Desarrollo de aplicaciones web

Conoce el banco del futuro

JUAN GONZÁLEZ FERNÁNDEZ

15/12/2023

Índice

1. Introducción

2. Descripción del Modelo Entidad-Relación

3. Descripción del Modelo Relacional Normalizado

4. Implementación en SQL

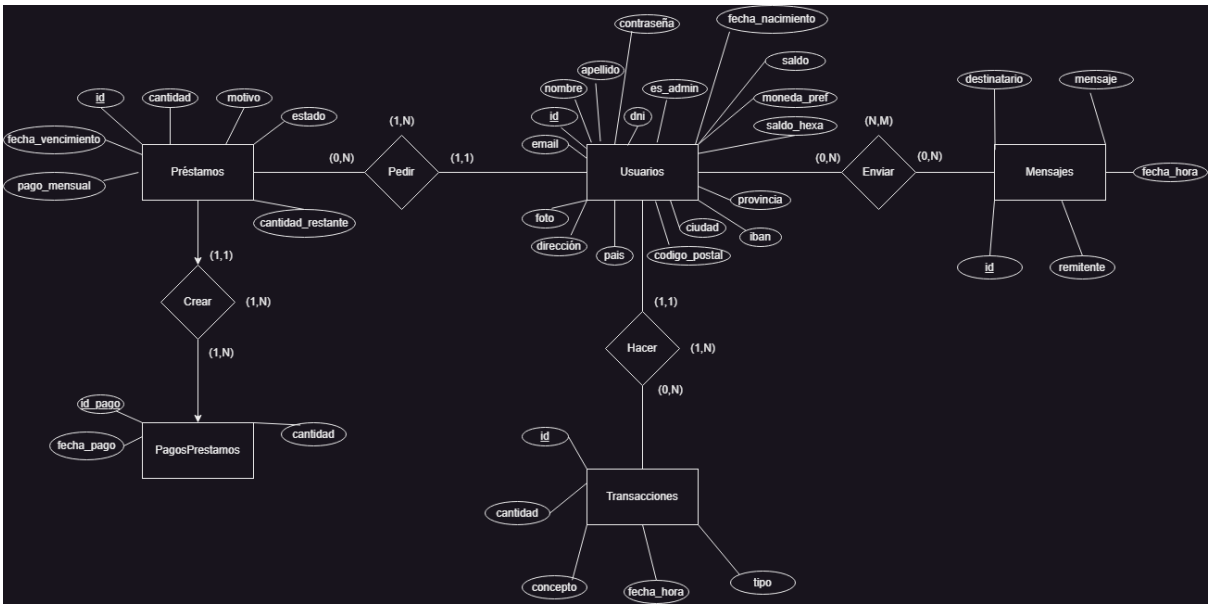
5. Conclusiones

1. Introducción

Este documento detalla el diseño y la implementación de la base de datos de “NEO”. Incluye la descripción de las tablas, sus atributos, y las relaciones entre ellas, siguiendo las reglas de normalización de datos para optimizar la integridad y el acceso a los datos. El documento también aborda la implementación práctica de este modelo en un sistema de gestión de base de datos relacional utilizando el lenguaje de definición de datos SQL.

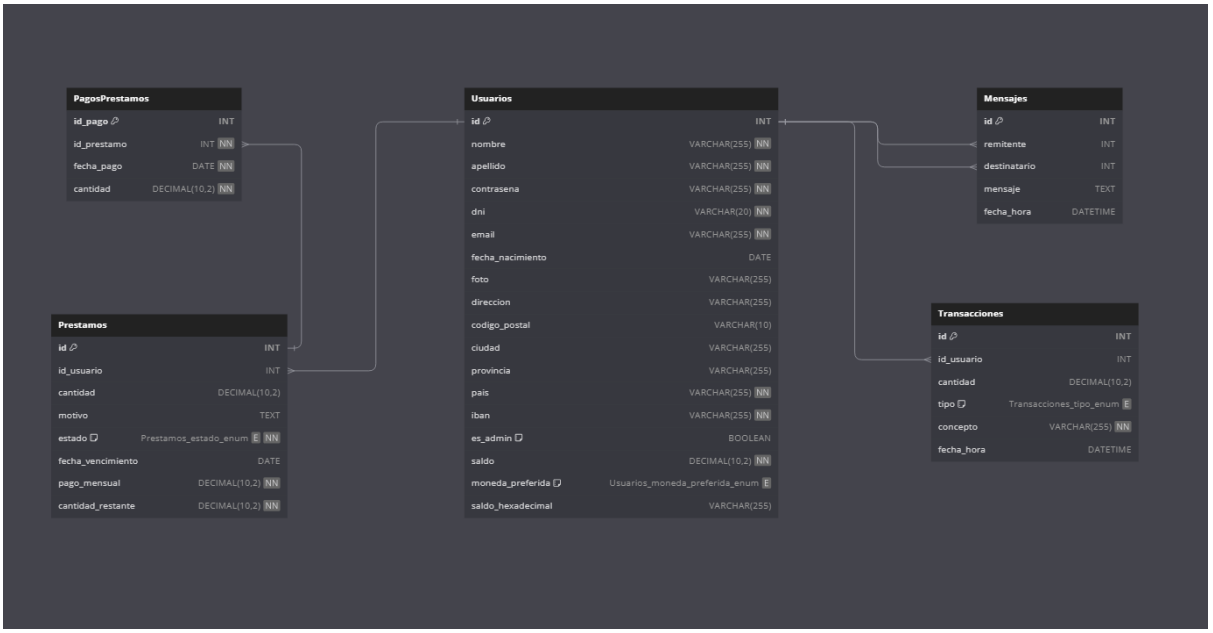
2. Descripción del Modelo Entidad-Relación

