UNIDAD 4 - Objetos y Tablas en SQL

Objetos de la Base de Datos

Comprendiendo los Objetos

Introducción a los Objetos de Base de Datos

Las bases de datos relacionales son complejas y contienen múltiples tipos de objetos que facilitan la gestión y manipulación de datos. Estos objetos son esenciales para el diseño, la implementación y el mantenimiento eficiente de bases de datos. A continuación, exploraremos los principales objetos de una base de datos relacional: tablas, vistas, procedimientos almacenados y claves.

Tablas

Las tablas son el corazón de cualquier base de datos relacional. Son estructuras que almacenan datos en filas y columnas:

- **Definición**: Una tabla se compone de columnas (también conocidas como campos) y filas (registros) donde cada columna tiene un tipo de dato definido y cada fila representa un registro único.
- **Uso**: Las tablas se utilizan para almacenar información de manera organizada. Por ejemplo, una tabla de empleados puede contener columnas para ID, nombre, posición y salario.
- **Relaciones**: Las tablas pueden relacionarse entre sí a través de claves primarias y foráneas para mantener la integridad de los datos.

Vistas

Las vistas son tablas virtuales creadas a partir de consultas sobre una o más tablas:

- **Definición**: Una vista es un objeto que muestra datos de una o varias tablas pero no almacena datos por sí misma, actúa como una ventana a datos almacenados en otras tablas.

- **Uso**: Las vistas se utilizan para simplificar consultas complejas, asegurar la seguridad de los datos presentando solo información relevante a usuarios específicos, y proporcionar una capa de abstracción sobre los datos subyacentes.
- **Tipos de vistas**: Existen vistas actualizables y no actualizables, dependiendo de si permiten modificaciones a los datos que reflejan.

Procedimientos Almacenados

Los procedimientos almacenados son conjuntos de instrucciones SQL que se ejecutan en la base de datos.

- **Definición**: Un procedimiento almacenado es un conjunto de comandos SQL precompilados que se almacena y se ejecuta en la base de datos. Estos procedimientos pueden contener instrucciones de control de flujo, operaciones condicionales, y transacciones completas. A diferencia de las funciones, no pueden ser llamados directamente dentro de una consulta SELECT, pero pueden ser ejecutados desde un cliente o dentro de otros procedimientos almacenados.
- **Uso**: Se utilizan para encapsular la lógica de negocios, realizar operaciones complejas, aplicar reglas de negocio, y gestionar transacciones. Su uso contribuye a mejorar la seguridad al controlar el acceso a los datos y puede aumentar la eficiencia de las aplicaciones al minimizar el tráfico entre el cliente y la base de datos.
- **Ventajas**: Los procedimientos almacenados permiten estandarizar procesos repetitivos, reducir errores, y simplificar el mantenimiento del código al centralizar la lógica de negocio. Además, ofrecen mejoras en el rendimiento al aprovechar la compilación y optimización que realiza el servidor de la base de datos.

Funciones

Las funciones son un conjunto de instrucciones SQL que se ejecutan en la base de datos y están diseñadas para devolver un valor específico.

- **Definición**: Una función es un bloque de código SQL almacenado en la base de datos que, a diferencia de un procedimiento almacenado, siempre devuelve un valor de salida o retorno, que puede ser un valor escalar (como un número o una cadena) o un conjunto de resultados. Las funciones pueden ser invocadas dentro de consultas SQL, lo que las hace útiles para realizar cálculos o transformar datos directamente en las consultas.
- **Uso**: Se utilizan para simplificar y estandarizar las consultas al encapsular lógica de negocio recurrente, como cálculos, transformaciones de datos, o validaciones. Las funciones también facilitan la reutilización del código al permitir que las operaciones comunes sean definidas una vez y utilizadas en múltiples lugares en las consultas SQL.
- **Ventajas**: Las funciones ayudan a mantener el código más limpio y modular, ya que encapsulan lógica que se puede reutilizar fácilmente. Al ser incluidas directamente en las consultas SQL, pueden mejorar la legibilidad y eficiencia de estas consultas, al reducir la necesidad de operaciones repetitivas.

Triggers (Disparadores)

Los triggers, o disparadores, son conjuntos de instrucciones SQL que se ejecutan automáticamente en respuesta a ciertos eventos en la base de datos.

