
DROP TABLE nombre_tabla;
```

Esto significa que todos los datos contenidos en esa tabla se pierden permanentemente. Por ejemplo, si se ejecuta el siguiente comando:

```
DROP TABLE clientes;
```

La tabla `clientes`, que podría contener información como nombres, direcciones y números de teléfono, será eliminada y no se podrá acceder a esos datos a menos que se haya realizado previamente una copia de seguridad.

Un escenario donde se aplica `DROP TABLE` es cuando se decide reorganizar una base de datos. Si se identifica que una tabla se ha vuelto obsoleta debido a cambios en los requerimientos del negocio, se puede optar por eliminarla tras crear una nueva tabla que incorpore el diseño adecuado para almacenar los datos necesarios.

Además, es relevante tener en cuenta la integridad referencial. Cuando una tabla está relacionada con otras mediante claves foráneas, su eliminación debe ser tratada con cuidado. Por ejemplo, si se tiene una tabla `ordenes` que hace referencia a la tabla `clientes`, y se intenta eliminar `clientes` sin antes eliminar las órdenes asociadas, se puede generar un error de integridad referencial. Para manejar esta situación, **se puede optar por eliminar las órdenes relacionadas primero o utilizar la opción `ON DELETE CASCADE`**, que permite que, al eliminar una entrada en la tabla referenciada, se eliminen automáticamente las entradas correspondientes en la tabla que contiene la clave foránea. Su uso sería así:

```
ALTER TABLE ordenes
ADD CONSTRAINT fk_cliente FOREIGN KEY (cliente_id)
REFERENCES clientes(id)
ON DELETE CASCADE;
```

El uso del comando `TRUNCATE TABLE` es diferente de `DROP TABLE`. La sintaxis es:

```
TRUNCATE TABLE nombre_tabla;
```

Al utilizar `TRUNCATE`, todos los registros de la tabla especificada son eliminados, pero la estructura de la tabla permanece intacta. Este comando es más eficiente en comparación con `DELETE`, pues **no genera una entrada en el log para cada fila eliminada y no requiere que se mantenga la integridad referencial**. Esto es conveniente en situaciones donde se desea eliminar grandes volúmenes de datos de manera rápida. Por ejemplo, en un sistema de ventas, puede ser necesario truncar una tabla de `transacciones` al final de un período de prueba para limpiar los datos sin afectar la estructura de la tabla.

Por otro lado, es importante realizar copias de seguridad. Utilizando MySQL, se puede crear una copia de seguridad de toda la base de datos con el comando `mysqldump`:

```
mysqldump -u usuario -p nombre_base_datos > respaldo.sql
```

Esto genera un archivo que contiene todas las instrucciones necesarias para recrear la base de datos en su estado previo, lo que representa una medida de seguridad notable antes de cualquier eliminación.

También es pertinente considerar la implementación de permisos y roles de usuario en un entorno de bases de datos. Limitar quién puede ejecutar comandos de eliminación es un enfoque estratégico para prevenir eliminaciones accidentales o malintencionadas. A través de la asignación

de privilegios adecuados, se puede garantizar que solo los usuarios autorizados tengan la capacidad de modificar o eliminar estructuras de bases de datos.

Al llevar a cabo una eliminación de bases de datos o tablas, se deben tener en cuenta procedimientos establecidos y seguir prácticas recomendadas. Realizar auditorías previas, mantener un registro de las operaciones, y llevar a cabo un análisis de impacto en el sistema son pasos que contribuyen a una adecuada gestión de las eliminaciones.

Finalmente, realizar pruebas de eliminación en entornos de desarrollo o pruebas antes de aplicar cambios en producción es una práctica recomendable. Esto permite verificar el comportamiento del sistema y garantiza que se minimiza el riesgo de interrupciones en la operación normal de la base de datos o la pérdida de datos importantes.

3. LENGUAJE DE MANIPULACIÓN DE DATOS (DML)

El Lenguaje de Manipulación de Datos (DML) representa una porción del lenguaje de consulta utilizado en bases de datos, permitiendo la interacción con la información almacenada en sistemas relacionales. **Este lenguaje ofrece las instrucciones necesarias para realizar acciones sobre los datos, incluyendo la inserción, modificación y eliminación de registros**.

Las instrucciones de inserción permiten añadir nuevos datos a las tablas dentro de la base de datos, mientras que la modificación está destinada a alterar los registros existentes. La eliminación permite suprimir aquellos datos que ya no son requeridos. **Cada una de estas operaciones se ejecuta mediante sentencias específicas, tales como 'INSERT', 'UPDATE' y 'DELETE'**.

Por otro lado, el DML facilita la recuperación de datos mediante la sentencia 'SELECT', que permite extraer información específica según diferentes criterios. Esta funcionalidad de consulta resulta significativa para el análisis y la obtención de información relevante en una base de datos.

Además, **el DML permite implementar el control de transacciones**, lo que asegura que las operaciones realizadas sobre la base de datos se ejecuten de forma coherente y segura. Este aspecto es relevante en entornos en los que múltiples usuarios interactúan con los datos al mismo tiempo, ayudando a mantener la integridad y la precisión de la información.

Al ser un componente del lenguaje SQL, el DML interrelaciona con otras partes del mismo, como el Lenguaje de Definición de Datos (DDL), que maneja la estructura de las bases de datos, y el Lenguaje de Control de Datos (DCL), que se ocupa de los permisos y accesos a la información. Esta interacción posibilita un manejo adecuado y eficiente del sistema de bases de datos, integrando tanto la definición como la manipulación de la información.

3.1. INSERCIÓN, ACTUALIZACIÓN Y BORRADO DE REGISTROS

La inserción, actualización y borrado de registros son acciones fundamentales para la gestión de datos en bases de datos relacionales mediante el Lenguaje de Manipulación de Datos (DML). Cada una de estas acciones tiene un funcionamiento específico y se utiliza en diferentes circunstancias para garantizar que la información almacenada sea precisa y relevante.

La inserción de registros implica añadir nuevas filas a una tabla existente. Este proceso se puede realizar de dos maneras: mediante la inclusión de todos los campos de forma explícita o solo de algunos de ellos. Cuando se inserta un registro sin especificar todos los campos, resulta importante entender cómo funcionan los valores por defecto en la estructura de la tabla.

Por ejemplo, si en una tabla de `clientes` solo se necesita registrar el nombre y el correo electrónico, y los otros campos tienen valores predeterminados, el comando podría ser:

```
INSERT INTO clientes (nombre, email)  VALUES ('Juan Pérez', 'juan.perez@example.com');
```

En este caso, los campos que no se especifican asumirán los valores predeterminados establecidos en el esquema de la tabla. Si se requiere insertar un registro completo, el comando podría ser:

```
INSERT INTO clientes (nombre, email, telefono, direccion)  VALUES ('María López',
'maria.lopez@example.com', '123456789', 'Calle Falsa 123');
```

Otra operación en la inserción es la inserción masiva, que permite agregar múltiples registros simultáneamente. Por ejemplo, si se desea cargar un conjunto de datos de libros, se podría realizar de la siguiente manera:

```
INSERT INTO libros (titulo, autor, genero, anio_publicacion)
VALUES  ('Cien Años de Soledad', 'Gabriel García Márquez', 'Realismo Mágico', 1967),
('La Casa de los Espíritus', 'Isabel Allende', 'Ficción', 1982),
('Crónica de una muerte anunciada', 'Gabriel García Márquez', 'Novela', 1981);
```

La operación de actualización permite modificar uno o varios campos de un registro existente. Utilizar correctamente la cláusula `WHERE` es determinante para evitar cambios indeseados. Por ejemplo, si se necesita actualizar el teléfono de un cliente cuyo nombre es "Juan Pérez", el comando sería:

```
UPDATE clientes SET telefono = '987654321'  WHERE nombre = 'Juan Pérez';
```

Si se desea aumentar el año de publicación de los libros en un cierto género, se podría usar:

```
UPDATE libros SET anio_publicacion = anio_publicacion + 1  WHERE genero = 'Ficción';
```

Además, se pueden utilizar condiciones más complejas en la cláusula `WHERE`, como el uso de operadores lógicos:

```
UPDATE clientes SET email = 'juan.perez@actualizado.com'  WHERE nombre = 'Juan Pérez'
AND telefono = '123456789';
```

Esto asegura que solo se actualizará el registro que cumpla ambas condiciones.

La operación de borrado permite eliminar registros específicos de una tabla. Utilizar la cláusula `WHERE` es igual de importante. Un ejemplo para eliminar un cliente cuyo registro se considera obsoleto sería:

```
DELETE FROM clientes WHERE nombre = 'María López';
```

En circunstancias donde se desea eliminar un conjunto de registros, como todos los libros de un autor específico, el comando podría ser:

```
DELETE FROM libros WHERE autor = 'Gabriel García Márquez';
```

En algunos casos, puede ser útil eliminar todos los registros de una tabla, lo que se puede hacer omitiendo la cláusula `WHERE`. Por ejemplo, si se necesita reiniciar la tabla de `libros`, se puede usar:

```
DELETE FROM libros;
```

Sin embargo, esto debe hacerse con cuidado, ya que se perderán todos los registros inmediatamente.

En lo que respecta a las transacciones, las operaciones de inserción, actualización y borrado pueden agruparse para asegurar que se realicen de forma atómica. Por ejemplo, en una aplicación de tienda, al procesar un pedido, puede ser necesario insertar un nuevo registro en la tabla de pedidos y actualizar el stock de productos. La estructura de la transacción podría ser:

```
BEGIN;
INSERT INTO pedidos (cliente_id, producto_id, cantidad, fecha)
VALUES (1, 5, 2, '2023-10-10');
UPDATE productos
SET stock = stock - 2
WHERE id = 5;
COMMIT;
```

Si alguna de las operaciones falla, se puede utilizar `ROLLBACK` para revertir todos los cambios realizados en la transacción. Esto permite mantener la coherencia de los datos incluso ante errores.

Cada operación desempeña un propósito importante en la gestión dinámica de bases de datos, permitiendo adaptar las necesidades cambiantes de las aplicaciones y garantizar que los datos reflejen la realidad de manera precisa. La comprensión y correcto uso de estas instrucciones DML son imprescindibles para cualquier aplicación que interactúe con bases de datos durante el desarrollo de software.

3.2. CONSULTAS A LA BASE DE DATOS

Las consultas a la base de datos son un componente central del Lenguaje de Manipulación de Datos (DML). Permiten interactuar de manera eficaz con los datos almacenados en bases de datos relacionales utilizando SQL (Structured Query Language). A continuación, se describen en detalle las distintas secciones de las consultas a la base de datos, incluyendo la selección, condiciones de filtrado, funciones de agregación, agrupamiento, uniones, ordenación, subconsultas y limitación de resultados.

La selección de datos se realiza con el comando 'SELECT', que permite especificar las columnas que se desean obtener de las tablas. Un ejemplo básico de consulta es:

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

Aquí, se seleccionan las columnas "nombre" y "edad" de la tabla "empleados". Sin embargo, las consultas pueden ser más complejas. Por ejemplo, si se quiere obtener información sobre todos los empleados y sus respectivos salarios, se puede ampliar la consulta:

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

Esta consulta recupera tres columnas, brindando una visión más amplia de la información sobre cada empleado.

Las consultas pueden incluir condiciones de filtrado mediante la cláusula WHERE, que restringe los resultados según criterios específicos. Un ejemplo podría ser la búsqueda de empleados cuyo salario supere los 3000 euros:

```
SELECT nombre, salario FROM empleados WHERE salario > 3000;
```

Las condiciones pueden combinarse con operadores lógicos como AND y OR, permitiendo crear filtros más complejos. Por ejemplo, para obtener empleados que tengan más de 30 años o que se encuentren en el departamento de ventas, la consulta se vería así:

```
SELECT nombre, edad FROM empleados WHERE edad > 30 OR departamento = 'Ventas';
```

Además, se pueden utilizar operadores de comparación como BETWEEN para definir un rango de valores. Por ejemplo, si se desea recuperar a los empleados cuya edad esté entre 25 y 35 años, la consulta sería:

```
SELECT nombre, edad FROM empleados WHERE edad BETWEEN 25 AND 35;
```

Las funciones de agregación permiten realizar cálculos sobre un conjunto de valores y regresan un solo valor resultado. Funciones como COUNT, SUM, AVG, MAX y MIN son frecuentemente utilizadas. Un caso de uso típico con COUNT sería para determinar el número total de empleados en la empresa:

```
SELECT COUNT(*) FROM empleados;
```

Este comando devuelve un único valor que representa el total de registros en la tabla empleados. Si se quiere calcular el salario promedio de los empleados de un departamento específico, se aplicaría la función AVG:

```
SELECT AVG(salario) FROM empleados WHERE departamento = 'Recursos Humanos';
```

Esto recupera el salario medio solo para aquellos empleados en el departamento de Recursos Humanos.

La cláusula **GROUP BY** permite agrupar resultados en función de una o más columnas, lo que es útil para aplicar funciones de agregación a grupos de datos. Por ejemplo, para contar el número de empleados en cada departamento, se puede utilizar la siguiente consulta:

```
SELECT departamento, COUNT(*) FROM empleados GROUP BY departamento;
```

Este comando mostrará una lista de departamentos y el número de empleados que pertenecen a cada uno.

Se pueden aplicar condiciones sobre grupos utilizando la cláusula 'HAVING'[2], que filtra los resultados después de haber realizado el agrupamiento. Por ejemplo, si se desea mostrar solo aquellos departamentos con más de diez empleados:

```
SELECT departamento, COUNT(*) FROM empleados GROUP BY departamento HAVING
COUNT(*) > 10;
```

Esto devolverá solo los departamentos donde la cantidad de empleados es superior a diez.

Las uniones (JOINs) son importantes para combinar datos de diferentes tablas y están relacionadas. La unión más común es el **INNER JOIN**, que devuelve solo las filas donde hay coincidencia en ambas tablas. Supongamos que se tiene una segunda tabla llamada "departamentos":

```sql
SELECT e.nombre, d.nombre_departamento
FROM empleados e
INNER JOIN departamentos d ON e.departamento_id = d.id;
```

Este comando permite obtener los nombres de los empleados junto con el nombre de su respectivo departamento, utilizando la relación definida entre las tablas.

Otro tipo de unión es la **LEFT JOIN**, que devuelve todas las filas de la tabla de la izquierda y las filas coincidentes de la tabla de la derecha. Si no hay coincidencias, se devolverán NULL para las columnas de la tabla de la derecha. Por ejemplo:

```sql
SELECT e.nombre, d.nombre_departamento
FROM empleados e
LEFT JOIN departamentos d ON e.departamento_id = d.id;
```

Este comando recupera todos los empleados y los departamentos asociados, incluidos aquellos empleados que no pertenecen a ningún departamento.

[2] 'HAVING' filtra a 'GROUP BY', es decir, siempre va detrás

La cláusula ORDER BY se utiliza para ordenar los resultados de las consultas. Por defecto, esta ordenación se realiza de manera ascendente, **pero puede especificarse que sea descendente utilizando el modificador 'DESC'.** Por ejemplo, para recuperar empleados ordenados por salario de mayor a menor, se ejecutaría:

```
SELECT nombre, salario FROM empleados ORDER BY salario DESC;
```

Es posible ordenar por varias columnas simultáneamente. Por ejemplo, si se deseara obtener empleados ordenados por departamento y luego por edad de manera ascendente:

```
SELECT nombre, edad, departamento FROM empleados ORDER BY departamento ASC, edad ASC;
```

Esto organiza los resultados jerárquicamente primero por departamento y, dentro de cada departamento, por edad.

Las subconsultas son consultas anidadas que pueden utilizarse dentro de otra consulta. Pueden aparecer en varias partes de una consulta, incluyendo las cláusulas SELECT, FROM y WHERE. Por ejemplo, se puede utilizar una subconsulta en la cláusula WHERE para seleccionar empleados cuyos salarios sean superiores al salario promedio de todos los empleados:

```
SELECT nombre FROM empleados WHERE salario > (SELECT AVG(salario) FROM empleados);
```

Esta consulta permite identificar a aquellos empleados que están por encima del umbral del salario promedio.

La limitación de resultados se puede gestionar mediante la cláusula 'LIMIT', que establece un máximo en la cantidad de registros devueltos. Por ejemplo, para listar solo los primeros cinco empleados según su salario:

```
SELECT nombre, salario FROM empleados ORDER BY salario DESC LIMIT 5;
```

Esto devuelve únicamente los cinco empleados con los salarios más altos.

Además, es importante considerar la optimización de las consultas. Usar índices en columnas usados en cláusulas WHERE, JOIN o ORDER BY puede mejorar los tiempos de respuesta de las consultas. Por ejemplo, si se establece un índice sobre la columna "departamento_id" en la tabla empleados, las búsquedas que involucren ese campo serán más rápidas.

No obstante, la creación de índices debe ser balanceada según el tipo de operaciones que se realicen a menudo. **Demasiados índices pueden ralentizar las operaciones de inserción y actualización de los datos, dado que el sistema debe mantener actualizados todos los índices asociados cada vez que se modifica la tabla**.

Finalmente, es relevante tener en cuenta que SQL es sensible a las diferencias en mayúsculas y minúsculas dependiendo del motor de la base de datos utilizado, lo que puede afectar la forma en que se escriben las consultas. La escritura adecuada y el uso de nombres de tabla y columna de forma coherente son indispensables para evitar errores y lograr el óptimo rendimiento en las consultas.

3.3. CONTROL DE TRANSACCIONES

El control de transacciones en bases de datos relacionales se organiza en torno a las propiedades ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad), que garantizan la integridad de los datos y aseguran que las operaciones se ejecuten correctamente. A continuación, se explican estas propiedades junto con ejemplos y situaciones prácticas.

Atomicidad

La atomicidad significa que una transacción se ejecuta como una unidad indivisible. Esto implica que, si una transacción está compuesta por varias operaciones, todas deben completarse con éxito para que los cambios se apliquen. Si alguna operación falla, todos los cambios realizados anteriormente se revierten, evitando así que haya modificaciones incompletas en la base de datos.

Ejemplo: En una aplicación bancaria, cuando un cliente realiza un retiro, esta operación incluye debitar su cuenta y registrar la transacción. Si se produce un fallo después de debitar la cuenta, pero antes de registrar la transacción, la cuenta no debe reflejar un saldo incorrecto. En este caso, la transacción debe revertirse completamente.

Consistencia

La propiedad de consistencia garantiza que las reglas y restricciones impuestas a los datos se mantengan al ejecutar transacciones. Cada transacción debe llevar la base de datos de un estado válido a otro también válido, de acuerdo con las reglas de integridad definidas.

Ejemplo: En un sistema de gestión de inventarios, si existe la regla que establece que el stock de un producto no puede ser negativo, la transacción no debería finalizar si se intenta realizar una venta con una cantidad mayor a la disponible.

Aislamiento

El aislamiento se refiere a la capacidad de ejecutar transacciones de forma simultánea sin interferencias entre ellas. Cada transacción debe operar independientemente de las demás, de manera que su ejecución no afecte el estado de los datos que otras transacciones están utilizando.

Ejemplo: En un sistema de control de stock, si dos empleados intentan reservar el último artículo disponible al mismo tiempo, el aislamiento garantiza que uno de ellos verá que el artículo no está disponible y no podrá finalizar la transacción, aunque la otra operación continúe.

Durabilidad

La durabilidad se refiere a la garantía de que, una vez confirmada una transacción, sus efectos son permanentes, incluso si ocurre un fallo en el sistema. Tras la ejecución de la instrucción COMMIT, todos los cambios realizados deben ser visibles para futuras transacciones y no pueden ser desechados.

> En un sistema de reservas de vuelos, si un cliente compra un billete y la transacción se confirma, el sistema debe asegurarse de que esta información permanezca, independientemente de cualquier fallo de energía o de hardware. Esto significa que el registro de facturación debe guardarse de forma permanente en la base de datos.

En adición a las propiedades ACID, es relevante considerar los comandos que controlan las transacciones. Los comandos básicos en el uso de DML incluyen:

- `**BEGIN TRANSACTION**`: Inicia una nueva transacción.
- `**COMMIT**`: Confirma una transacción previamente iniciada, aplicando todos los cambios realizados.
- `**ROLLBACK**`: Revierte todos los cambios de la transacción actual, asegurando que no se realicen modificaciones en la base de datos.

Ejemplo práctico de control de transacciones

Imaginemos un escenario en el que una tienda en línea gestiona el proceso de compra. El flujo de la transacción en SQL podría verse así:

```sql
BEGIN TRANSACTION;

-- Verificamos el stock de un producto antes de procesar la compra.
SELECT stock FROM productos WHERE id = 101;

-- Si el stock es suficiente, realizamos la transacción.
UPDATE productos SET stock = stock - 1 WHERE id = 101;

-- Añadimos el registro de compra.
INSERT INTO compras (producto_id, usuario_id, fecha) VALUES (101, 205, GETDATE());