El diagrama de Entidad-Relación (E-R) explica cómo las entidades interactúan dentro del sistema. En el centro, tenemos la entidad “Usuarios”, que representa a los individuos que operan dentro del banco. Cada usuario puede estar asociado con múltiples “Préstamos“, como se denota por la relación 1:N, indicando que un solo usuario puede solicitar uno o más préstamos, pero cada préstamo está vinculado a un único usuario.
Por otro lado, la relación N:M entre “Usuarios“ y “Mensajes“ sugiere que un usuario puede enviar y recibir varios mensajes, y cada mensaje tiene un remitente y uno o varios destinatarios. De la misma forma, “Transacciones“ está en relación 1:N con “Usuarios“, reflejando que un usuario puede realizar varias transacciones, ya sea añadiendo o retirando fondos, pero cada transacción es realizada por un solo usuario.
La entidad “Préstamos“ está conectada a “PagosPrestamos” a través de una relación 1:N, donde un préstamo puede tener asociados varios pagos para amortizar la cantidad a deber, pero cada pago está directamente relacionado con un préstamo específico.



3. Descripción del Modelo Relacional Normalizado

Al analizar el modelo relacional normalizado, observamos que las tablas han sido diseñadas para minimizar la redundancia y garantizar la integridad de los datos. Cada tabla representa una entidad y está estructurada con claves primarias únicas, que identifican cada fila. Por ejemplo, la tabla “Usuarios” tiene una clave primaria id, que garantiza que cada usuario tenga un identificador único.
Las claves foráneas, como id_usuario en la tabla “Prestamos”, establecen una relación referencial con la tabla Usuarios, asegurando que cada préstamo esté ligado a un usuario existente en el sistema. Esta estructura normalizada ayuda a prevenir inconsistencias y facilita la actualización, inserción y eliminación de registros en la base de datos.
La normalización también se refleja en la separación de entidades y relaciones: “PagosPrestamos” se mantiene aparte de “Prestamos” para desacoplar los pagos individuales de los detalles del préstamo, y “Mensajes” se maneja de forma independiente para permitir una comunicación flexible entre usuarios.



