UT_04.2_Diseño de bases de datos relacionales

Introducción

En el mundo actual, impulsado por los datos, almacenar y gestionar la información de forma eficiente es crucial para empresas y organizaciones de todos los tamaños. Las bases de datos relacionales se han convertido en una potente solución para organizar y manipular datos de forma estructurada y escalable. En esta entrada del blog, exploraremos los fundamentos de las bases de datos relacionales, sus sistemas de gestión y los principios que sustentan el diseño eficaz de bases de datos.

¿Qué es una base de datos?

Una base de datos es una colección estructurada de datos que se organiza y almacena de manera que facilite su recuperación, manipulación y gestión eficientes. Piensa en ella como un archivador digital, en el que, en lugar de carpetas y documentos físicos, tienes tablas y registros perfectamente organizados para facilitar el acceso.

¿Qué es una base de datos relacional?

Una base de datos relacional es un tipo de base de datos que organiza los datos en tablas (relaciones) con filas (registros) y columnas (campos). Estas tablas están interconectadas a través de relaciones, lo que permite acceder a los datos y combinarlos de diversas maneras. Imagine una colección de hojas de cálculo, cada una de las cuales representa un aspecto diferente de sus datos, pero con la capacidad de vincular y combinar información entre ellas a la perfección.

RDBMS

Un sistema de gestión de bases de datos relacionales (RDBMS) es una aplicación de software diseñada para crear, gestionar e interactuar con bases de datos relacionales. Proporciona un marco estructurado para almacenar, recuperar y manipular datos dentro de la base de datos. Algunos ejemplos populares de RDBMS son MySQL, PostgreSQL, Oracle y Microsoft SQL Server.

Introducción a SQL

SQL (lenguaje de consulta estructurado) es el lenguaje de programación estándar que se utiliza para interactuar con bases de datos relacionales. Permite crear, leer, actualizar y eliminar datos dentro de la base de datos, así como definir y modificar la propia estructura de la base de datos. SQL es como un lenguaje universal que permite comunicarse con diferentes plataformas RDBMS.

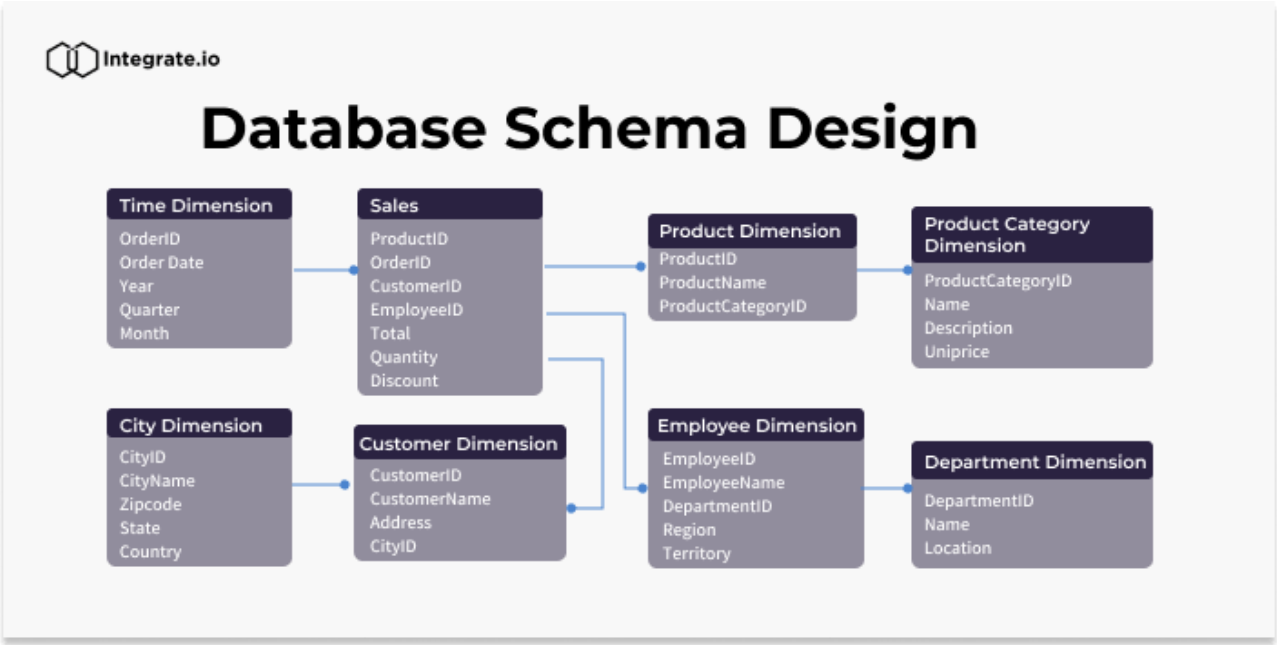
Convenciones de nomenclatura

En SQL, seguir convenciones de nomenclatura coherentes es fundamental para la claridad y la facilidad de mantenimiento. He aquí un ejemplo:

```
-- Buenas convenciones de nomenclatura
CREATE TABLE customers (
   customer_id INT PRIMARY KEY,
   first_name VARCHAR(100),
   last_name VARCHAR(100),
   email VARCHAR(100)
);
```

¿Qué es el diseño de bases de datos?

El diseño de bases de datos es el proceso de crear una estructura eficiente y organizada para almacenar y gestionar datos en una base de datos. Implica definir tablas, columnas, relaciones y restricciones para garantizar la integridad de los datos, minimizar la redundancia y optimizar el rendimiento. Un diseño adecuado de la base de datos es la base para crear aplicaciones robustas y escalables.



