TECNICATURA SUPERIOR EN CIENCIAS DE DATOS E INTELIGENCIA ARTIFICIAL

MÓDULO INNOVACIÓN EN GESTIÓN DE DATOS

PRIMER AÑO

Proyecto Final

1. Descripción del Proyecto

El proyecto "Sistema de Venta de Juegos para PC - PlayHouse" consiste en el desarrollo de una aplicación para facilitar la adquisición de juegos digitales y la gestión integral de usuarios, carritos de compras y transacciones. La aplicación cuenta con funcionalidades para administrar usuarios y juegos, aplicar descuentos por membresías, y gestionar categorías de juegos y carritos de compra.

1.2 Características principales.

- **Gestión de Usuarios**: Registro, modificación, y eliminación de usuarios, incluye control de membresías.
- **Catálogo de Juegos**: Gestión de juegos con clasificación por categorías y precios.
- **Carrito de Compras**: Sistema de carrito que permite añadir o eliminar juegos y realizar la compra de los juegos seleccionados.
- **Transacciones**: Registra las compras realizadas, aplica descuentos según el tipo de membresía del usuario.

2. Justificación

- El proyecto surge como solución a la necesidad de centralizar y optimizar la gestión de ventas de juegos digitales (imaginando que la empresa ya cuenta con una experiencia en venta de juegos físicos). Este sistema permite administrar el inventario de juegos, realizar transacciones seguras y facilitar el acceso de usuarios a su historial de compras y descuentos, proporcionando una experiencia de compra digital completa.

3. Objetivos

Objetivo del Proyecto

Desarrollar una aplicación que permita a los usuarios explorar y adquirir juegos digitales de manera eficiente y que ofrezca a los administradores herramientas para gestionar los juegos, usuarios, categorías y transacciones.

Objetivo Específicos

- Implementar un sistema que gestione el catálogo de juegos, categorías y usuarios de forma integrada.
- Facilitar a los usuarios la compra de juegos y el acceso a un carrito de compras digital.
- Registrar las transacciones con descuentos aplicables según el tipo de cuenta (Estándar o Plus).
- Proporcionar a los administradores herramientas de gestión y administración de usuarios y productos.

4. Metodología

Se ha utilizado una metodología ágil para el desarrollo de este proyecto, lo que ha permitido trabajar de manera incremental. A continuación, se detallan las fases de desarrollo:

Fases del Desarrollo:

1. **Análisis y Diseño**:
    - Identificación de requerimientos y modelado de la base de datos.
    - Diseño de la arquitectura de la aplicación.
2. **Desarrollo de Base de Datos**:
    - Creación de las tablas en MySQL y definición de las relaciones.
3. **Desarrollo de Módulos**:
    - **Gestión de Usuarios**: Registro, actualización, eliminación y visualización de usuarios.
    - **Gestión de Juegos**: Adición, modificación y eliminación de juegos, con verificación de la categoría y precio.
    - **Gestión de Carrito**: Permite la interacción del usuario con su carrito, como agregar o quitar juegos.
    - **Transacciones**: Registro y seguimiento de compras.
4. **Pruebas e Integración**:
    - Pruebas de los módulos de forma independiente y en conjunto.
    - Integración final de todos los componentes.

5. Cronograma

El proyecto se ha dividido en fases con un cronograma basado en el enfoque ágil.

Fase del Proyecto	Actividades	Semana 1	Semana 2	Semana 3	Semana 4
Fase 1: Análisis y Diseño	Relevamiento de requerimientos	X			
	Diseño de la arquitectura de la aplicación		X		

	Modelado de la base de datos		X	X	
Fase 2: Desarrollo de Base de Datos	Creación de esquemas y tablas en MySQL			X	
	Implementación de relaciones y constraints			X	X
	Configuración de seguridad y respaldos				X
Fase 3: Desarrollo de Módulos	Implementación de la gestión de usuarios		X	X	
	Implementación del módulo de juegos			X	X
	Desarrollo de la gestión de carrito				X
	Implementación de la lógica de compras	X	X		
Fase 4: Integración y Pruebas	Integración de todos los módulos			X	
	Pruebas unitarias		X	X	X
Fase 5: Documentación y Presentación	Documentación técnica		X		X
	Preparación de la presentación final				X