4. Implementación en SQL

La implementación del modelo de base de datos en SQL se inicia con la creación de la base de datos. A continuación, se definen las tablas utilizando la declaración “CREATE TABLE”. Cada tabla tiene una serie de columnas con tipos de datos especificados, como VARCHAR para cadenas de texto, “DECIMAL” para cantidades monetarias, “INT” para números enteros, y “DATE” o “DATETIME” para fechas y timestamps.
Las claves primarias se definen con la cláusula “PRIMARY KEY”, y las claves foráneas con “FOREIGN KEY”, estableciendo las relaciones entre tablas como se especificó en el modelo relacional. Las restricciones “NOT NULL” y “UNIQUE” garantizan que ciertos campos sean obligatorios y únicos respectivamente, como el dni y el email en la tabla Usuarios.
La base de datos también incorpora en la tabla Usuarios un campo “es_admin” de tipo “BOOLEAN” para indicar si un usuario tiene privilegios administrativos, y un campo “moneda_preferida” que utiliza un tipo de dato ENUM para restringir los valores a un conjunto predefinido de monedas.
Cada inserción, como la del usuario 'admin' para que pueda gestionar la aplicación completa, es realizada mediante la declaración “INSERT INTO”, asignando valores específicos a cada uno de los campos definidos en las tablas. Esto demuestra cómo se pueden agregar registros al modelo ya estructurado.
Las demás tablas aplican lo mismo pero con diferentes variables cada una con una funcionalidad diferente a la otra.

```sql
DROP DATABASE IF EXISTS Proyecto_Finanzas;
CREATE DATABASE Proyecto_Finanzas;
USE Proyecto_Finanzas;
-- Crear tabla de Usuarios
CREATE TABLE Usuarios (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(255) NOT NULL,
    apellido VARCHAR(255) NOT NULL,
    contrasena VARCHAR(255) NOT NULL,
    dni VARCHAR(20) UNIQUE NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    fecha_nacimiento DATE,
    foto VARCHAR(255),
    direccion VARCHAR(255),
    codigo_postal VARCHAR(10),
    ciudad VARCHAR(255),
    provincia VARCHAR(255),
    pais VARCHAR(255) NOT NULL,
    iban VARCHAR(255) UNIQUE NOT NULL,
    es_admin BOOLEAN DEFAULT FALSE,
    saldo DECIMAL(10,2) NOT NULL,
    moneda_preferida ENUM('EUR', 'USD', 'JPY', 'GBP', 'RUB') DEFAULT 'EUR',
    saldo_hexadecimal VARCHAR(255)
);
```

```sql
CREATE TABLE Prestamos (
    id INT AUTO_INCREMENT PRIMARY KEY,
    id_usuario INT,
    cantidad DECIMAL(10,2),
    motivo TEXT,
    estado ENUM('pendiente', 'aprobado', 'rechazado', 'completado') NOT NULL,
    fecha_vencimiento DATE,
    pago_mensual DECIMAL(10,2) NOT NULL,
    cantidad_restante DECIMAL(10,2) NOT NULL,
    FOREIGN KEY (id_usuario) REFERENCES Usuarios(id)
);
```

```sql
CREATE TABLE Transacciones (
    id INT AUTO_INCREMENT PRIMARY KEY,
    id_usuario INT,
    cantidad DECIMAL(10,2),
    tipo ENUM('añadir', 'retirar'),
    concepto VARCHAR(255) NOT NULL,
    fecha_hora DATETIME,
    FOREIGN KEY (id_usuario) REFERENCES Usuarios(id)
);
```

```sql
CREATE TABLE Mensajes (
    id INT AUTO_INCREMENT PRIMARY KEY,
    remitente INT,
    destinatario INT,
    mensaje TEXT,
    fecha_hora DATETIME,
    FOREIGN KEY (remitente) REFERENCES Usuarios(id),
    FOREIGN KEY (destinatario) REFERENCES Usuarios(id)
);
```

```sql
CREATE TABLE Mensajes (
    id INT AUTO_INCREMENT PRIMARY KEY,
    remitente INT,
    destinatario INT,
    mensaje TEXT,
    fecha_hora DATETIME,
    FOREIGN KEY (remitente) REFERENCES Usuarios(id),
    FOREIGN KEY (destinatario) REFERENCES Usuarios(id)
);

CREATE TABLE PagosPrestamos (
    id_pago INT AUTO_INCREMENT PRIMARY KEY,
    id_prestamo INT NOT NULL,
    fecha_pago DATE NOT NULL,
    cantidad DECIMAL(10, 2) NOT NULL,
    FOREIGN KEY (id_prestamo) REFERENCES Prestamos(id)
);

INSERT INTO Usuarios (nombre, apellido, contrasena, dni, email, fecha_nacimiento, pais, iban, es_admin, saldo, moneda_preferida, saldo_hexadecimal)
VALUES ('admin', 'admin', 'admin', '12345678A', 'admin@admin.com', '2001-09-11', 'EEUU', 'IBANAdmin', TRUE, 11092001, 'EUR', '0');

-- LA CONTRASEÑA CIFRADA ES Juan2004 para acceder a la cuenta admin
INSERT INTO Usuarios (nombre, apellido, contrasena, dni, email, fecha_nacimiento, pais, iban, es_admin, saldo, moneda_preferida, saldo_hexadecimal)
VALUES ('admin', 'admin', '$2y$10$npFvBVlq.YK6MYKYCVa0r.TuGVim8WjwVIzbVno2A4b0JLuhLMfTa', '12342678A', 'admin2@admin.com', '2001-09-11', 'EEUU', 'IBANAadmin', TRUE, 11092001, 'EUR', '0');
```

5. Conclusiones

Al diseñar la base de datos, nos centramos en hacerla sólida y fácil de manejar. La normalización nos ayudó a evitar datos duplicados y a mantener todo en orden, lo que hace que la base de datos funcione rápido y sin errores.
Con nuestro modelo relacional, es fácil ver cómo todo se conecta, y al llevar esto a SQL, hemos creado un esquema que refleja estas conexiones. Esto significa que si "NEO" crece o cambia, su base de datos podrá adaptarse sin problemas.
En resumen, hemos hecho una base de datos que no solo funciona bien ahora, sino que está lista para lo que venga en el futuro, y eso es una gran ventaja para futuras implementaciones.