CardTraders

https://github.com/enoocdev/CardTradersBaseDeDato

Nome Alumno/a: Enooc Domínguez Quiroga

Enooc Domínguez Quiroga

Curso: ***1º DAM*** **Materia:** ***Bases de Datos – Proyecto Final 24/25***

Contido

1. Introducción

CardTraders es una plataforma digital diseñada para facilitar la compra, venta e intercambio de cartas coleccionables (TCG) entre usuarios, ofreciendo un entorno seguro, transparente y confiable. La plataforma actúa como intermediario validando la autenticidad de cartas premium, gestionando transacciones protegidas y ofreciendo herramientas como valoraciones de usuarios, catálogo en tiempo real y servicio de resolución de disputas. La misión de CardTraders es conectar a coleccionistas y jugadores de todo el mundo, haciendo que cada transacción sea sencilla y segura.

Esta concepción inicial del proyecto CardTraders establece las directrices fundamentales para el diseño y la implementación de su sistema de base de datos. Las funcionalidades descritas, como la intermediación en transacciones, la validación de autenticidad, la gestión de valoraciones y un catálogo dinámico, implican la necesidad de estructurar la información de manera que se puedan representar usuarios, cartas, las interacciones entre ellos (compra, venta, intercambio), así como los procesos de soporte como la validación y la resolución de disputas. Estos elementos son cruciales y anticipan las entidades y relaciones que conformarán el núcleo del modelo de datos del sistema.

2. Descripción del Problema / Requisitos

La presente sección detalla los problemas que el proyecto CardTraders busca solucionar y los requisitos funcionales y no funcionales que la base de datos debe satisfacer para dar soporte a la plataforma.

Problema a Resolver

El mercado de cartas coleccionables (TCG) presenta diversos desafíos para aficionados y jugadores. Entre ellos se encuentran:

- La dispersión de vendedores y compradores, dificultando encontrar contrapartes para transacciones específicas.
- La falta de plataformas centralizadas que ofrezcan un entorno seguro y confiable, lo que incrementa el riesgo de fraude o de recibir artículos que no se corresponden con lo acordado.
- La dificultad para verificar la autenticidad y el estado de conservación de cartas valiosas, especialmente en transacciones a distancia.
- La ausencia de mecanismos estandarizados y fiables para la resolución de disputas entre usuarios.

CardTraders se propone como una solución integral a estos problemas, actuando como un intermediario que aporta seguridad, transparencia y herramientas especializadas para la

comunidad de TCG. La base de datos es un componente esencial para materializar esta propuesta de valor, gestionando de forma eficiente y segura toda la información necesaria.

Requisitos Funcionales

Los requisitos funcionales definen las operaciones específicas que el sistema de base de datos debe permitir para soportar las funcionalidades de CardTraders, inferidas de su descripción general :

RF01: Gestión de Usuarios:

- Permitir el registro de nuevos usuarios con datos personales y de contacto.
- Gestionar el inicio y cierre de sesión de usuarios.
- Almacenar y permitir la actualización de perfiles de usuario, incluyendo información sobre su colección personal y reputación.

RF02: Gestión de Cartas y Catálogo:

- Permitir a los usuarios listar cartas para venta o intercambio.
- Mantener un catálogo detallado de cartas, incluyendo atributos como nombre, juego al que pertenece (p.ej., Magic: The Gathering, Pokémon, Yu-Gi-Oh!), edición, rareza, estado de conservación, descripción textual e imágenes.
- Facilitar la búsqueda y filtrado avanzado de cartas en el catálogo en tiempo real.

RF03: Autenticación de Cartas:

- Soportar un proceso para la validación de la autenticidad de cartas consideradas "premium" o de alto valor, registrando el estado de dicha validación.

RF04: Gestión de Transacciones:

- Registrar operaciones de compra, venta e intercambio de cartas entre usuarios.
- Almacenar los detalles de cada transacción, incluyendo las partes involucradas, las cartas objeto de la transacción, los precios acordados (si aplica) y las fechas.
- Gestionar el estado de las transacciones (p.ej., iniciada, en proceso, completada, cancelada).