El primer paso para comenzar a trabajar sería crear la base de datos en nuestro SGBDD con el comando

`CREATE DATABASE mi_base_de_datos;`

Y posteriormente activarla con

`USE mi_base_de_datos`;

Integridad de los datos

La integridad de los datos se refiere a la precisión, coherencia y fiabilidad de los datos almacenados en una base de datos. Garantiza que los datos sigan reglas y restricciones específicas, evitando errores e incoherencias. Hay tres tipos de integridad de los datos:

1. **Integridad de la entidad**: garantiza que cada fila de una tabla sea identificable de forma única mediante una clave principal, y que la clave principal no pueda tener valores nulos.
2. **Integridad referencial**: mantiene las relaciones entre tablas garantizando que los valores de las claves externas de una tabla coincidan con los valores de las claves primarias de otra tabla.
3. **Integridad de dominio**: impone entradas válidas para una columna determinada restringiendo el tipo de datos, el formato y el rango de valores que se pueden almacenar.

```
-- Ejemplo: forzando integridad de los datos
CREATE TABLE orders (
    order_id INT PRIMARY KEY, -- Entity integrity
    customer_id INT FOREIGN KEY REFERENCES customers(customer_id), -- Referential
integrity
    order_date DATE NOT NULL, -- Domain integrity
    total_amount DECIMAL(10, 2) CHECK (total_amount >= 0) -- Domain integrity
);
```

Términos de bases de datos

- **Tabla**: Conjunto de datos relacionados organizados en filas y columnas.
- **Fila**: Una sola instancia o entrada en una tabla (también conocida como registro o tupla).
- **Columna**: Una característica o propiedad específica de los datos de una tabla (también conocida como campo o atributo).
- **Clave primaria**: Una columna o combinación de columnas que identifica de forma única cada fila de una tabla.
- **Clave externa**: Una columna o combinación de columnas que hace referencia a la clave primaria de otra tabla, estableciendo una relación entre las dos tablas.
- **Unión**: Operación que combina filas de dos o más tablas basándose en una columna relacionada.
- **Índice**: Estructura de datos que mejora el rendimiento de las operaciones de recuperación de datos mediante la creación de una representación ordenada de los datos de una tabla.
- **Vista**: tabla virtual que se genera dinámicamente a partir de una o varias tablas subyacentes.
- **Procedimiento almacenado**: colección precompilada de sentencias SQL que se pueden ejecutar como una sola unidad.
- **Desencadenador**: tipo especial de procedimiento almacenado que se ejecuta automáticamente cuando se produce un evento específico en una tabla, como una sentencia INSERT, UPDATE o DELETE.

Valores atómicos

En el diseño de bases de datos, es importante almacenar valores atómicos, lo que significa almacenar las piezas de información más pequeñas que no se pueden dividir más. Este principio ayuda a mantener la integridad de los datos y a evitar la redundancia.

Por ejemplo, en lugar de almacenar el nombre completo de un cliente en una sola columna, es mejor separarlo en columnas de nombre y apellido. De esta manera, se puede buscar, ordenar o manipular fácilmente cada parte del nombre de forma independiente.

```
-- Ejemplo: Almacenamiento de valores atómicos
CREATE TABLE customers (
    customer_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100)
);
```

Introducción a las claves

Las claves son componentes esenciales en el diseño de bases de datos que ayudan a garantizar la integridad de los datos y a establecer relaciones entre las tablas. Sirven como identificadores únicos para los registros y permiten una recuperación y manipulación eficientes de los datos.

Índice de clave primaria

Una clave primaria es una columna o una combinación de columnas que identifica de forma única cada registro de una tabla. Garantiza que cada registro sea único y se pueda localizar fácilmente. Las claves primarias suelen indexarse para mejorar el rendimiento de las consultas.

```
CREATE TABLE customers (
    customer_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100)
);
```

Tabla de consulta

Una tabla de consulta, también conocida como tabla de referencia o tabla de códigos, es una tabla que contiene conjuntos predefinidos de valores que se pueden utilizar para rellenar columnas en otras tablas. Las tablas de consulta ayudan a mantener la integridad de los datos al garantizar la coherencia y reducir la redundancia.

```
-- Tabla de consulta para categorías de productos
CREATE TABLE product_categories (
    category_id INT PRIMARY KEY,
    category_name VARCHAR(100)
);

-- Tabla de productos que hace referencia a la tabla de consulta
CREATE TABLE products (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(100),
    category_id INT,
    FOREIGN KEY (category_id) REFERENCES product_categories(category_id)
);
```

Superclave y clave candidata

Una superclave es un conjunto de una o más columnas de una tabla que identifica de forma única cada registro. Una clave candidata es una superclave mínima, lo que significa que no contiene columnas innecesarias. En otras palabras, una clave candidata es una superclave con el número mínimo de columnas necesarias para identificar de forma única cada registro.

Clave principal y clave alternativa

Una clave principal es una clave candidata elegida como identificador único principal de una tabla. Una clave alternativa, también conocida como clave única, es cualquier otra clave candidata que podría haber sido elegida clave principal, pero no lo fue.

```
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    email VARCHAR(100) UNIQUE, -- Alternate key
```

```
    first_name VARCHAR(50),
    last_name VARCHAR(50)
);
```

Clave sustituta y clave natural

Una clave sustituta es una clave artificial (a menudo un número secuencial o un GUID) que se utiliza como clave principal en una tabla. No tiene ningún significado inherente ni relación con los datos en sí. Una clave natural, por otro lado, es una clave derivada de los propios datos, como el ID de un empleado o el código de un producto.

```
-- Clave sustituta
CREATE TABLE orders (
    order_id INT PRIMARY KEY IDENTITY(1,1), -- Surrogate key
    customer_name VARCHAR(100),
    order_date DATE
);

-- Clave natural
CREATE TABLE products (
    product_code VARCHAR(10) PRIMARY KEY, -- Natural key
    product_name VARCHAR(100),
    price DECIMAL(10,2)
);
```

¿Debo usar claves sustitutas o claves naturales?

