UD2

IMPLEMENTACIÓN DE BASES DE DATOS

MP_0484
Bases de Datos

2.3 Creación de bases de datos II

Introducción

Vamos a continuar profundizando en la implementación de bases de datos. En este punto en el que ya conocemos tanto los tipos de datos de MySQL como la instrucción básica (CREATE), profundizaremos en el uso de la herramienta, avanzaremos en el uso de comandos DDL (tanto ALTER como CREATE) y repasaremos algunos de los errores más comunes que nos encontramos en este paso, entre ellos las dificultades que nos presenta el paso de las relaciones de herencia desde el modelo E/R a la implementación final.

Modificar y eliminar estructuras

Veamos en primer lugar los comandos DDL ALTER y DROP.

Modificación de una tabla

ALTER TABLE nombre_tabla [opciones];

Estos son sus componentes principales:

- **nombre_tabla**: aquí se especifica el nombre de la tabla que queremos modificar.
- **opciones**: son las instrucciones específicas para modificar la estructura de la tabla. Las opciones más comunes incluyen:
    - **ADD**: Añadir una nueva columna o restricción a la tabla. Ejemplo:
      ALTER TABLE nombre_tabla ADD nueva_columna tipo_dato [restricción];
    - **DROP**: Eliminar una columna o restricción existente. Ejemplo:
      ALTER TABLE nombre_tabla DROP COLUMN nombre_columna;
    - **MODIFY**: Cambiar las propiedades de una columna, como el tipo de dato o su nombre. Ejemplo:
      ALTER TABLE nombre_tabla MODIFY nombre_columna nuevo_tipo_dato;
    - **RENAME**: Cambiar el nombre de la tabla. Ejemplo:
      ALTER TABLE nombre_tabla RENAME TO nuevo_nombre_tabla;

Eliminación de una tabla o una base de datos

DROP TABLE [IF EXISTS] nombre_tabla;

En este caso únicamente debemos introducir en **nombre_tabla** el nombre de la tabla que deseamos eliminar. Funciona de la misma manera con **DROP DATABASE**.

Herencia en SQL

Tal y como hemos visto anteriormente, se trata de relaciones que se usan para unificar entidades, agrupándolas en una entidad más general (generalización) o para dividir una entidad general en entidades más específicas (especificación). Aunque hoy en día todas estas relaciones suelen considerarse generalizaciones o también relaciones de herencia.

La implementación de relaciones de herencia en SQL no es trivial porque SQL se basa en el modelo relacional y no en el modelo entidad/relación. Sin embargo, es posible simular la herencia mediante diferentes enfoques. Los dos métodos más comunes son:

- **Herencia de tabla**
- **Herencia de clases de datos**

Estamos creando una base de datos para una tienda de coches y motocicletas usados. Viéndolo en su conjunto, solo venden vehículos que tienen atributos comunes (VIN, marca, modelo y potencia), sin embargo, los automóviles tienen atributos distintivos (como el número de puertas o la capacidad del maletero) que las motocicletas (como el estilo o el sidecar, figura 1). ¿Cómo podemos implementar esto usando SQL?