6. Presentación del Proyecto

6.1 Estructura del Código

El proyecto está dividido en módulos de Python que se encargan de diferentes funciones dentro del sistema, permitiendo una organización clara y una gestión modular. A continuación se describen algunos de los módulos:

- **carrito.py**: Permite agregar y eliminar juegos en el carrito, así como visualizar el contenido y el total del carrito.

```
iport mysql.connector
from aplicacion.usuario import mostrar_usuario
from aplicacion.juego import mostrar_juego

def agregar_al_carrito(conexion):
    try:
        cursor = conexion.cursor()

        mostrar_usuario(conexion)
        id_usuario = int(input("Ingrese el ID del usuario: "))
        mostrar_juego(conexion)
        id_juego = int(input("Ingrese el ID del juego a agregar: "))

        # Si no ingresa usuario o juego...
        if not id_usuario or not id_juego:
            raise ValueError("ERROR: El ID de usuario y el ID de juego son
obligatorios.")

        # Se verifica si el usuario existe
        cursor.execute("SELECT * FROM Usuario WHERE idUsuario = %s",
(id_usuario,)) # <-- esto es una tupla, por eso la ","
        usuario = cursor.fetchone()
        if not usuario:
            raise ValueError("ERROR: El usuario no existe.")

        # Se verifica si el juego existe
        cursor.execute("SELECT * FROM Juego WHERE idJuego = %s",
(id_juego,))
        juego = cursor.fetchone()
        if not juego:
            raise ValueError("ERROR: El juego no existe.")
```

```
        # Se obtiene el ID del carrito del usuario
        cursor.execute("SELECT idCarrito FROM Carrito WHERE
Usuario_idUsuario = %s", (id_usuario,))
        carrito = cursor.fetchone()
        if not carrito:
            raise ValueError("ERROR: El usuario no tiene un carrito.")

        carrito_id = carrito[0]

        # Se verifica si el juego ya está en el carrito
        cursor.execute("SELECT * FROM Carrito_has_Juego WHERE
Carrito_idCarrito = %s AND Juego_idJuego = %s",
                       (carrito_id, id_juego))
        juego_en_carrito = cursor.fetchone()
        if juego_en_carrito:
            raise ValueError("ERROR: El juego ya está en el carrito.")

        # Se agrega un juego al carrito
        insertar_query = "INSERT INTO Carrito_has_Juego (Carrito_idCarrito,
Juego_idJuego) VALUES (%s, %s)"
        cursor.execute(insertar_query, (carrito_id, id_juego))
        conexion.commit()

        print("Juego agregado al carrito correctamente!")

    except mysql.connector.Error as err:
        print(f"Error al agregar juego al carrito: {err}")
        conexion.rollback()
    except ValueError as errorDeValor:
        print(errorDeValor)
    finally:
        cursor.close()

def eliminar_del_carrito(conexion):
    try:
        cursor = conexion.cursor()
        mostrar_carrito(conexion)  # Muestra el carrito actual del usuario

        id_usuario = int(input("Ingrese el ID del usuario: "))
```

```
        id_juego = int(input("Ingrese el ID del juego a eliminar: "))

        # Verifica que se hayan introducido id's válidos
        if not id_usuario or not id_juego:
            raise ValueError("ERROR: El ID de usuario y el ID de juego son
obligatorios.")

        # Se obtiene el ID del carrito del usuario pasado como arg.
        cursor.execute("SELECT idCarrito FROM Carrito WHERE
Usuario_idUsuario = %s", (id_usuario,))
        carrito = cursor.fetchone()
        if not carrito:
            raise ValueError("ERROR: El usuario no tiene un carrito.")

        carrito_id = carrito[0]

        # Se eliminar el juego del carrito
        eliminar_query = "DELETE FROM Carrito_has_Juego WHERE
Carrito_idCarrito = %s AND Juego_idJuego = %s"
        cursor.execute(eliminar_query, (carrito_id, id_juego))
        conexion.commit()

        print("Juego eliminado del carrito correctamente!")

    except mysql.connector.Error as err:
        print(f"Error al eliminar juego del carrito: {err}")
        conexion.rollback()
    except ValueError as ve:
        print(ve)
    finally:
        cursor.close()

def mostrar_carrito(conexion):
    try:
        cursor = conexion.cursor()
        mostrar_usuario(conexion)
        id_usuario = int(input("Ingrese el ID del usuario: "))

        # Se obtienen los juegos del carrito del usuario, nombre y precio
```

```
        query = """
            SELECT j.idJuego, j.nombre, j.precio
            FROM Carrito_has_Juego chj
            INNER JOIN Juego j ON chj.Juego_idJuego = j.idJuego
            INNER JOIN Carrito c ON chj.Carrito_idCarrito = c.idCarrito
            WHERE c.Usuario_idUsuario = %s
        """
        cursor.execute(query, (id_usuario,))

        resultados = cursor.fetchall()
        # Si no encuentra nada...
        if not resultados:
            print("El carrito está vacío.")
            return

        # Se muestran los resultados
        print("Contenido del carrito:")
        print("-" * 45)
        print("{:<5} {:<30} {:<10}".format("ID", "Nombre", "Precio"))
        print("-" * 45)
        total = 0
        for fila in resultados:
            print("{:<5} {:<30} {:<10.2f}".format(fila[0], fila[1], fila[2]))
            total += fila[2]
        print("-" * 45)
        print(f"Total: {total:.2f}")

    except mysql.connector.Error as err:
        print(f"Error al mostrar el carrito: {err}")
    finally:
        cursor.close()
```