La elección entre claves sustitutas y claves naturales depende de varios factores, entre ellos la naturaleza de los datos, la probabilidad de que se produzcan cambios en los datos y la posibilidad de duplicación o conflictos.

A menudo se prefieren las claves sustitutas porque son:

- Inmutables: no cambian con el tiempo, incluso si los datos cambian.
- Únicas garantizadas: son generadas por el sistema de base de datos, lo que garantiza su unicidad.
- Opacas: no revelan ninguna información sobre los datos en sí, lo que puede ser beneficioso para la seguridad y la privacidad.

Las claves naturales, por otro lado, pueden ser ventajosas cuando:

- Los datos tienen una unicidad inherente, como los códigos de producto o los ID de los empleados.
- Es poco probable que los datos cambien con el tiempo, lo que reduce el riesgo de conflictos o duplicaciones.
- Se necesitan identificadores legibles y significativos para los humanos.

Clave externa

Una clave externa es una columna o una combinación de columnas de una tabla que hace referencia a la clave principal de otra tabla. Establece un vínculo entre las dos tablas y garantiza la integridad

referencial, asegurando que los datos de la tabla secundaria sean válidos y coherentes con los datos de la tabla principal.

```
CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    customer_id INT,
    order_date DATE,
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```

NOT NULL Clave externa

En algunos casos, puede ser conveniente aplicar una restricción NOT NULL a una columna de clave externa, lo que significa que la columna no puede tener un valor nulo. Esta restricción garantiza que todos los registros de la tabla secundaria estén asociados a un registro válido de la tabla principal.

```
CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    customer_id INT NOT NULL,
    order_date DATE,
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```

Restricciones de clave externa

Las restricciones de clave externa definen las reglas de integridad referencial entre tablas. Estas restricciones pueden incluir acciones que se deben realizar cuando se actualiza o elimina un registro referenciado en la tabla principal, como por ejemplo:

- CASCADE: cuando se actualiza o elimina un registro de la tabla principal, también se actualizan o eliminan los registros correspondientes de la tabla secundaria.
- SET NULL: cuando se actualiza o elimina un registro de la tabla principal, los valores de clave externa correspondientes de la tabla secundaria se establecen en NULL.
- NO ACTION: cuando se actualiza o elimina un registro de la tabla principal, los valores de clave externa correspondientes de la tabla secundaria permanecen sin cambios y la operación se revierte si viola la integridad referencial.

```
CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    customer_id INT,
    order_date DATE,
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id) ON UPDATE CASCADE ON
DELETE SET NULL
);
```

Clave simple, clave compuesta, clave compuesta

- Una clave simple es una sola columna que se utiliza como clave principal o clave externa.
- Una clave compuesta es una combinación de dos o más columnas que se utilizan como clave principal o clave externa.

- Una clave compuesta es una combinación de dos o más claves simples que se utilizan como clave externa.

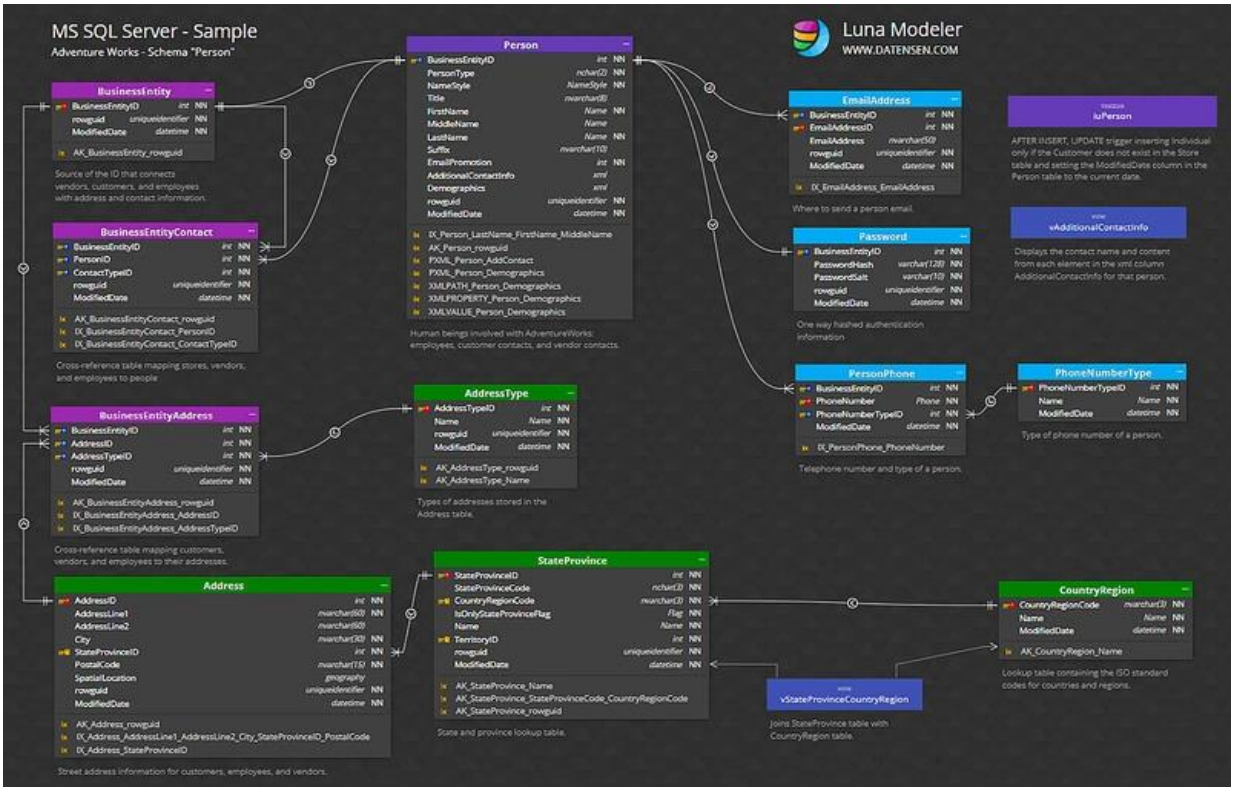
```
-- Clave simple
CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    ...
);

-- Clave compuesta
CREATE TABLE order_items (
    order_id INT,
    product_id INT,
    quantity INT,
    PRIMARY KEY (order_id, product_id)
);

-- Clave compuesta
CREATE TABLE shipments (
    shipment_id INT PRIMARY KEY,
    order_id INT,
    product_id INT,
    FOREIGN KEY (order_id, product_id) REFERENCES order_items(order_id, product_id)
);
```