IF @@ERROR <> 0
    ROLLBACK; -- En caso de error, revertimos los cambios.
ELSE
    COMMIT; -- Si todo ha ido bien, confirmamos la transacción.

```

En este ejemplo, el proceso garantiza que solo se reduzca el stock si está disponible y que se inserte un registro de compra solo si ambas operaciones se realizan sin errores. Implementar un enfoque adecuado para el control de transacciones crea un entorno seguro y confiable para la manipulación de datos en situaciones críticas.

La gestión del aislamiento también permite ajustar la configuración a distintos niveles de aislamiento. Estos niveles determinan cómo se debe manejar la visibilidad de los datos entre transacciones en curso. Por ejemplo, si se establece un nivel de aislamiento de 'READ COMMITTED', ninguna transacción podrá leer datos que no han sido confirmados por otras, lo que previene la lectura de información inconsistentes.

Un sistema de control de versiones en bases de datos ayuda a mantener un historial de cambios aplicados en las transacciones. Esta práctica facilita la auditoría y el seguimiento de errores y ofrece la posibilidad de revertir cambios cuando sea necesario. Dicho enfoque contribuye a la transparencia y a la responsabilidad en la gestión de datos.

Para mantener la integridad de los datos en entornos donde ocurren múltiples operaciones simultáneamente, se recomienda implementar un manejo adecuado de errores. Esto incluye la creación de funciones que gestionen la contención de transacciones y que informen a los usuarios sobre el estado de sus operaciones, garantizando así una experiencia satisfactoria.

El control de transacciones representa un componente central en el diseño de bases de datos relacionales. Permite asegurar que las operaciones de manipulación se realicen de manera segura y confiable, respetando las propiedades ACID y gestionando adecuadamente los diferentes niveles de aislamiento. La adecuada implementación del control de transacciones, junto con la gestión efectiva de errores, resulta importante para mantener la integridad de los sistemas de información en entornos profesionales.

4. GESTIÓN DE CLAVES PRIMARIAS Y FORÁNEAS Y SU IMPACTO EN BASES DE DATOS ESCALABLES

La gestión de claves primarias y foráneas es un aspecto importante en el diseño de bases de datos relacionales, dado que estas claves garantizan la integridad y coherencia de los datos.

> Las claves primarias identifican de manera única cada registro en una tabla, lo que establece un esquema claro en el manejo de la información. Las claves foráneas, por su parte, crean vínculos entre diferentes tablas al referirse a las claves primarias de otras, facilitando la normalización y evitando la duplicidad de datos.

El impacto de una correcta gestión de estas claves en bases de datos escalables es considerable. A medida que las aplicaciones crecen y aumenta el volumen de datos, la capacidad para mantener relaciones adecuadas entre los conjuntos de datos se convierte en un factor importante para lograr un rendimiento óptimo. **La escalabilidad se ve afectada por la forma en que estas claves están definidas y utilizadas, ya que una estructura diseñada adecuadamente permite un acceso más ágil y eficiente a la información**, incluso en escenarios con altas demandas de usuarios y operaciones.

Además, el diseño de claves primarias y foráneas influye directamente en las estrategias de replicación y distribución de bases de datos. En entornos distribuidos, la latencia en las consultas y el manejo de transacciones pueden optimizarse si las relaciones entre las entidades son lógicas y bien definidas. Por lo tanto, una gestión adecuada de estas claves no solo facilita la integridad referencial, sino que también apoya la toma de decisiones sobre cómo aumentar la capacidad de las instancias de la base de datos para atender futuros requerimientos.

La modificación o eliminación de registros relacionados mediante claves foráneas debe manejarse con cuidado, debido a la relación directa que mantienen con la integridad de los datos. Implementar restricciones de integridad referencial ayuda a prevenir pérdidas de información coherente durante estas operaciones. Esta práctica se vuelve relevante en el caso de bases de datos escalables, donde los cambios en la estructura de los datos pueden tener un efecto significativo en la operación general del sistema.

En sistemas altamente escalables, puede ser beneficioso aplicar técnicas como la fragmentación de datos, donde las tablas se dividen en partes más pequeñas para mejorar la gestión y el rendimiento. Sin embargo, una fragmentación inadecuada que no considere las claves primarias y foráneas puede llevar a un aislamiento de datos y complicar las relaciones, lo que obstaculiza el acceso ágil a la información. Por consiguiente, la planificación estratégica en la definición y manejo de estas claves es necesaria para asegurar que la escalabilidad no comprometa la integridad y relación entre los datos.

Preparar bases de datos para el crecimiento futuro incluye evaluar la posibilidad de implementar índices relacionados con las claves, dado que esto influye directamente en la eficacia de las

consultas. Las decisiones sobre índices deben analizarse en función de las características de las operaciones más comunes en la base de datos y el tipo de acceso a los datos esperado en un entorno escalable.

4.1. IMPORTANCIA DE LAS CLAVES EN EL DISEÑO DE BASES DE DATOS

Las claves son componentes que tienen un impacto significativo en el diseño de bases de datos relacionales, al proporcionar la estructura necesaria para mantener la integridad, normalización y eficiencia en el almacenamiento de datos. Existen dos tipos principales de claves: claves primarias y claves foráneas, que desempeñan funciones específicas dentro del sistema de gestión de bases de datos.

Las claves primarias consisten en atributos o combinaciones de atributos en una tabla que garantizan la unicidad de cada fila. Esto significa que no puede haber dos filas iguales en la misma tabla respecto al valor de la clave primaria. **La elección de una clave primaria adecuada es un aspecto importante**. Por ejemplo, en una tabla que gestiona la información de los productos, se podría utilizar el código de producto como clave primaria. Este código, al ser único para cada producto, simplifica la identificación y permite realizar operaciones de búsqueda y actualización de datos de manera más efectiva.

> Es recomendable optar por un atributo que no cambie con frecuencia al seleccionar una clave primaria. Cualquier cambio en la clave primaria conlleva la necesidad de actualizar todas las referencias relacionadas.

Utilizar un identificador natural, como un número de identificación personal, puede ser práctico; sin embargo, en algunos casos, emplear un número generado automáticamente o un UUID (Identificador Único Universal) garantiza su unicidad y estabilidad.

Las claves foráneas son atributos en una tabla que hacen referencia a la clave primaria de otra tabla. Establecen relaciones entre tablas y permiten a la base de datos mantener la integridad referencial. Por ejemplo, en un sistema de ventas, puede existir una tabla de clientes con datos de estos y una tabla de pedidos que registra los pedidos realizados. La tabla de pedidos incluiría una clave foránea que apunta a la clave primaria en la tabla de clientes. Esto asegura que cada pedido esté vinculado a un cliente existente. Si se intenta insertar un pedido que referencia a un cliente que no existe, se generará un error, protegiendo así la coherencia de los datos.

Un caso de uso en un sistema de gestión de inventario puede ilustrar la importancia de las relaciones entre tablas. En este sistema, se podría tener una tabla de proveedores y una tabla de productos. Cada producto en la tabla de productos tendría una clave foránea que apunta al proveedor que lo suministra. Así, si un proveedor es eliminado de la base de datos, se deben implementar políticas que determinen qué hacer con los productos asociados. Una opción es impedir la eliminación del proveedor si existen productos referenciados, lo que permite mantener la integridad de la información.