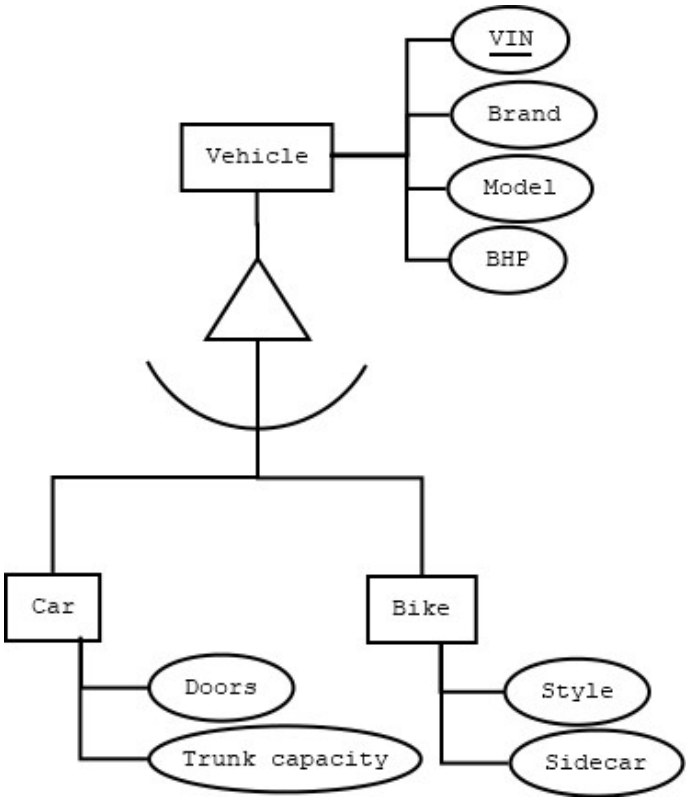

Figura 1. Modelo E/R con relación de herencia

Herencia de tabla única

Todos los atributos de las entidades se almacenan en una sola tabla. En la tabla se crean columnas para todos los aributos, muchas de las cuales pueden estar vacías (NULL).

```
CREATE TABLE Vehicle (
        VIN INT PRIMARY KEY,
        Brand VARCHAR(50),
        Model VARCHAR(50),
        NumberOfDoors INT, -- Specific attributes for cars
        Sidecar BOOLEAN -- Specific attributes for motorcycles
);
```

Esto permite consultas y actualizaciones más rápidas, pero desperdicia mucho espacio en tablas vacías. Además, la integridad de los datos puede ser más difícil de mantener.

Herencia de clases de datos

En este caso, se crea una tabla independiente para cada clase de la jerarquía de herencia. La tabla de una subclase incluye un FK que apunta a su tabla de superentidad

```
CREATE TABLE Vehicle (
        VIN INT PRIMARY KEY,
        Brand VARCHAR(50),
        Model VARCHAR(50),
);
CREATE TABLE Car (
        VIN INT PRIMARY KEY,
        Doors INT,
        FOREIGN KEY (ID) REFERENCES Vehicle(VIN)
);
CREATE TABLE Motorcycle (
        VIN INT PRIMARY KEY,
        Sidecar BOOLEAN,
        FOREIGN KEY (ID) REFERENCES Vehicle(VIN)
);
```

Esto ahorra espacio y proporciona mayor integridad de los datos, pero aumenta la complejidad y potencialmente ralentiza actualizaciones y consultas.

Errores más comunes

Estos son algunos de los errores que más se suelen repetir en las actividades:

- **Las claves foráneas** crean **relaciones**, **no las líneas.**
- Asegúrate de utilizar un tipo de **datos adecuado** para cada situación. Este es uno de los primeros pasos a la hora de garantizar la integridad de los **datos**.
- **El orden de creación** es importante. También será importante cuando se trabaje con datos.
- **La ingeniería inversa** debe **coincidir** con **el diseño**, a menos que lo mejores durante la implementación. En caso de que esto suceda, volvemos a revisar el diseño.

Asimismo, repasemos las relaciones n:n y las tablas intermedias (figura 2).

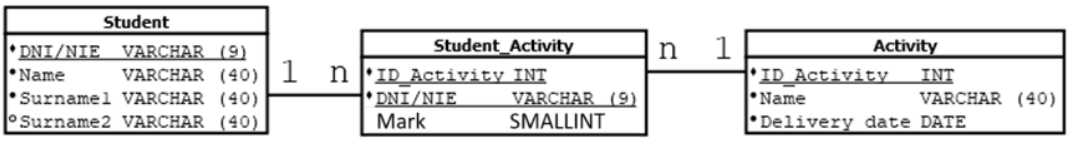

Figura 2. Modelo relacional con relación n:n

Para implementar estas relaciones, hemos de crear la propia tabla intermedia en SQL (figura 3):

```
CREATE TABLE Student (
    DNI_NIE VARCHAR(9),
    Name VARCHAR(40) NOT NULL,
    Surname1 VARCHAR(40) NOT NULL,
    Surname2 VARCHAR(40),
    CONSTRAINT PK_Student PRIMARY KEY (DNI_NIE)
);
CREATE TABLE Activity (
    ID_activity INT AUTO_INCREMENT,
    Name VARCHAR(40) NOT NULL,
    Delivery_date DATE NOT NULL,
    CONSTRAINT PK_Activity PRIMARY KEY (ID_activity)
);
CREATE TABLE Student_Activity (
    ID_activity INT,
    DNI_NIE VARCHAR(9),
    Mark SMALLINT,
    CONSTRAINT FK_Activity FOREIGN KEY (ID_activity) REFERENCES Activity(ID_activity),
    CONSTRAINT FK_Student FOREIGN KEY (DNI_NIE) REFERENCES Student(DNI_NIE)
);
```

Figura 3. Implementación de relación n:n en SQL