Relaciones

Las relaciones son la piedra angular de las bases de datos relacionales, ya que permiten conectar y combinar datos de diferentes tablas. Existen tres tipos principales de relaciones:

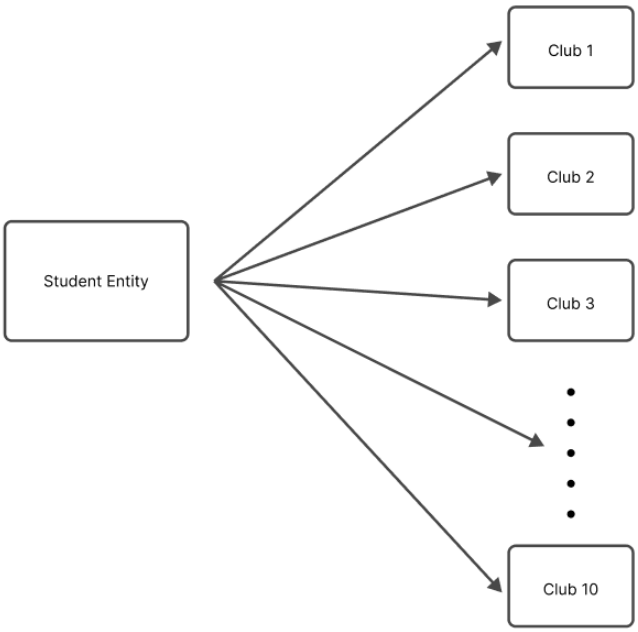


Relaciones uno a uno

En una relación uno a uno, cada registro de una tabla está asociado a un único registro de otra tabla, y viceversa. Por ejemplo, consideremos una base de datos en la que cada empleado tiene un único superior y cada superior gestiona a un único empleado. Este tipo de relación es relativamente poco frecuente en la práctica.

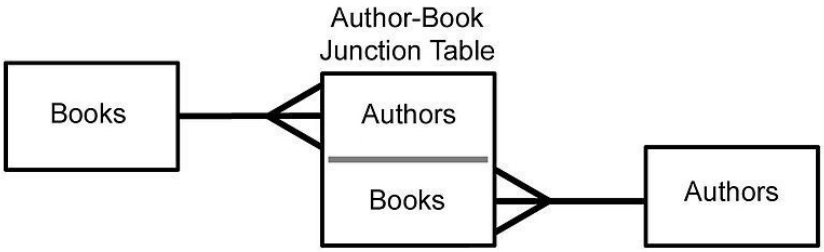
Relaciones uno a muchos

En una relación uno a muchos, cada registro de una tabla (el lado «uno») puede estar asociado a varios registros de otra tabla (el lado «muchos»). Por ejemplo, en una base de datos de una escuela, un profesor puede impartir varias clases, pero cada clase es impartida por un solo profesor.



Relaciones muchos a muchos

En una relación muchos a muchos, cada registro de una tabla puede asociarse con varios registros de otra tabla, y viceversa. Por ejemplo, en una base de datos de una universidad, un estudiante puede matricularse en varios cursos, y cada curso puede tener varios estudiantes matriculados.



Resumen de las relaciones

- Uno a uno: un registro de la tabla A está relacionado con un único registro de la tabla B, y viceversa.
- Uno a muchos: un registro de la tabla A puede estar relacionado con varios registros de la tabla B, pero un registro de la tabla B solo puede estar relacionado con un registro de la tabla A.
- Muchos a muchos: varios registros de la tabla A pueden estar relacionados con varios registros de la tabla B, y viceversa.