La relación entre claves primarias y foráneas también es relevante en el proceso de normalización de bases de datos. La normalización busca minimizar la redundancia de datos y mejorar la integridad. Al dividir tablas en estructuras más pequeñas y relacionadas mediante claves primarias y foráneas, se evita la duplicación. Por ejemplo, en un sistema de gestión de recursos humanos, se podría separar la información de empleados y salarios en tablas distintas. La tabla de empleados contendría la clave primaria de cada empleado, mientras que la tabla de salarios tendría una clave foránea que refiere a la clave primaria de los empleados. De esta manera, se evita la duplicación de datos y se mejora la organización de la información.

El diseño de bases de datos también debe considerar el rendimiento en la gestión de claves. A medida que aumenta la cantidad de datos y las relaciones se vuelven más complejas, es importante que la base de datos esté optimizada para consultas rápidas y eficientes. Crear índices en las columnas que actúan como claves primarias y foráneas mejora significativamente el tiempo de respuesta durante las consultas.

> Si un sistema de gestión de alquileres permite buscar rápidamente inquilinos por su número de identificación, la creación de un índice en la columna correspondiente facilita esta operación, incluso con grandes volúmenes de datos.

Un ejemplo práctico en el ámbito del comercio electrónico puede mostrar cómo un diseño eficiente de relaciones entre tablas impacta en el rendimiento del sistema. Supongamos que se desea analizar las ventas por categoría de producto. En este caso, los datos se encuentran relacionados en varias tablas: una tabla de productos, que clasifica cada producto según su categoría, y una tabla de ventas que registra cada transacción. A través de la utilización de claves foráneas, se pueden realizar consultas que unan estas tablas y generen informes sobre las categorías de productos más vendidos. La capacidad de realizar estas consultas de forma eficiente se apoya en un diseño de base de datos estructurado que utiliza relaciones adecuadas.

Las claves también son necesarias para la escalabilidad en bases de datos. A medida que la cantidad de datos crece, se pueden implementar prácticas como la replicación de datos y la distribución de la carga.

> En un sistema de base de datos distribuido, las claves primarias son importantes para garantizar que cada registro sea único en la red. Incorporar un esquema de partición que considere las claves primarias permite dividir los datos en diferentes servidores sin perder la coherencia mediante el uso de claves foráneas que atraviesan las distintas particiones.

> En una aplicación de gestión de tráfico, es necesario almacenar información sobre vehículos y propietarios en diferentes servidores. Con una clave primaria única, el sistema puede identificar y acceder a los datos sin problemas, asegurando que la información se mantenga consistente a lo largo de la red.

Además, el diseño de las bases de datos que incluye la gestión de claves debe estar respaldado por procedimientos de mantenimiento, como el manejo de índices y la evaluación de su impacto en el rendimiento. A medida que se insertan o eliminan registros, el rendimiento de las consultas puede verse afectado, y es importante mantener el sistema optimizado. En bases de datos modernas que manejan grandes volúmenes de información, el diseño también debe permitir la evolución, de modo que los sistemas puedan adaptarse a nuevas necesidades comerciales o cambios en las relaciones entre datos sin comprometer la integridad.

La implementación de claves en el diseño de bases de datos no solo establece relaciones y asegura la integridad de los datos, sino que también facilita la escalabilidad y el rendimiento, aspectos que son importantes en entornos donde los datos son relevantes. La adecuada implementación y gestión de estas claves deben ser considerados de manera integral para asegurar una operación eficaz en el manejo de bases de datos, permitiendo a las organizaciones analizar, gestionar y utilizar sus datos de manera efectiva.

4.2. ÍNDICES Y OPTIMIZACIÓN DE CONSULTAS

La gestión de claves primarias y foráneas en bases de datos relacionales se enfoca en establecer vínculos y coherencia entre diferentes tablas. Las claves primarias garantizan que cada registro en una tabla sea único, y las claves foráneas permiten la referencia a registros en otras tablas, facilitando de esta manera la integridad de los datos. Esta estructura es la base para construir el acceso a la información, pero a medida que el volumen de datos aumenta, se vuelve necesario optimizar las consultas.

Los índices son estructuras que permiten acelerar la búsqueda y recuperación de datos en las tablas. Se puede entender un índice como una copia de una o más columnas de una tabla organizadas de forma que permiten la búsqueda rápida. Cuando se ejecuta una consulta, el sistema de gestión de bases de datos (SGBD) puede utilizar el índice para localizar rápidamente los registros en lugar de escanear toda la tabla. **Esta optimización resulta especialmente útil en tablas con un alto volumen de registros**.

El uso de índices tiene un impacto directo en el rendimiento de las consultas. Por ejemplo, si un usuario desea encontrar todos los empleados con un apellido determinado en una tabla que contiene miles de registros, la consulta `SELECT FROM empleados WHERE apellido='García'` puede tardar un tiempo considerable si el SGBD realiza un escaneo secuencial de la tabla. Al crear un índice en el campo `apellido`, la consulta se puede ejecutar de manera más eficiente. **El índice permite que el SGBD acceda directamente a las posiciones de los registros que cumplen con el criterio, reduciendo el tiempo de respuesta**.

Existen diferentes tipos de índices, como los índices únicos, que garantizan que los valores de los datos sean únicos, e índices compuestos. Estos últimos son útiles en situaciones donde se requiere filtrar por más de un campo. Por ejemplo, una empresa puede necesitar encontrar empleados por

su `departamento` y `fecha de contratación`. Un índice que contemple ambos campos, `departamento` y `fecha_contratacion`, mejorará la eficiencia de esta consulta.

Además, los índices pueden ser de tipo *B-tree* o *hash*, cada uno adecuado para diferentes tipos de operaciones. Un índice *B-tree* es efectivo para búsquedas entre rangos, mientras que un índice hash es ideal para igualaciones en columnas específicas. Por ejemplo, si se tiene una tabla `ventas` con un gran número de transacciones y se desea localizar rápidamente las ventas realizadas en una fecha específica, un índice *B-tree* sobre la columna de `fecha` proporcionaría eficiencia.

La selección de campos para los índices debe basarse en las consultas más comunes que se realicen. **También es importante evitar un uso excesivo de índices, ya que cada índice adicional puede ralentizar las operaciones de escritura, como inserciones y actualizaciones**. Cada vez que se agrega o modifica un registro, los índices asociados también deben actualizarse, lo que puede consumir tiempo dependiendo del volumen de registros existentes. *Por ejemplo, si se añade un nuevo registro a la tabla de empleados, todos los índices que involucren el campo del apellido tendrán que actualizar su estructura, lo que puede llevar tiempo*.

El proceso de análisis de rendimiento y optimización tiene varias fases:

- Primero, se deben identificar las consultas que presentan un rendimiento deficiente mediante un análisis de registros de consultas lentas. Este análisis puede detectar patrones de uso de índices que no coinciden con consultas efectivas.
- En segundo lugar, la evaluación del plan de ejecución de las consultas mediante la instrucción `EXPLAIN` permite observar si el SGBD aplica los índices correspondientes de manera eficiente. Por ejemplo, al aplicar `EXPLAIN SELECT FROM empleados WHERE apellido='García'`, el resultado mostrará si un índice se utiliza o si se realiza un escaneo completo de la tabla.

Además, la modificación de las consultas puede ser otra estrategia de optimización. En ciertos casos, reescribir la consulta permite que el SGBD utilice un índice ya existente. Por ejemplo, si una consulta no activa un índice, cambiar el orden de los filtros o utilizar otras funciones de SQL puede resultar en una ejecución más eficaz.