- **categoria.py**: Facilita la administración de categorías de juegos, incluyendo creación, modificación y eliminación de categorías.

```
import mysql.connector

def agregar_categoria(conexion):
    try:
        cursor = conexion.cursor()

        tipo = input("Ingrese el nombre de la categoría: ")

        # Valida que se ingrese un nombre
        if not tipo:
            raise ValueError("ERROR: El nombre de la categoría es obligatorio.")

        # Se verifica si la categoría ya existe
        cursor.execute("SELECT * FROM Categoria WHERE tipo = %s", (tipo,))
        categoria_existente = cursor.fetchone()
        if categoria_existente:
            raise ValueError("ERROR: La categoría ya existe.")

        # Se inserta la categoría en la base de datos
        insertar_query = "INSERT INTO Categoria (tipo) VALUES (%s)"
        cursor.execute(insertar_query, (tipo,))
        conexion.commit()

        print("Categoría agregada correctamente.")

    except mysql.connector.Error as err:
        print(f"Error al agregar categoría: {err}")
        conexion.rollback()
    except ValueError as ve:
        print(ve)
    finally:
        cursor.close()

def modificar_categoria(conexion):
    try:
        cursor = conexion.cursor()
        mostrar_categoria(conexion)  # mostramos las categorías para facilitar la
elección del id
```

```
        id_categoria = int(input("Ingrese el ID de la categoría a modificar: "))

        # Se verifica si la categoría existe
        cursor.execute("SELECT * FROM Categoria WHERE idCategoria = %s",
(id_categoria,))
        categoria_existente = cursor.fetchone()
        if not categoria_existente:
            raise ValueError("ERROR: La categoría no existe.")

        nuevo_tipo = input("Nuevo nombre de la categoría (dejar en blanco para
no modificar): ")

        if not nuevo_tipo:
            print("No se ingresaron cambios.")
            return

        # Se verifica si el nuevo nombre de categoría ya existe, omitiendo la que
se acaba de crear
        cursor.execute("SELECT * FROM Categoria WHERE tipo = %s AND
idCategoria != %s", (nuevo_tipo, id_categoria))
        categoria_existente = cursor.fetchone()
        if categoria_existente:
            raise ValueError("ERROR: Ya existe una categoría con ese nombre.")

        actualizar_query = "UPDATE Categoria SET tipo = %s WHERE
idCategoria = %s"
        cursor.execute(actualizar_query, (nuevo_tipo, id_categoria))
        conexion.commit()

        print("Categoría modificada correctamente.")
    except mysql.connector.Error as err:
        print(f"Error al modificar categoría: {err}")
        conexion.rollback()
    except ValueError as ve:
        print(ve)
    finally:
        cursor.close()

def eliminar_categoria(conexion):
```

```
    try:
        cursor = conexion.cursor()
        mostrar_categoria(conexion)  # mostramos las categorías para facilitar la elección del id

        id_categoria = int(input("Ingrese el ID de la categoría a eliminar: "))

        # Se verifica si la categoría existe
        cursor.execute("SELECT * FROM Categoria WHERE idCategoria = %s", (id_categoria,))
        categoria_existente = cursor.fetchone()
        if not categoria_existente:
            raise ValueError("ERROR: La categoría no existe.")

        # Se verifica si la categoría tiene juegos asociados
        cursor.execute("SELECT * FROM Juego WHERE Categoria_idCategoria = %s", (id_categoria,))
        juegos = cursor.fetchall()
        if juegos:
            print("No se puede eliminar la categoría porque tiene juegos asociados.")
            return

        # Eliminar la categoría
        eliminar_query = "DELETE FROM Categoria WHERE idCategoria = %s"
        cursor.execute(eliminar_query, (id_categoria,))
        conexion.commit()

        print("Categoría eliminada correctamente.")
    except mysql.connector.Error as err:
        print(f"Error al eliminar categoría: {err}")
        conexion.rollback()
    except ValueError as ve:
        print(ve)
    finally:
        cursor.close()

def mostrar_categoria(conexion):
    try:
```

```
        cursor = conexion.cursor()
        cursor.execute("SELECT * FROM Categoria")
        categorias = cursor.fetchall()

        if not categorias:
            print("No hay categorías registradas.")
            return

        print("Categorías registradas:")
        print("-" * 30)  # Línea de separación, 30 guiones
        print("{:<5} {:<25}".format("ID", "Tipo")) # Trunca y alínea a la izquierda
        print("-" * 30)  # Línea de separación
        for categoria in categorias:
            print("{:<5} {:<25}".format(categoria[0], categoria[1]))

    except mysql.connector.Error as err:
        print(f"Error al mostrar categorías: {err}")
    finally:
        cursor.close()
```