Diseño de relaciones (implementación SQL)

1 - Diseño de relaciones uno a uno

Para diseñar una relación uno a uno, puede incluir todas las columnas de ambas tablas en una sola tabla o crear dos tablas separadas y utilizar una restricción de clave externa para vincularlas.

```
-- Opción 1: tabla única
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(50),
```

```
    last_name VARCHAR(50),
    manager_first_name VARCHAR(50),
    manager_last_name VARCHAR(50)
);

-- Opción 2: Dos tablas con clave externa
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    manager_id INT UNIQUE,
    FOREIGN KEY (manager_id) REFERENCES employees(employee_id)
);
```

2 - Diseño de relaciones uno a muchos

Para diseñar una relación uno a muchos, normalmente se crean dos tablas: una tabla principal (el lado «uno») y una tabla secundaria (el lado «muchos»). La tabla secundaria incluye una columna de clave externa que hace referencia a la clave principal de la tabla principal.

```
-- Tabla principal
CREATE TABLE teachers (
    teacher_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50)
);

-- Tabla secundaria
CREATE TABLE classes (
    class_id INT PRIMARY KEY,
    class_name VARCHAR(100),
    teacher_id INT,
    FOREIGN KEY (teacher_id) REFERENCES teachers(teacher_id)
);
```

3 - Diseño de relaciones muchos a muchos

Para diseñar una relación muchos a muchos, normalmente se crea una tercera tabla (denominada tabla de unión o tabla asociativa) que vincula las dos tablas principales. Esta tabla de unión incluye columnas de clave externa que hacen referencia a la clave primaria

```
-- Tabla 1
CREATE TABLE students (
    student_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50)
);
```

```
-- Tabla 2
CREATE TABLE courses (
    course_id INT PRIMARY KEY,
    course_name VARCHAR(100),
    description TEXT
);
-- Tabla de unión o intermediaria
CREATE TABLE enrollments (
    enrollment_id INT PRIMARY KEY,
    student_id INT,
    course_id INT,
    FOREIGN KEY (student_id) REFERENCES students(student_id),
    FOREIGN KEY (course_id) REFERENCES courses(course_id)
);
```

Tablas padre y tablas hijo

En una relación uno a muchos o muchos a muchos, la tabla del lado **«uno»** se denomina a menudo tabla padre, mientras que la tabla del lado **«muchos»** se denomina tabla hijo. La tabla hijo contiene una clave externa que hace referencia a la clave principal de la tabla padre.

Por ejemplo, en la relación profesor-clase, la tabla `teachers` es la tabla principal y la tabla `classes` es la tabla secundaria. Del mismo modo, en la relación estudiante-curso, las tablas `students` y `courses` son tablas principales, mientras que la tabla `enrollments` es intermediaria.

Introducción al modelado de relaciones entre entidades

El modelado de relaciones entre entidades (modelado ER) es una técnica utilizada en el diseño de bases de datos para representar visualmente la estructura lógica de una base de datos. Ayuda a identificar las entidades (tablas), los atributos (columnas) y las relaciones entre ellos, lo que facilita la comprensión y la comunicación del diseño de la base de datos.

Los diagramas ER constan de los siguientes componentes:

- **Entidades**: representadas por rectángulos, las entidades son las tablas u objetos de la base de datos.
- **Atributos**: enumerados dentro del rectángulo de la entidad, los atributos son las columnas o campos que describen la entidad.
- **Relaciones**: representadas por líneas que conectan entidades, las relaciones describen las asociaciones entre entidades.

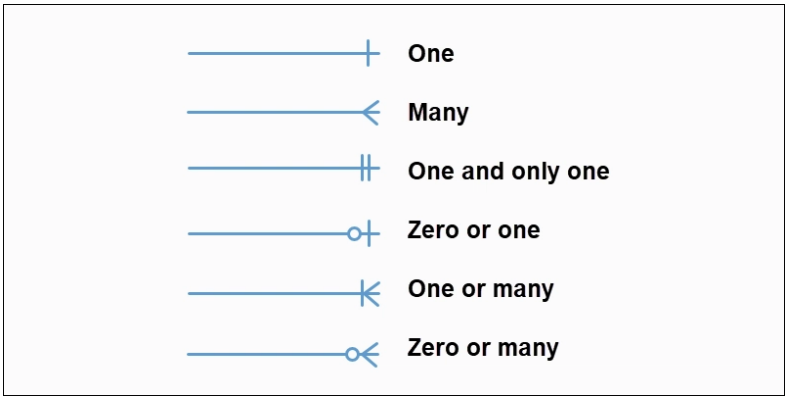
Cardinalidad

La cardinalidad define la relación numérica entre dos entidades. Especifica el número máximo de instancias de una entidad que se pueden asociar con una sola instancia de otra entidad. Las cardinalidades más comunes son:

- Uno a uno (1:1): una instancia de la entidad A puede asociarse con una instancia como máximo de la entidad B, y viceversa.

- Uno a muchos (1:N): una instancia de la entidad A puede asociarse con varias instancias de la entidad B, pero una instancia de la entidad B solo puede asociarse con una instancia de la entidad A.
- Muchos a muchos (M:N): varias instancias de la entidad A pueden asociarse con varias instancias de la entidad B, y viceversa.

En los diagramas ER, la cardinalidad se representa utilizando una notación específica, como una línea única para una a uno, una línea con una flecha para uno a muchos y una línea con flechas en ambos extremos para relaciones muchos a muchos.



Modalidad

La modalidad se refiere a si la existencia de una instancia de entidad depende de su relación con otra entidad. Hay dos tipos de modalidad:

- Modalidad parcial: la existencia de una instancia no depende de su relación con otra entidad. Por ejemplo, un cliente puede existir sin tener ningún pedido.
- Modalidad total: la existencia de una instancia depende de su relación con otra entidad. Por ejemplo, un artículo de pedido no puede existir sin un pedido.

En los diagramas ER, la modalidad se representa utilizando una notación específica, como una barra simple para la modalidad parcial y una barra doble para la modalidad total.