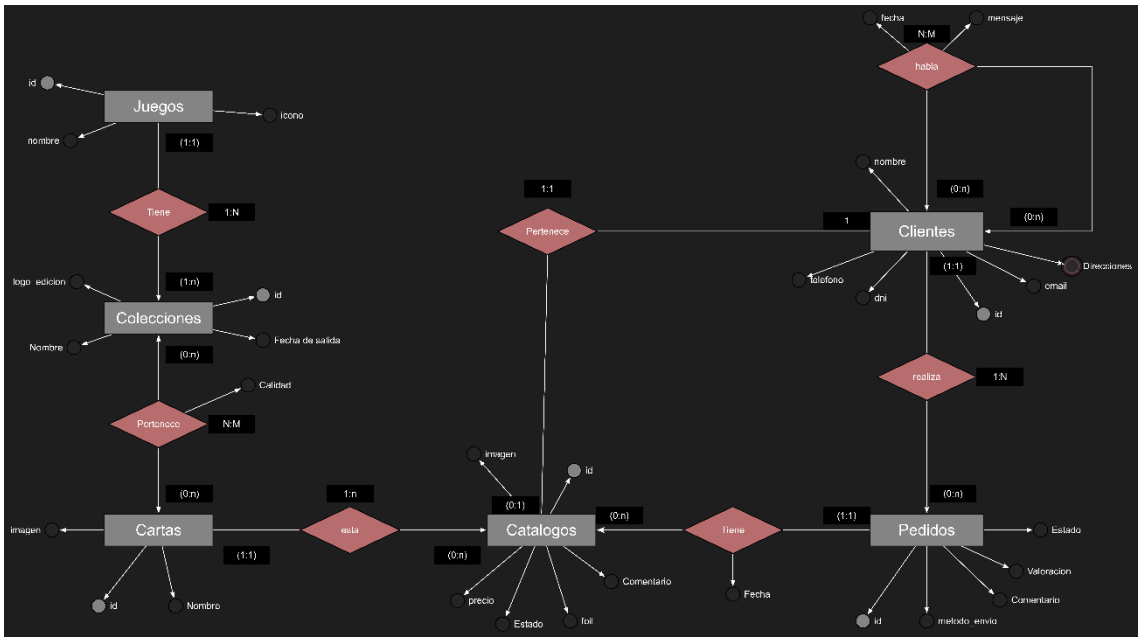
RF05: Sistema de Valoraciones y Reputación:

- Permitir a los usuarios valorar las transacciones completadas.
- Permitir a los usuarios valorar a otros usuarios con los que han interactuado.
- Calcular y actualizar la reputación de los usuarios basada en las valoraciones recibidas.