- **Definición**: Un trigger es un bloque de código SQL que se ejecuta automáticamente cuando ocurre un evento específico en la base de datos, como una inserción, actualización o eliminación de registros en una tabla. Los triggers están vinculados a eventos en tablas o vistas y pueden ser configurados para activarse antes o después de que el evento ocurra.
- **Uso**: Se utilizan para automatizar procesos de auditoría, mantener registros históricos, o ejecutar cálculos automáticos en la base de datos. Los triggers permiten que las reglas de negocio y validaciones sean aplicadas de manera consistente y automática sin la intervención del usuario.
- **Ventajas**: Los triggers permiten automatizar tareas repetitivas y garantizar que las reglas de negocio se apliquen de manera consistente. Son útiles para mantener la integridad de los datos y realizar operaciones complejas de manera automática sin necesidad de intervención manual.

Claves

Las claves son elementos esenciales en el diseño de bases de datos para asegurar la integridad y eficiencia del acceso a los datos:

- **Clave Primaria (Primary Key)**: Un campo único que identifica cada fila de una tabla.
- **Clave Foránea (Foreign Key)**: Un campo que identifica la relación entre dos tablas. La clave foránea de una tabla corresponde a la clave primaria de otra, formando una relación entre ambas.
- **Índices**: Estructuras adicionales que mejoran la velocidad de recuperación de los datos. No son claves, pero son esenciales para optimizar consultas.

Conclusión

Entender y utilizar correctamente estos objetos de base de datos es fundamental para cualquier profesional que trabaje con bases de datos SQL. Permiten una gestión eficaz de los datos y aseguran que las aplicaciones que dependen de estas bases de datos sean robustas, seguras y eficientes.

Fundamentos de Tablas SQL

Estructura de Tablas en SQL

Las tablas en SQL son el núcleo fundamental de cualquier base de datos relacional. Estas están compuestas de filas y columnas:

- **Columnas (Campos)**: Definen el tipo de datos que se puede almacenar (como INT, VARCHAR, DATE, etc.).
- **Filas (Registros)**: Cada fila en una tabla representa un conjunto de datos relacionados que corresponden a una entidad única.

La definición precisa de cada columna asegura la integridad y la correcta tipificación de los datos, facilitando operaciones como búsquedas, actualizaciones e inserciones.

Hay que tener en cuenta que los tipos de datos pueden variar dependiendo el motor y la edición del mismo.

Tipos de Tablas en SQL

1. **Tablas Transaccionales**:
    - Usadas en aplicaciones donde la integridad y la consistencia de los datos son críticas (como sistemas bancarios o de inventarios).
    - Soportadas por motores como InnoDB en MySQL, que ofrecen características como bloqueos de fila, transacciones y recuperación ante fallos.
2. **Tablas de Hechos**:
    - Centrales en sistemas de Business Intelligence y data warehousing.
    - Capturan eventos o transacciones del negocio, donde cada fila representa un evento y las columnas contienen datos cuantitativos y descriptivos del mismo.
    - Tipos de columnas en tablas de hecho:
        - **Aditivas**: Se pueden sumar a través de dimensiones (como ventas totales).
        - **Semi-aditivas**: Se pueden sumar para algunas dimensiones (como inventario al final del día).
        - **No-aditivas**: No se pueden sumar (como precios unitarios).

Importancia de las Claves en la Gestión de Datos

Las claves son esenciales en el manejo y diseño de bases de datos para mantener la integridad y mejorar el rendimiento de las consultas:

1. **Clave Primaria (Primary Key)**:
    - Identifica de manera única cada fila en una tabla.
    - Asegura que no existan duplicados y facilita una búsqueda rápida.
2. **Clave Foránea (Foreign Key)**:
    - Establece una relación entre dos tablas, donde la clave foránea de una tabla coincide con la clave primaria de otra.
    - Es crucial para mantener la integridad referencial, asegurando que las relaciones entre tablas sean lógicas y los datos sean consistentes.
3. **Índices**:
    - Estructuras adicionales que optimizan las recuperaciones de datos.
    - Aunque no son claves per se, actúan como tal al mejorar la velocidad de acceso a los datos.

- Se pueden definir índices únicos que actuaran en este caso como claves únicas en la tabla, asegurando que los registros no nulos que contenga no se van a repetir (claves candidatas a la hora de diseñar).