Introducción a la normalización de bases de datos

La normalización de bases de datos es el proceso de organizar los datos en una base de datos para reducir la redundancia, minimizar las anomalías de datos (anomalías de inserción, actualización y eliminación) y mejorar la integridad de los datos. Implica dividir una base de datos en tablas más pequeñas y definir las relaciones entre ellas basándose en reglas específicas o formas normales.

Los objetivos principales de la normalización de bases de datos son:

- Eliminar los datos redundantes
- Garantizar la integridad de los datos
- Facilitar la manipulación y el mantenimiento de los datos

Existen varias formas normales en la normalización de bases de datos, cada una de las cuales se basa en la anterior. Las formas normales más utilizadas son:

1. Primera forma normal (1NF)
2. Segunda forma normal (2NF)
3. Tercera forma normal (3NF)

1NF (Primera forma normal de normalización de bases de datos)

La primera forma normal (1NF) es la forma más básica de normalización. Establece que un atributo (columna) de una tabla debe tener valores atómicos, lo que significa que cada celda de la tabla debe contener un único valor, no un conjunto de valores.

Por ejemplo, consideremos una tabla con una columna llamada «Números de teléfono» que almacena varios números de teléfono de un cliente. Esto incumple la 1NF porque la columna contiene un conjunto de valores en lugar de un único valor. Para cumplir con la 1NF, sería necesario separar los números de teléfono en columnas individuales o crear una tabla separada para los números de teléfono.

```
-- Incumple la 1NF
CREATE TABLE customers (
    customer_id INT PRIMARY KEY,
    name VARCHAR(100),
    phone_numbers VARCHAR(200) -- Stores multiple phone numbers, violating 1NF
);

-- Cumple con 1NF
CREATE TABLE customers (
    customer_id INT PRIMARY KEY,
    name VARCHAR(100),
    phone1 VARCHAR(20),
    phone2 VARCHAR(20),
    phone3 VARCHAR(20)
);
```

2NF (Segunda forma normal de normalización de bases de datos)

La segunda forma normal (2NF) se basa en la 1NF y aborda el problema de las dependencias parciales. Una tabla está en 2NF si está en 1NF y todos los atributos no primarios (columnas) dependen completamente de la clave primaria completa.

En otras palabras, si una tabla tiene una clave primaria compuesta (formada por varias columnas), todas las columnas no clave deben depender de toda la clave primaria, no solo de una parte de ella.

Por ejemplo, consideremos una tabla con una clave primaria compuesta por `(student_id, course_id)` y una columna `grade`. Si la columna `grade` depende solo de `course_id` y no de la combinación de `course_id` y `student_id`, entonces la tabla incumple la 2NF.

```
-- Incumple la 2NF
CREATE TABLE student_courses (
    student_id INT,
    course_id INT,
    course_name VARCHAR(100), -- Depende solo de course_id, no de toda la clave primaria
    grade CHAR(2),
    PRIMARY KEY (student_id, course_id)
);
-- Cumple con la 2NF
CREATE TABLE student_courses (
```

```
    student_id INT,
    course_id INT,
    grade CHAR(2),
    PRIMARY KEY (student_id, course_id)
);

CREATE TABLE courses (
    course_id INT PRIMARY KEY,
    course_name VARCHAR(100)
);
```

3NF (tercera forma normal de normalización de bases de datos)

La tercera forma normal (3NF) se basa en la 2NF y aborda el problema de las dependencias transitivas. Una tabla está en 3NF si está en 2NF y todos los atributos no primarios son dependientes no transitivos de la clave primaria.

En otras palabras, si una columna no clave es dependiente de otra columna no clave, entonces la tabla viola la 3NF, y las columnas no clave deben separarse en su propia tabla.

Por ejemplo, consideremos una tabla con las columnas `student_id`, `student_name`, `class_id` y `class_name`. La columna `student_name` depende de la `student_id` y la columna `class_name` depende de la `class_id`. Sin embargo, la columna `class_name` también depende transitivamente de `student_id` a través de la columna `class_id`. Esto viola la 3NF.

```
-- Incumple la 3NF
CREATE TABLE student_classes (
    student_id INT,
    student_name VARCHAR(100),
    class_id INT,
    class_name VARCHAR(100),
    PRIMARY KEY (student_id, class_id)
);
-- Cumple con 3NF
CREATE TABLE students (
    student_id INT PRIMARY KEY,
    student_name VARCHAR(100)
);
CREATE TABLE classes (
    class_id INT PRIMARY KEY,
    class_name VARCHAR(100)
);
CREATE TABLE student_classes (
    student_id INT,
    class_id INT,
    PRIMARY KEY (student_id, class_id),
    FOREIGN KEY (student_id) REFERENCES students(student_id),
    FOREIGN KEY (class_id) REFERENCES classes(class_id)
```

```
);
```

Siguiendo los principios de normalización de bases de datos, se pueden crear bases de datos bien estructuradas y eficientes que minimizan la redundancia, mantienen la integridad de los datos y facilitan la manipulación y el mantenimiento de estos.