- **compra.py**: Gestiona la lógica para realizar una compra, aplicando descuentos según la membresía del usuario.

```
import mysql.connector
from aplicacion.usuario import mostrar_usuario

def realizar_compra(conexion):
    try:
        cursor = conexion.cursor()

        mostrar_usuario(conexion)
        id_usuario = int(input("Ingrese el ID del usuario que realiza la compra: "))

        # Se valida si el usuario existe
        cursor.execute("SELECT * FROM Usuario WHERE idUsuario = %s",
```

```
(id_usuario,))
        usuario = cursor.fetchone()
        if not usuario:
            raise ValueError("Error: El usuario no existe.")

        # Se obtiene el carrito del usuario
        cursor.execute("SELECT idCarrito FROM Carrito WHERE
Usuario_idUsuario = %s", (id_usuario,))
        carrito = cursor.fetchone()
        if not carrito:
            raise ValueError("ERROR: El usuario no tiene un carrito! ")
        carrito_id = carrito[0]

        # Se obtienen los juegos agregados al carrito
        cursor.execute("""
            SELECT Juego_idJuego, precio
            FROM Carrito_has_Juego
            INNER JOIN Juego ON Carrito_has_Juego.Juego_idJuego =
Juego.idJuego
            WHERE Carrito_idCarrito = %s
        """, (carrito_id,))
        juegos_carrito = cursor.fetchall()

        if not juegos_carrito:
            raise ValueError("ERROR: El carrito está vacío!")

        total = 0
        for juego_id, precio in juegos_carrito:

            # Se calcula el precio final y se aplica descuento si corresponde
            precio_final = precio
            if usuario[5]:  # Si el usuario tiene asignada una membresía esta
columna no sería Null
                cursor.execute("SELECT descuento FROM Membresia WHERE
idMembresia = %s", (usuario[5],))
                descuento = cursor.fetchone()[0]
                precio_final *= (1 - descuento / 100)
            total += precio_final
```

```
            # y registramos la compra en la base de datos
            insertar_query = """
                INSERT INTO Compra (Usuario_idUsuario, Juego_idJuego, fecha,
total)
                VALUES (%s, %s, NOW(), %s)
            """
            valores = (id_usuario, juego_id, precio_final)
            cursor.execute(insertar_query, valores)

        conexion.commit()

        # vaciamos el carrito una vez realizada la compra
        cursor.execute("DELETE FROM Carrito_has_Juego WHERE
Carrito_idCarrito = %s", (carrito_id,))
        conexion.commit()

        print(f"La compra se realizó correctamente. Total: {total:.2f}") #acá
formateamos para que muestre float de dos dígitos

    except mysql.connector.Error as err:
        print(f"Error al realizar la compra: {err}")
        conexion.rollback()
    except ValueError as errorDeValor:
        print(errorDeValor)
    finally:
        cursor.close()

def mostrar_compras(conexion):
    try:
        cursor = conexion.cursor()

        # Obtenemos todas las compras con información del usuario y del juego
        query = """
            SELECT c.idCompra, u.email, j.nombre, c.fecha, c.total
            FROM Compra c
            INNER JOIN Usuario u ON c.Usuario_idUsuario = u.idUsuario
            INNER JOIN Juego j ON c.Juego_idJuego = j.idJuego
        """
        cursor.execute(query)
```

```
        resultados = cursor.fetchall()
        if not resultados:
            print("No hay compras registradas.")
            return

        # Mostramos los resultados
        print("Compras registradas:")
        print("-" * 80)
        print("{:<10} {:<30} {:<30} {:<20} {:<10}".format("ID Compra", "Email
Usuario", "Nombre Juego", "Fecha", "Total"))
        print("-" * 80)
        for fila in resultados:
            # Le datos formato a la fecha
            fecha_formateada = fila[3].strftime("%Y-%m-%d %H:%M:%S")
            print("{:<10} {:<30} {:<30} {:<20} {:<10.2f}".format(fila[0], fila[1], fila[2],
fecha_formateada, fila[4]))

    except mysql.connector.Error as err:
        print(f"Error al mostrar compras: {err}")
    finally:
        cursor.close()
```