Conclusión

El correcto diseño y uso de tablas en SQL conlleva una comprensión detallada de su estructura y tipos, así como de la implementación estratégica de claves. Esto no solo optimiza el rendimiento y la eficiencia de la base de datos sino que también garantiza la integridad y seguridad de los datos críticos del negocio.

Tipos de Tablas en SQL

1. **Tablas Transaccionales**:
    - Usadas en aplicaciones donde la integridad y la consistencia de los datos son críticas (como sistemas bancarios o de inventarios).
    - Soportadas por motores como InnoDB en MySQL, que ofrecen características como bloqueos de fila, transacciones y recuperación ante fallos.
2. **Tablas de Hechos**:
    - Centrales en sistemas de Business Intelligence y data warehousing.
    - Capturan eventos o transacciones del negocio, donde cada fila representa un evento y las columnas contienen datos cuantitativos y descriptivos del mismo.
    - Tipos de columnas en tablas de hecho:
        - **Aditivas**: Se pueden sumar a través de dimensiones (como ventas totales).
        - **Semi-aditivas**: Se pueden sumar para algunas dimensiones (como inventario al final del día).
        - **No-aditivas**: No se pueden sumar (como precios unitarios).

Importancia de las Claves en la Gestión de Datos

Las claves son esenciales en el manejo y diseño de bases de datos para mantener la integridad y mejorar el rendimiento de las consultas:

1. **Clave Primaria (Primary Key)**:
    - Identifica de manera única cada fila en una tabla.
    - Asegura que no existan duplicados y facilita una búsqueda rápida.
2. **Clave Foránea (Foreign Key)**:
    - Establece una relación entre dos tablas, donde la clave foránea de una tabla coincide con la clave primaria de otra.
    - Es crucial para mantener la integridad referencial, asegurando que las relaciones entre tablas sean lógicas y los datos sean consistentes.
3. **Índices**:
    - Estructuras adicionales que optimizan las recuperaciones de datos.

- Aunque no son claves per se, actúan como tal al mejorar la velocidad de acceso a los datos.
- Se pueden definir índices únicos que actuaran en este caso como claves únicas en la tabla, asegurando que los registros no nulos que contenga no se van a repetir (claves candidatas a la hora de diseñar).

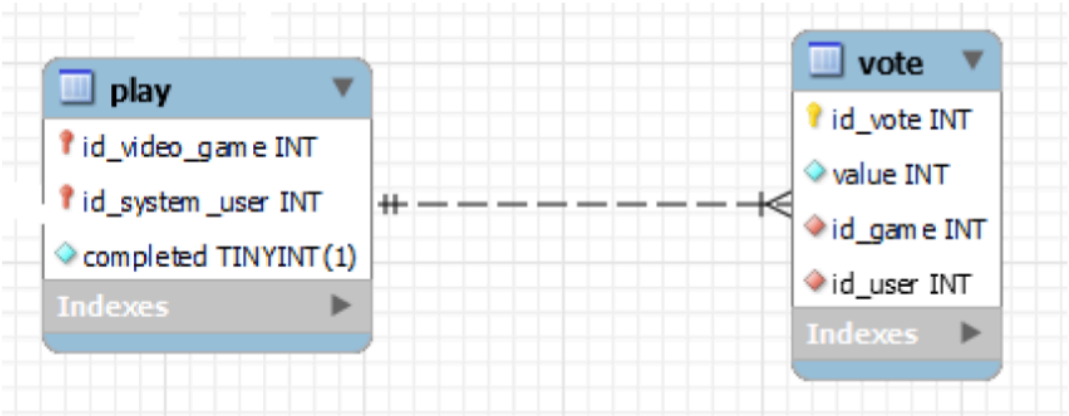
Conclusión

El correcto diseño y uso de tablas en SQL conlleva una comprensión detallada de su estructura y tipos, así como de la implementación estratégica de claves. Esto no solo optimiza el rendimiento y la eficiencia de la base de datos sino que también garantiza la integridad y seguridad de los datos críticos del negocio.