Índices (índices agrupados, no agrupados y compuestos)

Los índices son estructuras de datos que mejoran el rendimiento de las operaciones de recuperación de datos en una base de datos. Crean una representación ordenada de los datos de una tabla, lo que permite realizar búsquedas y consultas más rápidas. Existen varios tipos de índices:

- **Índice agrupado**: Determina el **orden físico** en el que se almacenan los datos en el disco.
    - Los datos de la tabla están **ordenados físicamente** según la columna del índice agrupado.
    - Solo puede haber **un único** índice agrupado por tabla.
    - Normalmente se crea automáticamente cuando defines una **PRIMARY KEY**, a menos que ya exista otro índice agrupado.
- **Índice no agrupado**: Es el tipo de índice **por defecto** en casi todos los motores de bases de datos.
    - El índice está **separado** de los datos reales de la tabla.
    - Contiene una copia de las columnas indexadas + un puntero (referencia) a la fila real en la tabla.
    - Es como el índice al final de un libro: te dice en qué página está cada tema, pero el contenido real está en otro lado.
- **Índice compuesto**: Es un índice que incluye **varias columnas juntas** (hasta 16 o 32 según el motor).
    - Se usa cuando tus consultas filtran o ordenan por **más de una columna al mismo tiempo**.

```
-- Índice agrupado
CREATE CLUSTERED INDEX idx_customers_name
ON customers (last_name, first_name);
-- Índice no agrupado
CREATE NONCLUSTERED INDEX idx_orders_date
ON orders (order_date);
-- Índice compuesto
CREATE INDEX idx_products_category_price
ON products (category_id, price);
```

Tipos de datos

En el mundo de las bases de datos, los tipos de datos son como diferentes formas de contenedores que almacenan tipos específicos de información. Del mismo modo que no se almacenarían líquidos en una cesta ni objetos sólidos en un frasco, las bases de datos deben aplicar tipos de datos específicos para garantizar la integridad y la coherencia de los datos.

Algunos tipos de datos comunes en SQL son:

- `INT` : Almacena números enteros, como `42 o 17`
- `FLOAT` : Almacena números decimales, como `3.14159 o 0.00005`
- `VARCHAR` : Almacena datos de texto, como nombres o descripciones.
- `DATE` : Almacena valores de fecha y hora, como `'2023-05-06' o '2024-01-01 12:34:56'`
- `BOOLEAN`: Almacena valores verdaderos/falsos, como (verdadero) o (falso) o `1 y 0`

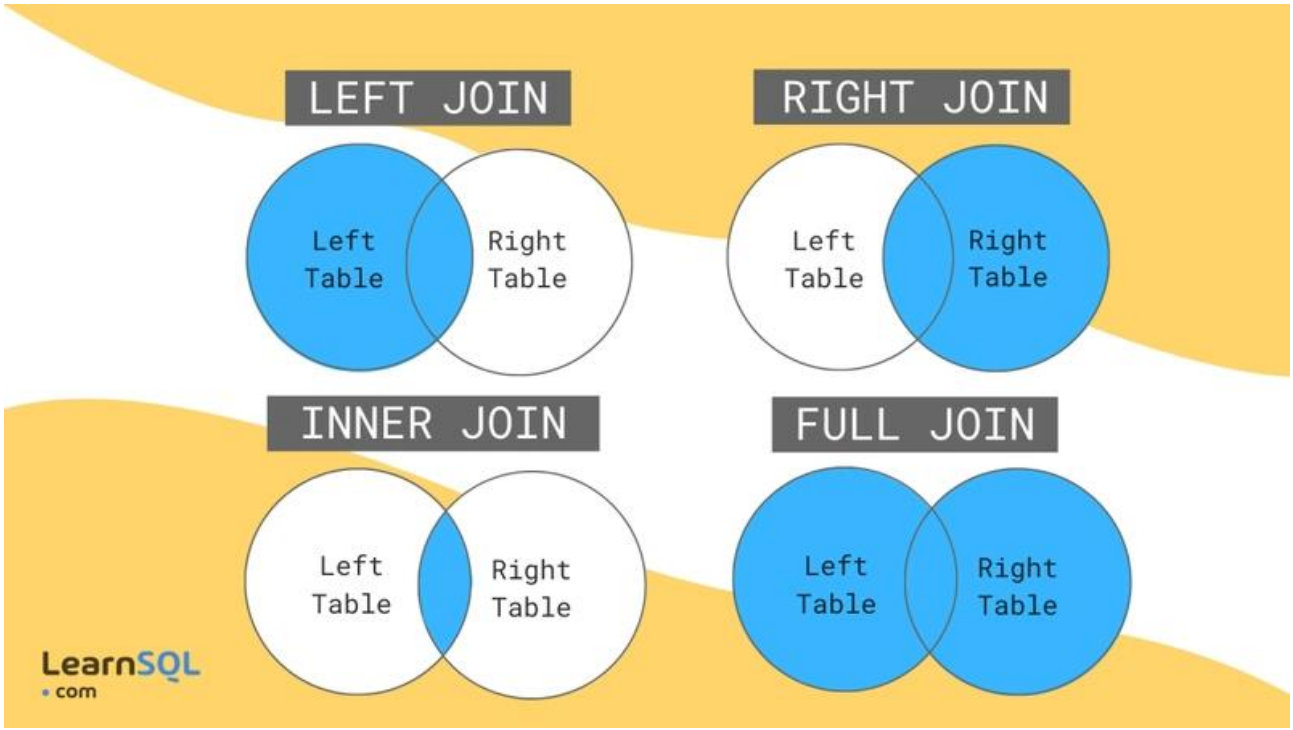
Elegir el tipo de datos adecuado es fundamental, ya que afecta a la forma en que se almacenan, consultan y manipulan los datos. Por ejemplo, intentar almacenar una cadena grande en una columna daría lugar a un error o a un truncamiento de datos con INT

```
CREATE TABLE users (
    id INT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    age INT,
    is_active BOOLEAN DEFAULT 1
);
```

En este ejemplo, creamos una tabla `users` con columnas `id` (entero), `name` (cadena de hasta 50 caracteres), `age` (entero) y `is_active` (booleano, con un valor predeterminado de 1 o verdadero).