- **juego.py**: Módulo dedicado a la gestión del catálogo de juegos.

```
import mysql.connector
from aplicacion.categoria import mostrar_categoria

def agregar_juego(conexion):
    try:
        cursor = conexion.cursor()

        nombre = input("Ingrese el nombre del juego: ")
        precio = float(input("Ingrese el precio del juego: "))
        mostrar_categoria(conexion)
        categoria_id = int(input("Ingrese el ID de la categoría: "))
```

```
        # Se valida que se hayan ingresado todos los datos pedidos
        if not nombre or not precio or not categoria_id:
            raise ValueError("ERROR: Todos los campos son obligatorios.")

        # Se verifica si el nombre del juego ya existe
        cursor.execute("SELECT * FROM Juego WHERE nombre = %s",
(nombre,))
        juego_existente = cursor.fetchone()
        if juego_existente:
            raise ValueError("ERROR: Ya existe un juego con ese nombre.")

        # Se verifica si la categoría existe
        cursor.execute("SELECT * FROM Categoria WHERE idCategoria = %s",
(categoria_id,))
        categoria_existente = cursor.fetchone()
        if not categoria_existente:
            raise ValueError("ERROR: La categoría no existe.")

        # Se inserta el juego en la base de datos
        insertar_query = """
            INSERT INTO Juego (nombre, precio, Categoria_idCategoria)
            VALUES (%s, %s, %s)
        """
        valores = (nombre, precio, categoria_id)
        cursor.execute(insertar_query, valores)
        conexion.commit()

        print("Juego agregado correctamente.")

    except mysql.connector.Error as err:
        print(f"Error al agregar juego: {err}")
        conexion.rollback()
    except ValueError as ve:
        print(ve)
    finally:
        cursor.close()

def modificar_juego(conexion):
    try:
```

```
        cursor = conexion.cursor()
        mostrar_juego(conexion)  # Se muestran todos los juegos

        id_juego = int(input("Ingrese el ID del juego a modificar: "))

        # Se verifica si el juego existe
        cursor.execute("SELECT * FROM Juego WHERE idJuego = %s",
(id_juego,))
        juego_existente = cursor.fetchone()
        if not juego_existente:
            raise ValueError("ERROR: El juego no existe.")

        nombre = input("Nuevo nombre (dejar en blanco para no modificar): ")
        precio = input("Nuevo precio (dejar en blanco para no modificar): ")
        mostrar_categoria(conexion)
        categoria_id = input("Nuevo ID de categoría (dejar en blanco para no
modificar): ")

        actualizar_campos = []
        valores = []

        if nombre:
            actualizar_campos.append("nombre = %s")
            valores.append(nombre)
        if precio:
            actualizar_campos.append("precio = %s")
            valores.append(float(precio))
        if categoria_id:
            # Se verifica si la categoría existe
            cursor.execute("SELECT * FROM Categoria WHERE idCategoria =
%s", (int(categoria_id),))
            categoria_existente = cursor.fetchone()
            if not categoria_existente:
                raise ValueError("ERROR: La categoría no existe.")
            actualizar_campos.append("Categoria_idCategoria = %s")
            valores.append(int(categoria_id))

        if not actualizar_campos:
            print("No se ingresaron cambios.")
```

```
            return

        actualizar_query = f"UPDATE Juego SET {', '.join(actualizar_campos)}
WHERE idJuego = %s"
        valores.append(id_juego)
        cursor.execute(actualizar_query, valores)
        conexion.commit()

        print("Juego modificado correctamente.")
    except mysql.connector.Error as err:
        print(f"Error al modificar juego: {err}")
        conexion.rollback()
    except ValueError as errorDeValor:
        print(errorDeValor)
    finally:
        cursor.close()

def eliminar_juego(conexion):
    try:
        cursor = conexion.cursor()
        mostrar_juego(conexion)  # se muestran los juegos para que el usuario
elija cuál eliminar

        id_juego = int(input("Ingrese el ID del juego a eliminar: "))

        # Se verifica si el juego existe
        cursor.execute("SELECT * FROM Juego WHERE idJuego = %s",
(id_juego,))
        juego_existente = cursor.fetchone()
        if not juego_existente:
            raise ValueError("Error: El juego no existe.")

        # Se verifica si el juego tiene compras asociadas
        cursor.execute("SELECT * FROM Compra WHERE Juego_idJuego =
%s", (id_juego,))
        compras = cursor.fetchall()
        if compras:
            print("No se puede eliminar el juego porque tiene compras asociadas.")
            return
```

```
        # Se elimina el juego
        eliminar_query = "DELETE FROM Juego WHERE idJuego = %s"
        cursor.execute(eliminar_query, (id_juego,))
        conexion.commit()

        print("Juego eliminado correctamente.")
    except mysql.connector.Error as err:
        print(f"Error al eliminar juego: {err}")
        conexion.rollback()
    except ValueError as errorDeValor:
        print(errorDeValor)
    finally:
        cursor.close()

def mostrar_juego(conexion):
    try:
        cursor = conexion.cursor()

        # Se obtienen todos los datos de los juegos, también la categoría
        query = """
            SELECT j.idJuego, j.nombre, j.precio, c.tipo AS Categoria
            FROM Juego j
            INNER JOIN Categoria c ON j.Categoria_idCategoria = c.idCategoria
        """
        cursor.execute(query)

        resultados = cursor.fetchall()
        if not resultados:
            print("No hay juegos registrados.")
            return

        # Se muestran los resultados
        print("Juegos registrados:")
        print("-" * 60)  # Línea de separación
        print("{:<5} {:<30} {:<10} {:<15}".format("ID", "Nombre", "Precio",
"Categoría"))
        print("-" * 60)  # Línea de separación
        for fila in resultados:
```

```
            print("{:<5} {:<30} {:<10} {:<15}".format(fila[0], fila[1], fila[2], fila[3]))

    except mysql.connector.Error as err:
        print(f"Error al mostrar juegos: {err}")
    finally:
        cursor.close()
```