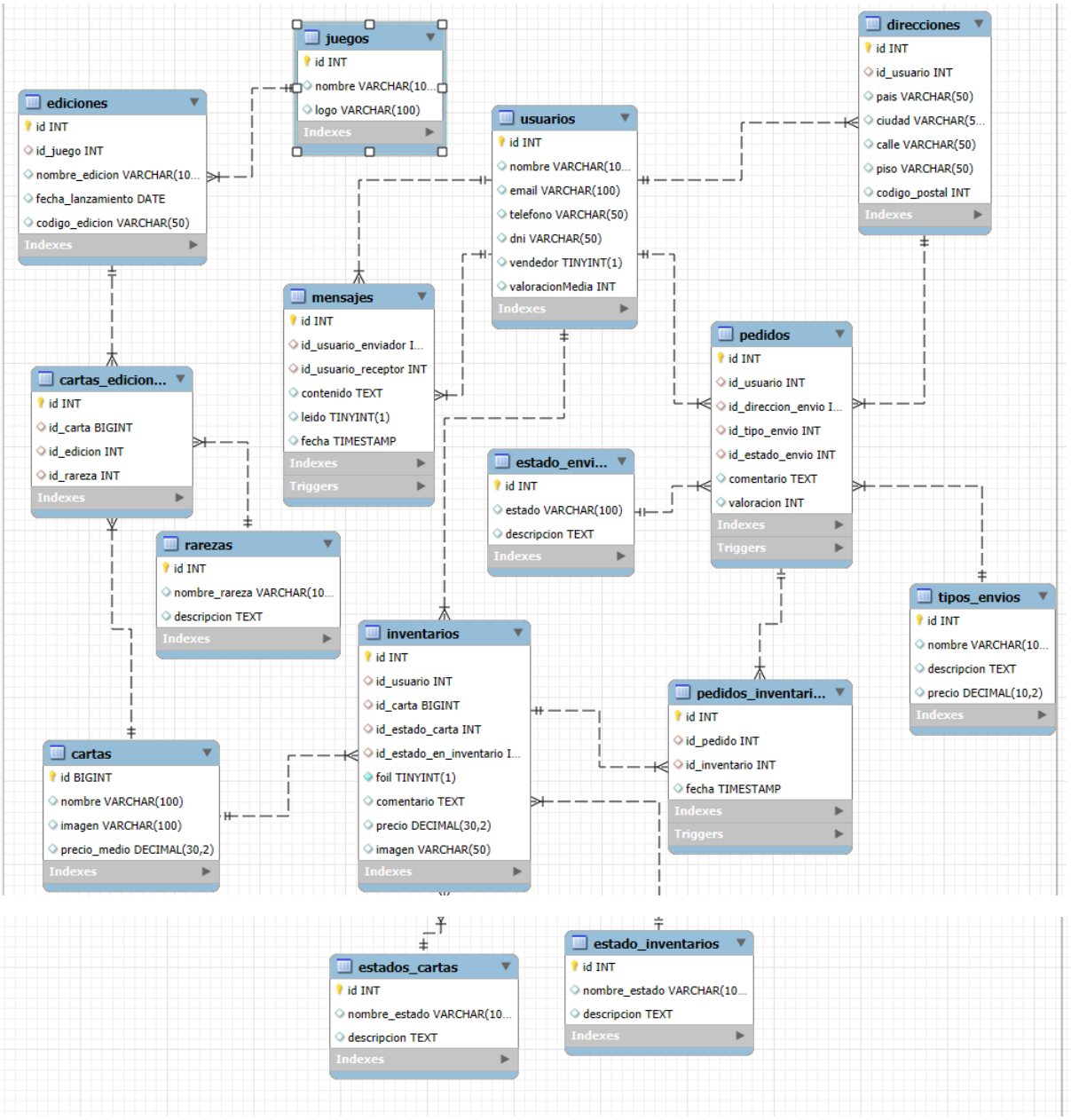
RF06: Resolución de Disputas:

- Soportar un sistema para que los usuarios puedan iniciar disputas relacionadas con transacciones.
- Registrar la información relevante de cada disputa, incluyendo las partes, el motivo, las evidencias presentadas y la resolución final.

3. Modelo Conceptual



4. Modelo Relacional



5. Proceso de Normalización

Para asegurar la robustez, eficiencia e integridad de la base de datos del proyecto, se llevó a cabo un riguroso proceso de normalización y refinamiento a partir del modelo conceptual inicial. Este proceso se centró en la aplicación de las Formas Normales y en la adecuada representación de las relaciones entre entidades.

En primer lugar, se garantizó la Primera Forma Normal (1NF) en toda la estructura. Esto implicó asegurar que cada tabla poseyera una clave primaria única para identificar sus registros y que todos los atributos contuvieran valores atómicos e indivisibles. Un ejemplo práctico fue la gestión del "listado" de artículos de un pedido, que en el modelo inicial podría considerarse un atributo multivaluado; esto se resolvió implementando la tabla de unión pedidos_inventarios, donde cada artículo del pedido constituye un registro independiente.