Tipos de Relaciones en Bases de Datos

Relación Uno-a-Uno

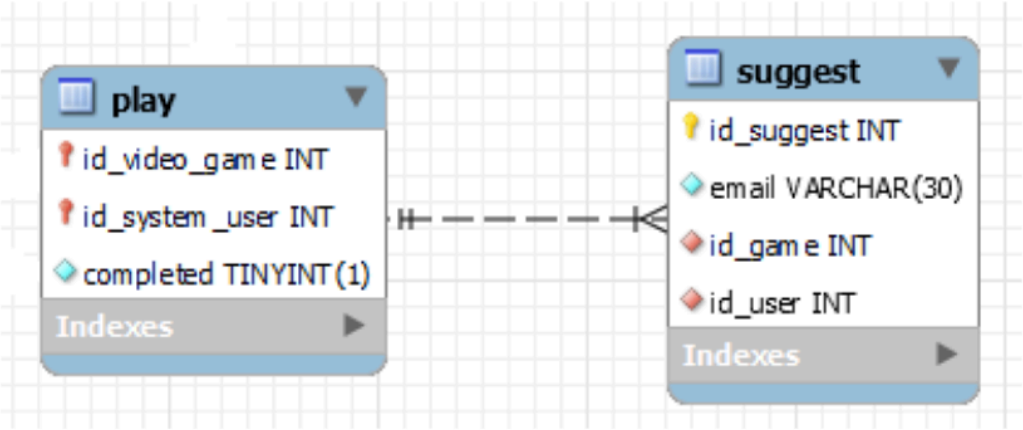
- **Descripción:** Cada registro, en cada tabla, aparece solo una vez, y tienen una relación unívoca.
- **Ejemplo Práctico:** En el ejemplo se muestra que para cada juego que juega un usuario puede realizar su voto una única vez



Relación Uno-a-Muchos

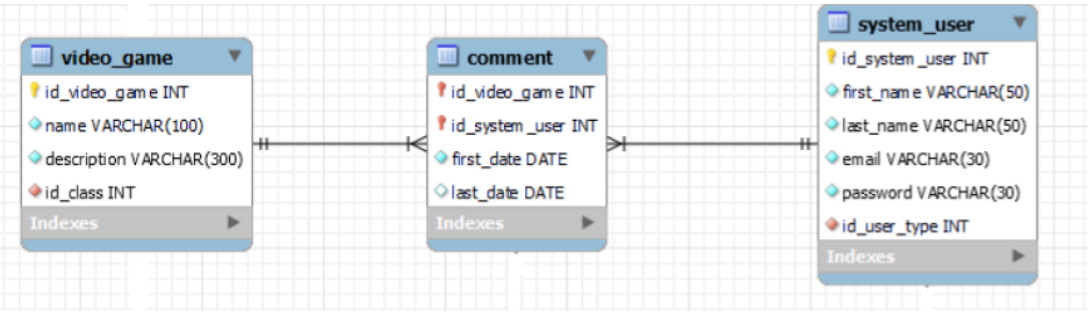
- **Descripción:** Un registro en una tabla puede tener relación con varios elementos de otra tabla.

- **Ejemplo Práctico:** En el ejemplo que se presenta se observa que un usuario de un sistema de juegos en línea es de un tipo (USER_TYPE) y a su vez pueden haber varios usuarios del mismo tipo.



Relación Muchos-a-Muchos

- **Descripción:** Uno o más registros en una tabla pueden tener relación con uno o más elementos de otra tabla. En la práctica no se puede implementar con base de datos la relación muchos a muchos, es por eso que se requiere de una tabla adicional (o tabla puente) que establezca una relación Uno-A-Muchos con cada tabla por separado y de esa forma dejar conformada la relación.
- **Ejemplo Práctico**



Implementación Práctica de Objetos

Diseño y Normalización de Tablas

Introducción a la Normalización y Diseño de Esquemas de Bases de Datos

El proceso de **normalización** es esencial para minimizar la redundancia y mejorar la integridad en el diseño de esquemas de bases de datos. A continuación, exploraremos los principios fundamentales de la normalización y cómo aplicarlos para diseñar esquemas eficientes y efectivos.

¿Qué es la Normalización?

La normalización es una técnica para diseñar esquemas de base de datos que organiza las tablas de acuerdo a ciertas reglas para minimizar la duplicación de datos y evitar anomalías de datos. El objetivo principal es estructurar la base de datos de manera que se garantice la integridad de los datos y que las dependencias entre ellos estén claramente definidas, optimizando así la coherencia y eficiencia del almacenamiento.