- **usuario.py**: Permite registrar, modificar, eliminar y visualizar usuarios con su tipo de membresía.

```
import mysql.connector

def registrar_usuario(conexion):
    try:
        cursor = conexion.cursor()
        #cursor.execute("START TRANSACTION")  # Iniciar transacción

        email = input("Ingrese el correo electrónico del usuario: ")
        password = input("Ingrese la contraseña del usuario: ")
        nombre = input("Ingrese el nombre del usuario: ")
        apellido = input("Ingrese el apellido del usuario: ")
        id_membresia = int(input("Ingrese el ID de la membresía: "))

        # Se validan datos obligatorios
        if not email or not password or not nombre or not apellido:
            raise ValueError("ERROR: Todos los campos son obligatorios.")

        # Se verifica si el correo electrónico ya está registrado
        cursor.execute("SELECT * FROM Usuario WHERE email = %s", (email,))
        usuario_existente = cursor.fetchone()
        if usuario_existente:
            raise ValueError("ERROR: El correo electrónico ya está registrado.")

        # Se verifica si el ID de membresía existe
```

```
        cursor.execute("SELECT * FROM Membresia WHERE idMembresia =
%s", (id_membresia,))
        membresia_existente = cursor.fetchone()
        if not membresia_existente:
            raise ValueError("ERROR: El ID de membresía no es válido.")

        # Se inserta el usuario en la base de datos
        insertar_query_usuario = """
            INSERT INTO Usuario (email, password, nombre, apellido,
Membresia_idMembresia)
            VALUES (%s, %s, %s, %s, %s)
        """
        valores_usuario = (email, password, nombre, apellido, id_membresia)
        cursor.execute(insertar_query_usuario, valores_usuario)

        # Se obtiene el ID del usuario recién insertado
        usuario_id = cursor.lastrowid

        # y se crea un carrito para el nuevo usuario
        insertar_carrito_query = "INSERT INTO Carrito (Usuario_idUsuario)
VALUES (%s)"
        valores_carrito = (usuario_id,)  # acá pasamos el usuario_id como
parámetro
        cursor.execute(insertar_carrito_query, valores_carrito)

        conexion.commit()

        print("Usuario y carrito creados correctamente.")

    except mysql.connector.Error as err:
        print(f"Error al registrar usuario: {err}")
        conexion.rollback()
    except ValueError as ve:
        print(ve)
    finally:
        cursor.close()

def modificar_usuario(conexion):
    try:
```

```
        cursor = conexion.cursor()

        # Se muestran usuarios para que el usuario elija cuál modificar
        mostrar_usuario(conexion)

        id_usuario = int(input("Ingrese el ID del usuario a modificar: "))

        # Verificar si el usuario existe
        cursor.execute("SELECT * FROM Usuario WHERE idUsuario = %s",
(id_usuario,))
        usuario_existente = cursor.fetchone()
        if not usuario_existente:
            raise ValueError("ERROR: El usuario no existe.")

        email = input("Nuevo correo electrónico (dejar en blanco para no
modificar): ")
        password = input("Nueva contraseña (dejar en blanco para no modificar):
")
        nombre = input("Nuevo nombre (dejar en blanco para no modificar): ")
        apellido = input("Nuevo apellido (dejar en blanco para no modificar): ")
        id_membresia = input("Nuevo ID de membresía (dejar en blanco para no
modificar): ")

        actualizar_campos = []
        valores = []

        if email:
            # Se verifica si el nuevo correo electrónico ya está registrado (se omite