A continuación, se trabajó para alcanzar la Segunda Forma Normal (2NF), asegurando que todos los atributos no clave de cada tabla dependieran de forma completa de su clave primaria. Esta norma es crucial en tablas con claves compuestas, como las que surgen al resolver relaciones muchos a muchos. Por ejemplo, en la tabla cartas_ediciones, que vincula cartas con ediciones, el atributo id_rareza depende conjuntamente tanto de id_carta como de id_edicion, reflejando que la rareza es específica de una carta dentro de una edición particular. Del mismo modo, la creación de una entidad separada para direcciones, vinculada tanto a usuarios como a pedidos, evita redundancias y asegura que la información de la dirección dependa únicamente de su propia entidad y no parcialmente de un pedido o un usuario que podría tener múltiples direcciones o pedidos.

Posteriormente, se aplicó la Tercera Forma Normal (3NF) con el objetivo de eliminar las dependencias transitivas, es decir, que ningún atributo no clave dependiera de otro atributo que tampoco forma parte de la clave. Este paso se materializó principalmente mediante la creación de tablas de consulta o "lookup". Por ejemplo, en lugar de almacenar la descripción de una rareza directamente en cartas_ediciones (donde dependería del nombre de la rareza, un atributo no clave), se creó la tabla rarezas. De forma análoga, se establecieron las tablas estados_cartas, estado_inventarios, estado_envio y tipos_envios para centralizar la información descriptiva de estos catálogos (como nombres, descripciones o precios), siendo referenciados desde las tablas principales (inventarios, pedidos) mediante sus respectivos identificadores.

Un aspecto fundamental del diseño fue la resolución de las relaciones muchos a muchos (N:M). Estas relaciones, como la existente entre Colecciones (ahora ediciones) y Cartas, o entre Pedidos e Inventarios (originalmente Catalogos), no pueden representarse directamente en un modelo relacional. Por ello, se optó por la creación de tablas asociativas o de unión. Así, cartas_ediciones vincula cada carta con sus ediciones e incluye información específica de esa

relación, como la rareza. De igual manera, pedidos_inventarios detalla los artículos específicos que componen cada pedido, y mensajes gestiona la comunicación bidireccional entre usuarios.

Finalmente, durante este tránsito del modelo conceptual al físico, se llevó a cabo un refinamiento de las entidades y sus atributos para mejorar la claridad y precisión del esquema. Por ejemplo, la entidad Clientes se generalizó a usuarios, permitiendo una mayor flexibilidad para futuros roles. Colecciones se concretó como ediciones, un término más específico en el contexto de los juegos, y Catalogos evolucionó a inventarios, reflejando con más exactitud su función de registrar ítems específicos disponibles con todos sus detalles. La introducción de la entidad direcciones es otro ejemplo de este refinamiento, normalizando la gestión de las direcciones postales de los usuarios.

Este enfoque metodológico en el diseño ha permitido construir una base de datos estructurada, coherente y optimizada para las necesidades del proyecto.