Formas Normales

- **Primera Forma Normal (1NF):** Asegura que en cada tabla, cada columna debe contener valores atómicos, y cada registro debe ser único. Esto implica la eliminación de conjuntos de datos repetitivos y el uso de claves primarias para identificar de manera única cada registro.

- **Segunda Forma Normal (2NF):** Se logra cuando la tabla está en 1NF y todos los atributos no clave dependen completamente de la clave primaria. En otras palabras, no debe haber dependencia parcial de la clave primaria en ningún atributo no clave, eliminando así cualquier redundancia funcional.
- **Tercera Forma Normal (3NF):** Una tabla se considera 3NF si está en 2NF y todos los atributos no clave son mutuamente independientes.
- **Forma Normal de Boyce-Codd (BCNF):** Es una versión más estricta de la 3NF, que se aplica cuando una tabla tiene más de una clave candidata. Una tabla está en BCNF si, para cada una de sus dependencias funcionales, el lado izquierdo de la dependencia es una superclave.
- **Cuarta Forma Normal (4NF):** Se logra cuando la tabla está en BCNF y no contiene dependencias multivaluadas, es decir, un atributo no clave no debe depender de más de un atributo no clave, evitando así que un registro contenga múltiples valores en un solo campo relacionado con diferentes aspectos de la entidad.
- **Quinta Forma Normal (5NF):** Una tabla está en 5NF si está en 4NF y no tiene ninguna dependencia de unión. Esto asegura que los datos no pueden descomponerse en otras tablas sin pérdida de información.

Diseño de Esquemas Eficientes

Un esquema de base de datos bien diseñado debe:

- **Reducir la redundancia:** Almacenar la misma información en múltiples lugares no solo aumenta el espacio de almacenamiento, sino que también incrementa el riesgo de inconsistencias y errores durante las actualizaciones. La normalización ayuda a eliminar esta redundancia, manteniendo los datos en su forma más sencilla y estructurada.
- **Aumentar la integridad de los datos:** Las restricciones de integridad, como claves foráneas, restricciones únicas, y valores predeterminados, ayudan a garantizar que los datos sean precisos, consistentes, y relacionados adecuadamente. Esto asegura que los cambios en una parte del sistema no introduzcan errores o inconsistencias en otras partes.
- **Mejorar el rendimiento:** Un diseño eficiente permite realizar consultas más rápidas y manejar mejor las operaciones de la base de datos.
- **Escalabilidad y Mantenimiento:** Un esquema bien diseñado facilita la escalabilidad de la base de datos, permitiendo agregar nuevas funcionalidades y ajustar el sistema con cambios mínimos. Además, una buena estructura de la base de datos simplifica el mantenimiento, la migración de datos, y la implementación de actualizaciones.

Técnicas de Diseño

1. **Identificación de entidades y atributos clave:** Identificar correctamente las entidades y sus interrelaciones es crucial para un modelo de datos efectivo.
2. **Uso de claves foráneas para integridad referencial:** Establecer relaciones correctas entre tablas para asegurar la consistencia de los datos.
3. **Índices para mejorar el rendimiento de las consultas:** Los índices son estructuras adicionales que mejoran la velocidad de recuperación de datos en una base de

datos. Permiten acelerar las consultas al proporcionar un acceso más rápido a los registros sin modificar los datos en sí.

Conclusiones

Aplicar la normalización y un diseño cuidadoso del esquema no solo mejora la eficiencia y el rendimiento de la base de datos, sino que también facilita su mantenimiento y escalabilidad a largo plazo. Aunque es importante equilibrar la normalización con la denormalización en ciertos casos, para optimizar el rendimiento en operaciones de lectura intensiva.

Las bases de datos altamente normalizadas pueden a veces requerir más operaciones de unión (joins), lo que puede ralentizar el rendimiento en consultas complejas. La denormalización, aunque aumenta la redundancia, puede ser útil en casos donde se necesita mejorar la velocidad de acceso a los datos.

Claves y Normalización

Conceptos Avanzados sobre Claves en SQL

Claves Primarias (Primary Keys)

Una clave primaria es un campo o conjunto de campos en una tabla que identifica de manera única cada registro. Es fundamental para mantener la integridad de los datos y garantizar que no existan duplicados.

Características principales:

- Solo puede haber una clave primaria por tabla.
- No puede contener valores nulos.
- Cada valor debe ser único.