El mantenimiento de los índices también es importante para conservar el rendimiento. Con el tiempo, a medida que se realizan muchas operaciones de inserción y eliminación, los índices pueden fragmentarse. **La fragmentación se refiere a la falta de continuidad en el almacenamiento de datos, lo que lleva al SGBD a realizar más operaciones de lectura para acceder a los mismos datos**. Por lo tanto, planificar tareas de reindexación y reconstrucción puede ser una actividad necesaria en las rutinas de mantenimiento.

Los índices se pueden utilizar también para búsquedas de texto completo, implementándose en situaciones donde se requiere localizar términos dentro de grandes bloques de texto. Este tipo de índices permite realizar búsquedas complejas que no se podrían llevar a cabo con un índice

estándar. Por ejemplo, en una aplicación que almacena artículos informativos, la búsqueda de palabras clave en el cuerpo del texto sería más eficiente mediante un índice de texto completo.

Las decisiones sobre la creación, modificación y mantenimiento de índices deben basarse en el análisis continuo del rendimiento de la base de datos y su adecuación a los requisitos de las aplicaciones que la utilizan. La aplicación efectiva de índices y la optimización de consultas lleva a mejorar la capacidad de respuesta del sistema de gestión de bases de datos, manteniendo la integridad de los datos.

4.3. DESAFÍOS Y SOLUCIONES AL ESCALAR BASES DE DATOS RELACIONALES

El escalado de bases de datos relacionales plantea varios desafíos inherentes a la arquitectura y diseño de estas estructuras, especialmente en relación con la gestión de claves primarias y foráneas. Comprender cómo impactan estos desafíos en la escalabilidad permite implementar soluciones efectivas.

La latencia en operaciones de escritura se presenta como un problema frecuente en sistemas de bases de datos altamente interconectados. Al realizar una operación que implica la inserción de un nuevo registro en una tabla con una clave foránea apuntando a otra tabla, es necesario validar la existencia del registro en la tabla referenciada. En un escenario con alto tráfico de datos, esto puede resultar en esperas considerables. Un caso práctico sería una plataforma de comercio electrónico, donde al registrar un nuevo pedido, se verifica que el cliente asociado exista en la tabla de clientes. **Si las tablas están distribuidas y bajo alta carga, las demoras pueden ser significativas**.

Para abordar este problema, se puede optar por el particionamiento de tablas, que implica dividir una tabla grande en varias partes más manejables que pueden ubicarse en diferentes servidores. Por ejemplo, si la tabla de pedidos se particiona según el mes del pedido, las inserciones de diferentes meses pueden manejarse en distintos servidores, reduciendo la contención de recursos. Sin embargo, la gestión de las relaciones entre diferentes particiones requiere atención especial, ya que las consultas que cruzan tablas particionadas pueden ser complejas y menos eficientes.

El *escalado horizontal* de bases de datos se refiere a la adición de más nodos en un sistema en lugar de aumentar la capacidad de un solo nodo. Con este enfoque, garantizar la integridad referencial en un entorno distribuido se vuelve un desafío. Por ejemplo, en una base de datos de red social donde las relaciones entre usuarios son constantes, es importante que los arreglos de amigos de un usuario en un servidor no se desincronicen con otros registros en diferentes nodos. Para esto, se pueden utilizar transacciones distribuidas, asegurando que todas las bases de datos involucradas en una operación se actualicen simultáneamente.

La migración a sistemas NoSQL puede ser viabilizada en ciertos escenarios, especialmente cuando la escalabilidad es prioritaria y los datos no requieren estrictamente una estructura relacional. Por ejemplo, en una aplicación de análisis de datos en tiempo real, donde se requiere altas tasas de

lectura y escritura sin un esquema rígido, una base de datos NoSQL como MongoDB permite almacenar estructuras de documentos sin depender de claves foráneas. Sin embargo, esta decisión puede tener repercusiones en la coherencia de los datos, lo que puede ser problemático en aplicaciones que demandan un control riguroso, como en el ámbito financiero.

La *desnormalización* es una técnica que consiste en introducir redundancias en los datos para optimizar el rendimiento. Al adoptar un esquema desnormalizado, se integran campos de tablas relacionadas en una sola tabla. Un ejemplo sería en un sistema de gestión de inventario, donde en vez de tener múltiples tablas para productos, proveedores y pedidos, se podría crear una tabla que contenga todos esos campos. Aunque esto facilita el acceso a información, puede incrementar el riesgo de inconsistencias. Es necesario implementar mecanismos de control y actualización de datos, como disparadores que aseguren que las modificaciones en un producto se reflejen en todos los registros de la tabla desnormalizada.

El uso de almacenamiento en caché también contribuye a mejorar el rendimiento de las bases de datos relacionales. Almacenar en caché los resultados de consultas frecuentes permite un acceso más rápido sin realizar constantes interrogaciones a la base de datos. Por ejemplo, en un sistema de búsqueda de hoteles, las búsquedas comunes por regiones pueden ser almacenadas en caché. Este enfoque reduce la carga sobre la base de datos subyacente y optimiza los tiempos de respuesta. Sin embargo, se debe controlar la coherencia del caché, especialmente tras actualizaciones en la base de datos, para evitar la entrega de información obsoleta.

La adopción de un enfoque de bases de datos poliglotas permite a las organizaciones emplear distintos tipos de bases de datos según las necesidades específicas de cada aplicación. *Por ejemplo, una aplicación de gestión de recursos humanos podría utilizar una base de datos relacional para almacenar información de empleados y relaciones jerárquicas, mientras que una base de datos NoSQL podría ser utilizada para almacenar archivos de medios o datos no estructurados*. **Esta estrategia requiere un diseño arquitectónico cuidadoso, donde las interacciones entre diferentes sistemas deben ser planificadas para asegurar la integración y el flujo de datos entre ellos**.

Cada una de estas estrategias y soluciones debe ser evaluada y seleccionada en función de las demandas específicas del negocio. La gestión de claves primarias y foráneas forma una parte integral de este proceso, ya que la forma en que se estructuran y relacionan los datos puede impactar directamente en la escalabilidad, el rendimiento y la integridad de la base de datos frente a un crecimiento continuo y un incremento en la carga de trabajo.

RESUMEN

Las bases de datos relacionales son sistemas diseñados para almacenar, gestionar y recuperar información de forma estructurada y eficiente. Este tipo de bases de datos se basa en el modelo relacional, introducido por Edgar F. Codd en la década de 1970, y organiza los datos en tablas. **Cada tabla, compuesta por filas y columnas, representa una entidad** y sus atributos, lo que facilita la comprensión y el acceso a la información. Cada tabla dentro de una base de datos relacional se identifica por una clave primaria, que asegura la singularidad de cada registro. Esta clave permite establecer relaciones con otras tablas, creando así un conjunto de datos interconectados que reflejan relaciones del mundo real. Estas relaciones pueden ser de uno a uno, uno a muchos o muchos a muchos, según cómo se conectan los datos.

El Lenguaje de Consulta Estructurado (SQL) es utilizado para interactuar con bases de datos relacionales. SQL permite realizar diversas operaciones, como insertar, actualizar, eliminar y consultar datos. La capacidad para manejar grandes volúmenes de información y la adaptabilidad a diferentes tipos de datos contribuyen a su popularidad en entornos empresariales y diferentes aplicaciones.

Las bases de datos relacionales incluyen dos lenguajes clave: Lenguaje de Definición de Datos (DDL) y Lenguaje de Manipulación de Datos (DML). DDL se usa para definir y modificar la estructura de la base de datos. A través de comandos como CREATE, ALTER y DROP, se pueden crear tablas, modificar su estructura y eliminarlas. La sintaxis de los comandos DDL permite definir tipos de datos, restricciones y relaciones entre tablas para asegurar la integridad de la información almacenada. Por ejemplo, para crear una tabla llamada 'Productos' se podría usar el siguiente código:

```
CREATE TABLE Productos (
    id INT PRIMARY KEY AUTO_INCREMENT,
    nombre VARCHAR(100) NOT NULL,
    precio DECIMAL(10, 2) NOT NULL,
    stock INT DEFAULT 0,
    fecha_ingreso DATE
);
```

Cada tabla debe tener una clave primaria que garantice la unicidad de los registros. Además, se pueden establecer claves foráneas para mantener relaciones entre diferentes tablas. Por ejemplo, una tabla de 'Pedidos' puede contener la clave foránea 'producto_id' que referencia a 'id' en la tabla 'Productos', asegurando que cada pedido esté relacionado a un producto existente.

Por otro lado, el DML permite la manipulación directa de los datos dentro de esas estructuras. Los comandos DML incluyen SELECT, INSERT, UPDATE y DELETE, los cuales facilitan la inserción, actualización, eliminación y consulta de los datos. Utilizar DML correctamente es necesario para

mantener la información actualizada y responder a las necesidades cambiantes de la organización. Aquí un ejemplo de uso de DML para registrar una compra:

```
BEGIN TRANSACTION;

-- Verificamos el stock de un producto antes de procesar la compra.
SELECT stock FROM productos WHERE id = 101;

-- Si el stock es suficiente, realizamos la transacción.
UPDATE productos SET stock = stock - 1 WHERE id = 101;

-- Añadimos el registro de compra.
INSERT INTO compras (producto_id, usuario_id, fecha)
VALUES (101, 205, GETDATE());

IF @@ERROR <> 0
    ROLLBACK; -- En caso de error, revertimos los cambios.
ELSE
    COMMIT; -- Si todo ha ido bien, confirmamos la transacción.
```

La integridad referencial es una propiedad que asegura que las relaciones entre tablas se mantengan correctamente, evitando la existencia de datos huérfanos o inconsistentes. Esta propiedad, junto con las características de acidez y consistencia en las transacciones, proporciona un marco sólido para gestionar datos en aplicaciones que requieren disponibilidad y fiabilidad.

Las bases de datos relacionales encuentran aplicación en diversos sectores, como el comercio electrónico y la gestión de recursos humanos, destacándose por su habilidad para mantener la coherencia y la integridad de los datos en sistemas complejos. Además, el diseño de las bases de datos que incluye la gestión de claves debe estar respaldado por procedimientos de mantenimiento, como el manejo de índices y la evaluación de su impacto en el rendimiento. A medida que se insertan o eliminan registros, el rendimiento de las consultas puede verse afectado, y es importante mantener el sistema optimizado.

Los índices son estructuras que permiten acelerar la búsqueda y recuperación de datos en las tablas. Por ejemplo, un índice en el campo `apellido` en una tabla de empleados hace que las consultas sean más eficientes. Sin embargo, **es necesario manejar con cuidado la creación de índices, ya que cada modificación de datos en la tabla también requiere la actualización del índice, lo que puede afectar el rendimiento en operaciones de inserción o modificación**. También son útiles para búsquedas de texto completo, mejorando la eficiencia de las consultas en grandes bloques de texto.

La seguridad en bases de datos relacionales incluye la implementación de mecanismos de autenticación y autorización que controlan el acceso a los datos. **Establecer roles de usuario y**

permisos específicos es importante para proteger la información sensible y cumplir con regulaciones sobre la protección de datos.

El modelado de datos relacional se centra en cómo se organiza la información en una base de datos y sigue un enfoque sistemático que abarca diversas etapas de conceptualización y diseño. Este proceso implica la creación de un modelo entidad-relación (ER), que luego se transforma en un modelo relacional apto para su uso en sistemas de gestión de bases de datos. **El modelado incluye la normalización**, q**ue busca eliminar redundancias y mejorar la integridad** de los datos dividiendo la información en diferentes tablas relacionadas.

Comprender el funcionamiento de las bases de datos relacionales, así como el DDL y DML, y saber gestionar adecuadamente las claves primarias y foráneas, son competencias necesarias para abordar con éxito los retos del desarrollo de aplicaciones que interactúan con datos estructurados. Este conocimiento permite a los diseñadores y desarrolladores crear aplicaciones robustas y eficientes que satisfacen las demandas del entorno digital actual.

[i] La experiencia aconseja que, independientemente de que una BBDD esté normalizada o no, los datos críticos se repartan en varias tablas por motivos de seguridad