6. Script de Creación de la Base de Datos

A continuación, se presenta el script SQL completo utilizado para la creación y definición de la estructura de la base de datos CardTraders. Este script incluye la creación de la base de datos, la definición de todas las tablas (usuarios, mensajes, direcciones, juegos, ediciones, cartas, rarezas, cartas_ediciones, estados_cartas, estado_inventarios, inventarios, tipos_envios, estado_envios, pedidos y pedidos_inventarios), sus columnas, tipos de datos, claves primarias, claves foráneas, restricciones de unicidad, valores por defecto y las acciones referenciales para el mantenimiento de la integridad de los datos.

```
--
================================================================================
=
-- Script de Creación de la Base de Datos: CardTraders
-- Descripción: Este script crea la base de datos CardTraders y todas sus tablas,
--              relaciones, e índices necesarios para el funcionamiento de la
--              plataforma de intercambio y gestión de cartas coleccionables.
--
================================================================================
=

CREATE DATABASE IF NOT EXISTS `CardTraders`;

USE `CardTraders`;
```

```
-- -----------------------------------------------------
-- Tabla: `usuarios`
-- Almacena la información de los usuarios registrados en la plataforma.
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `usuarios` (
    `id`                INT PRIMARY KEY AUTO_INCREMENT,
    `nombre`            VARCHAR(100),
    `email`             VARCHAR(100) UNIQUE,
    `telefono`          VARCHAR(50) UNIQUE,
    `dni`               VARCHAR(50) UNIQUE,
    `vendedor`          BOOLEAN DEFAULT FALSE,
    `valoracionMedia`   INT DEFAULT 0
) COMMENT = 'Información de los usuarios de la plataforma';


-- -----------------------------------------------------
-- Tabla: `mensajes`
-- Registra los mensajes intercambiados entre usuarios.
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mensajes` (
    `id`                    INT PRIMARY KEY AUTO_INCREMENT,
    `id_usuario_enviador`   INT,
    `id_usuario_receptor`   INT,
    `contenido`             TEXT,
    `leido`                 BOOLEAN,
    `fecha`                 TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT `fk_mensajes_usuario_enviador`
        FOREIGN KEY (`id_usuario_enviador`) REFERENCES `usuarios`(`id`)
        ON UPDATE CASCADE
        ON DELETE RESTRICT,
    CONSTRAINT `fk_mensajes_usuario_receptor`
        FOREIGN KEY (`id_usuario_receptor`) REFERENCES `usuarios`(`id`)
        ON UPDATE CASCADE
        ON DELETE RESTRICT
) COMMENT = 'Mensajes entre usuarios';
```

```
-- -----------------------------------------------------
-- Tabla: `direcciones`
-- Almacena las direcciones postales de los usuarios.
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `direcciones` (
    `id`                INT PRIMARY KEY AUTO_INCREMENT,
    `id_usuario`        INT,
    `pais`              VARCHAR(50),
    `ciudad`            VARCHAR(50),
    `calle`             VARCHAR(50),
    `piso`              VARCHAR(50),
    `codigo_postal`     INT,
    CONSTRAINT `fk_direcciones_usuario`
        FOREIGN KEY (`id_usuario`) REFERENCES `usuarios`(`id`)
        ON UPDATE CASCADE
        ON DELETE CASCADE
) COMMENT = 'Direcciones de envío de los usuarios';


-- -----------------------------------------------------
-- Tabla: `juegos`
-- Catálogo de los diferentes juegos de cartas disponibles.
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `juegos` (
    `id`        INT PRIMARY KEY AUTO_INCREMENT,
    `nombre`    VARCHAR(100),
    `logo`      VARCHAR(100)
) COMMENT = 'Catálogo de juegos de cartas';


-- -----------------------------------------------------
-- Tabla: `ediciones`
-- Define las distintas ediciones o expansiones de cada juego.
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ediciones` (
    `id`                    INT PRIMARY KEY AUTO_INCREMENT,
```

```
    `id_juego`              INT,