Introducción a las uniones

Las uniones son como puentes que conectan diferentes tablas en una base de datos, lo que le permite combinar y recuperar datos relacionados de múltiples fuentes. Son un concepto fundamental en las bases de datos relacionales y son esenciales para consultar y manipular datos de manera eficiente.



Unión interna

Una unión interna es como un apretón de manos amistoso entre dos tablas, en el que solo se incluyen en el conjunto de resultados las filas que tienen valores coincidentes en ambas tablas. Es una forma de combinar datos de varias tablas basándose en una columna o conjunto de columnas comunes.

```
SELECT users.name, orders.order_date
FROM users
INNER JOIN orders ON users.id = orders.user_id;
```

En este ejemplo, recuperamos la columna `name` de la tabla `users` y la columna `order_date` de la tabla, `orders` pero solo para las filas en las que coincide el id de la tabla `users` con el `user_id` de la tabla `orders.`

Inner Join en 3 tablas (ejemplo)

Supongamos que tenemos tres tablas: `users, orders y products`. Queremos recuperar el nombre del usuario, la fecha del pedido y el nombre del producto para cada pedido. Podemos lograrlo realizando una unión interna entre las tres tablas:

```
SELECT users.name, orders.order_date, products.product_name
FROM users
INNER JOIN orders ON users.id = orders.user_id
INNER JOIN products ON orders.product_id = products.id;
```

Aquí, primero unimos las tablas y en las columnas y . A continuación, unimos el resultado de esa unión con la tabla en las columnas y . De esta manera, podemos recuperar datos de las tres tablas en una sola consulta, pero solo para las filas que cumplen las condiciones de unión.

Introducción a las uniones externas

Mientras que las uniones internas son como un apretón de manos amistoso, las uniones externas son más bien como un abrazo de bienvenida. Incluyen no solo las filas coincidentes de ambas tablas, sino también las filas no coincidentes de una o ambas tablas, dependiendo del tipo de unión externa.

Unión externa derecha

Una unión externa derecha es como un cálido abrazo de la tabla derecha a la tabla izquierda. Incluye todas las filas de la tabla derecha, junto con las filas coincidentes de la tabla izquierda. Si no hay filas coincidentes en la tabla izquierda, el resultado contendrá valores NULL para las columnas de la tabla izquierda.

```
SELECT users.name, orders.order_date
FROM users
RIGHT OUTER JOIN orders ON users.id = orders.user_id;
```

En este ejemplo, recuperamos todas las filas de la tabla `orders` (la tabla derecha), junto con los valores name coincidentes de la tabla `users` (la tabla izquierda). Si un pedido no tiene un usuario coincidente, la columna `name` contendrá `NULL`.

JOIN con columnas NOT NULL

A veces, es posible que desee realizar una unión solo en columnas que no sean nulas. Esto puede ser útil cuando desea excluir filas con datos faltantes del conjunto de resultados.

```
SELECT users.name, orders.order_date
```

```
FROM users
INNER JOIN orders ON users.id = orders.user_id AND users.name IS NOT NULL;
```

En este ejemplo, realizamos una unión interna entre las tablas `users` y`orders`, pero añadimos una condición adicional `users.name IS NOT NULL` para garantizar que solo se incluyan en el conjunto de resultados las filas con valores `name` no nulos.

Unión externa entre tres tablas

De forma similar al ejemplo de la unión interna, podemos realizar uniones externas entre varias tablas. Supongamos que queremos recuperar todos los pedidos, junto con el nombre del usuario y el nombre del producto, incluso si hay valores que faltan en las tablas `users` o `products`.

```
SELECT users.name, orders.order_date, products.product_name
FROM orders
LEFT OUTER JOIN users ON orders.user_id = users.id
LEFT OUTER JOIN products ON orders.product_id = products.id;
```

Aquí, comenzamos con la tabla `orders` y realizamos una unión externa izquierda con las tablas `users` y `products`. Esto garantiza que todos los pedidos se incluyan en el conjunto de resultados, junto con los nombres de usuario y los nombres de productos coincidentes, si están disponibles. Si no hay filas coincidentes en las tablas `users` o `products`, las columnas respectivas contendrán valores `NULL`.

Alias

Los alias son como apodos para tablas o columnas en consultas SQL. Pueden hacer que las consultas sean más legibles y fáciles de entender, especialmente cuando se trata de nombres largos de tablas o columnas, o cuando se hace referencia a la misma tabla varias veces en una consulta.

```
SELECT u.name, o.order_date, p.product_name
FROM users u
INNER JOIN orders o ON u.id = o.user_id
INNER JOIN products p ON o.product_id = p.id;
```

En este ejemplo, utilizamos los alias `u` para la tabla `users`, `o` para la tabla `orders` y `p` para la tabla `products`. Esto hace que la consulta sea más concisa y fácil de leer, sin tener que repetir los nombres completos de las tablas varias veces.

Auto unión

Una auto unión es como una tabla que mantiene una conversación consigo misma. Es una forma de unir una tabla consigo misma, basándose en una condición o relación específica dentro de la misma tabla. Esto puede ser útil cuando se trata de estructuras de datos jerárquicas o recursivas, como las relaciones entre empleados y gerentes o las categorías anidadas.

```
SELECT e.name AS employee, m.name AS manager
FROM employees e
LEFT OUTER JOIN employees m ON e.manager_id = m.id;
```

En este ejemplo, realizamos una auto unión en la tabla `employees` para recuperar el nombre de cada empleado y el nombre de su gerente correspondiente. Utilizamos una unión externa izquierda para garantizar que todos los empleados se incluyan en el conjunto de resultados, incluso si no tienen un gerente asignado.