Ejemplo:
Supongamos que tenemos una tabla de usuarios:

```
CREATE TABLE Usuarios (

    UsuarioID INT PRIMARY KEY,

    Nombre VARCHAR2(50),

    Email VARCHAR2(50)
```

);

En este caso, `UsuarioID` es la clave primaria. Cada usuario debe tener un `UsuarioID` único, que sirve como identificador en la tabla.

Uso en una consulta:

SELECT * FROM Usuarios WHERE UsuarioID = 101;

Esta consulta buscará al usuario con el `UsuarioID` igual a 101.

Claves Foráneas (Foreign Keys)

Una clave foránea establece una relación entre dos tablas, vinculando un campo de una tabla "hija" con la clave primaria de una tabla "padre". Esto asegura que los datos en ambas tablas estén relacionados de manera coherente.

Características principales:

- Garantiza la integridad referencial.
- Los valores en la clave foránea deben existir en la tabla "padre" o ser nulos.

Ejemplo:
Supongamos que tenemos una tabla de pedidos y una tabla de usuarios:

CREATE TABLE Usuarios (

UsuarioID INT PRIMARY KEY,

Nombre VARCHAR2(50)

);

CREATE TABLE Pedidos (

PedidoID INT PRIMARY KEY,

UsuarioID INT,

Fecha DATE,

FOREIGN KEY (UsuarioID) REFERENCES Usuarios(UsuarioID)

);

Aquí, `UsuarioID` en la tabla `Pedidos` es una clave foránea que referencia `UsuarioID` en la tabla `Usuarios`.

Uso en una consulta:

```
SELECT Pedidos.PedidoID, Usuarios.Nombre

FROM Pedidos

JOIN Usuarios ON Pedidos.UsuarioID = Usuarios.UsuarioID;
```

Esta consulta une ambas tablas para obtener los nombres de los usuarios junto con los pedidos que realizaron.

Claves Candidatas (Candidate Keys)

Las claves candidatas son campos o combinaciones de campos que pueden servir como clave primaria porque identifican de manera única cada registro. Una tabla puede tener múltiples claves candidatas, pero solo una de ellas se selecciona como clave primaria.

Ejemplo:
En una tabla de empleados, tanto el número de empleado como el correo electrónico podrían ser claves candidatas:

```
CREATE TABLE Empleados (

    EmpleadoID INT,

    Email VARCHAR2(100),

    Nombre VARCHAR2(50),

    PRIMARY KEY (EmpleadoID)

);
```

Aquí, `EmpleadoID` se selecciona como la clave primaria, pero `Email` podría haber sido otra opción válida como clave candidata.

Verificar unicidad de una clave candidata:

```
SELECT Email, COUNT(*)

FROM Empleados

GROUP BY Email

HAVING COUNT(*) > 1;
```

Esto verifica si el correo electrónico contiene duplicados y, por ende, no puede ser una clave válida.

Claves Concatenadas (Composite Keys)

Una clave concatenada o compuesta combina dos o más columnas para identificar de manera única un registro. Se usa cuando una sola columna no puede garantizar la unicidad.

Ejemplo:
En una tabla que almacena las calificaciones de los estudiantes para diferentes materias:

```
CREATE TABLE Calificaciones (
    EstudianteID INT,
    MateriaID INT,
    Calificacion DECIMAL(5, 2),
    PRIMARY KEY (EstudianteID, MateriaID)
);
```

Aquí, la combinación de `EstudianteID` y `MateriaID` es única, asegurando que cada estudiante tenga solo una calificación por materia.

Uso en una consulta:

```
SELECT *
FROM Calificaciones
WHERE EstudianteID = 1 AND MateriaID = 101;
```

Esto busca la calificación de un estudiante específico en una materia específica.

Al elegir y diseñar las claves, es fundamental analizar el contexto de los datos y las necesidades de consulta para crear una base de datos eficiente y funcional.

Material Complementario

Reverse Engineering con MySQL Workbench

- Reverse Engineering. Ingeniería inversa en MySQL Workbench | Alex Chasi
- Forward & Reverse Engineer MySQL Workbench | Alejandro Alfaro

Foreign key

- Cómo funcionan los FOREIGN KEY | Vida MRR - Diseño y desarrollo web

Cuándo usar una clave compuesta u ordenada

- Base de Datos - Relación Muchos a Muchos - Clave primaria compuesta (concatenada) | Programación y más