    `nombre_edicion`        VARCHAR(100),
    `fecha_lanzamiento`     DATE,
    `codigo_edicion`        VARCHAR(50),
    CONSTRAINT `fk_ediciones_juego`
        FOREIGN KEY (`id_juego`) REFERENCES `juegos`(`id`)
        ON UPDATE CASCADE
        ON DELETE RESTRICT
) COMMENT = 'Ediciones o expansiones de los juegos';


-- -----------------------------------------------------
-- Tabla: `cartas`
-- Contiene la información general de cada carta coleccionable.
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `cartas` (
    `id`                BIGINT PRIMARY KEY AUTO_INCREMENT,
    `nombre`            VARCHAR(100),
    `imagen`            VARCHAR(100),
    `precio_medio`      DECIMAL(30,2) DEFAULT 0
) COMMENT = 'Información general de las cartas coleccionables';


-- -----------------------------------------------------
-- Tabla: `rarezas`
-- Catálogo de los diferentes niveles de rareza de las cartas.
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `rarezas` (
    `id`                INT PRIMARY KEY AUTO_INCREMENT,
    `nombre_rareza`     VARCHAR(100) UNIQUE,
    `descripcion`       TEXT
) COMMENT = 'Niveles de rareza de las cartas';


-- -----------------------------------------------------
-- Tabla: `cartas_ediciones`
-- Relaciona cartas con ediciones y su rareza específica.
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `cartas_ediciones` (
    `id`            INT PRIMARY KEY AUTO_INCREMENT,
    `id_carta`      BIGINT,
```

```
    `id_edicion`    INT,
    `id_rareza`     INT,
    CONSTRAINT `fk_cartas_ediciones_carta`
        FOREIGN KEY (`id_carta`) REFERENCES `cartas`(`id`)
        ON UPDATE CASCADE
        ON DELETE RESTRICT,
    CONSTRAINT `fk_cartas_ediciones_edicion`
        FOREIGN KEY (`id_edicion`) REFERENCES `ediciones`(`id`)
        ON UPDATE CASCADE
        ON DELETE RESTRICT,
    CONSTRAINT `fk_cartas_ediciones_rareza`
        FOREIGN KEY (`id_rareza`) REFERENCES `rarezas`(`id`)
        ON UPDATE CASCADE
        ON DELETE RESTRICT
) COMMENT = 'Relación N:M entre cartas y ediciones, incluyendo rareza';


-- -----------------------------------------------------
-- Tabla: `estados_cartas`
-- Catálogo de los estados de conservación de una carta.
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `estados_cartas` (
    `id`                INT PRIMARY KEY AUTO_INCREMENT,
    `nombre_estado`     VARCHAR(100),
    `descripcion`       TEXT
) COMMENT = 'Estados de conservación de las cartas (Mint, Near Mint, etc.)';


-- -----------------------------------------------------
-- Tabla: `estado_inventarios`
-- Catálogo de los estados de un artículo en el inventario de un usuario.
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `estado_inventarios` (
    `id`                INT PRIMARY KEY AUTO_INCREMENT,
    `nombre_estado`     VARCHAR(100),
    `descripcion`       TEXT
) COMMENT = 'Estados de un artículo en inventario (Disponible, Reservado,
Vendido)';


-- -----------------------------------------------------
```

```
-- Tabla: `inventarios`
-- Artículos específicos que un usuario posee o tiene a la venta.
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `inventarios` (
    `id`                        INT PRIMARY KEY AUTO_INCREMENT,
    `id_usuario`                INT,
    `id_carta`                  BIGINT,
    `id_estado_carta`           INT COMMENT 'FK a estados_cartas: Condición de la
carta',
    `id_estado_en_inventario`   INT COMMENT 'FK a estado_inventarios:
Disponibilidad del ítem',
    `foil`                      BOOLEAN NOT NULL DEFAULT FALSE,
    `comentario`                TEXT,
    `precio`                    DECIMAL(30,2) CHECK (`precio` > 0),
    `imagen`                    VARCHAR(50) COMMENT 'Imagen específica del ítem en
venta',
    CONSTRAINT `fk_inventarios_usuario`
        FOREIGN KEY (`id_usuario`) REFERENCES `usuarios`(`id`)
        ON UPDATE CASCADE
        ON DELETE RESTRICT,
    CONSTRAINT `fk_inventarios_carta`
        FOREIGN KEY (`id_carta`) REFERENCES `cartas`(`id`)
        ON UPDATE CASCADE
        ON DELETE RESTRICT,
    CONSTRAINT `fk_inventarios_estado_carta`
        FOREIGN KEY (`id_estado_carta`) REFERENCES `estados_cartas`(`id`)
        ON UPDATE CASCADE
        ON DELETE RESTRICT,
    CONSTRAINT `fk_inventarios_estado_en_inventario`
        FOREIGN KEY (`id_estado_en_inventario`) REFERENCES
`estado_inventarios`(`id`)
        ON UPDATE CASCADE
        ON DELETE RESTRICT
) COMMENT = 'Inventario de cartas de los usuarios';


-- -----------------------------------------------------
-- Tabla: `tipos_envios`
-- Catálogo de los métodos o tipos de envío disponibles.
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `tipos_envios` (
```

```
    `id`            INT PRIMARY KEY AUTO_INCREMENT,
    `nombre`        VARCHAR(100),
    `descripcion`   TEXT,
    `precio`        DECIMAL(10,2)
) COMMENT = 'Métodos y costes de envío';


-- -----------------------------------------------------
-- Tabla: `estado_envios`
-- Catálogo de los estados de un envío (Pendiente, Enviado, etc.).
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `estado_envios` (
    `id`            INT PRIMARY KEY AUTO_INCREMENT,
    `estado`        VARCHAR(100),
    `descripcion`   TEXT
) COMMENT = 'Estados de los envíos de pedidos';


-- -----------------------------------------------------
-- Tabla: `pedidos`
-- Registra los pedidos realizados por los usuarios.
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `pedidos` (
    `id`                    INT PRIMARY KEY AUTO_INCREMENT,
    `id_usuario`            INT,
    `id_direccion_envio`    INT,
    `id_tipo_envio`         INT,
    `id_estado_envio`       INT,
    `comentario`            TEXT,
    `valoracion`            INT CHECK (`valoracion` > 0 AND `valoracion` <= 5),
    CONSTRAINT `fk_pedidos_usuario`
        FOREIGN KEY (`id_usuario`) REFERENCES `usuarios`(`id`)
        ON UPDATE CASCADE
        ON DELETE RESTRICT,
    CONSTRAINT `fk_pedidos_direccion_envio`
        FOREIGN KEY (`id_direccion_envio`) REFERENCES `direcciones`(`id`)
        ON UPDATE CASCADE
        ON DELETE RESTRICT,
    CONSTRAINT `fk_pedidos_tipo_envio`
        FOREIGN KEY (`id_tipo_envio`) REFERENCES `tipos_envios`(`id`)
```

```
        ON UPDATE CASCADE
        ON DELETE RESTRICT,
    CONSTRAINT `fk_pedidos_estado_envio`
        FOREIGN KEY (`id_estado_envio`) REFERENCES `estado_envios`(`id`)
        ON UPDATE CASCADE
        ON DELETE RESTRICT
) COMMENT = 'Pedidos realizados por los usuarios';


-- -----------------------------------------------------
-- Tabla: `pedidos_inventarios`
-- Detalle de los artículos de inventario que componen cada pedido.
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `pedidos_inventarios` (
    `id`                INT PRIMARY KEY AUTO_INCREMENT,
    `id_pedido`         INT,
    `id_inventario`     INT,
    `fecha`             TIMESTAMP DEFAULT CURRENT_TIMESTAMP COMMENT 'Fecha de
inclusión del ítem en el pedido',
    CONSTRAINT `fk_pedidos_inventarios_pedido`
        FOREIGN KEY (`id_pedido`) REFERENCES `pedidos`(`id`)
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT `fk_pedidos_inventarios_inventario`
        FOREIGN KEY (`id_inventario`) REFERENCES `inventarios`(`id`)
        ON UPDATE CASCADE
        ON DELETE CASCADE
) COMMENT = 'Relación N:M entre pedidos e ítems de inventario';



--
=============================================================================================
=
-- Fin del Script de Creación de la Base de Datos CardTraders
--
=============================================================================================
=
```

7. Carga de Datos Inicial

Describe aquí...

8. Funciones y Procedimientos Almacenados

Describe aquí...

9. Triggers

Describe aquí...

10. Consultas SQL

Describe aquí...

11. Casos de Prueba y Simulación

Describe aquí...

12. Resultados y Verificación

Describe aquí...

13. Capturas de Pantalla (opcional)

Describe aquí...

14. Conclusiones y Mejoras Futuras

Describe aquí...

15. Enlace al Repositorio en GitHub

Describe aquí...