el del usuario actual)
            cursor.execute("SELECT * FROM Usuario WHERE email = %s AND
idUsuario != %s", (email, id_usuario))
            email_existente = cursor.fetchone()
            if email_existente:
                raise ValueError("ERROR: El correo electrónico ya está registrado
por otro usuario.")

            actualizar_campos.append("email = %s")
            valores.append(email)
        if password:
```

```
            actualizar_campos.append("password = %s")
            valores.append(password)
        if nombre:
            actualizar_campos.append("nombre = %s")
            valores.append(nombre)
        if apellido:
            actualizar_campos.append("apellido = %s")
            valores.append(apellido)
        if id_membresia:
            # Se verifica si el nuevo ID de membresía es válido
            cursor.execute("SELECT * FROM Membresia WHERE idMembresia =
%s", (int(id_membresia),))
            membresia_existente = cursor.fetchone()
            if not membresia_existente:
                raise ValueError("ERROR: El ID de membresía no es válido.")

            actualizar_campos.append("Membresia_idMembresia = %s")
            valores.append(int(id_membresia))

        if not actualizar_campos:
            print("No se ingresaron cambios.")
            return

        actualizar_query = f"UPDATE Usuario SET {', '.join(actualizar_campos)}
WHERE idUsuario = %s"
        valores.append(id_usuario)
        cursor.execute(actualizar_query, valores)
        conexion.commit()

        print("Usuario modificado correctamente.")
    except mysql.connector.Error as err:
        print(f"Error al modificar usuario: {err}")
        conexion.rollback()
    except ValueError as ve:
        print(ve)
    finally:
        cursor.close()
```

```
def eliminar_usuario(conexion):
    try:
        cursor = conexion.cursor()
        mostrar_usuario(conexion)  # Se muestran usuarios para que el usuario elija cuál eliminar

        id_usuario = int(input("Ingrese el ID del usuario a eliminar: "))

        # Se verifica si el usuario existe
        cursor.execute("SELECT * FROM Usuario WHERE idUsuario = %s", (id_usuario,))
        usuario_existente = cursor.fetchone()
        if not usuario_existente:
            raise ValueError("ERROR: El usuario no existe.")

        # Se verifica si el usuario tiene compras asociadas
        cursor.execute("SELECT * FROM Compra WHERE Usuario_idUsuario = %s", (id_usuario,))
        compras = cursor.fetchall()
        if compras:
            print("No se puede eliminar el usuario porque tiene compras asociadas.")
            return

        # Se emilina el carrito del usuario (si existe)
        cursor.execute("DELETE FROM Carrito WHERE Usuario_idUsuario = %s", (id_usuario,))

        # Se elinina el usuario
        eliminar_query = "DELETE FROM Usuario WHERE idUsuario = %s"
        cursor.execute(eliminar_query, (id_usuario,))
        conexion.commit()

        print("Usuario eliminado correctamente.")
    except mysql.connector.Error as err:
        print(f"Error al eliminar usuario: {err}")
        conexion.rollback()
    except ValueError as ve:
        print(ve)
```

```
    finally:
        cursor.close()


def mostrar_usuario(conexion):
    try:
        cursor = conexion.cursor()

        # Consulta para obtener los datos de los usuarios y el tipo de membresía
        query = """
            SELECT u.idUsuario, u.email, u.Nombre, u.Apellido, m.tipo AS
TipoMembresia
            FROM Usuario u
            INNER JOIN Membresia m ON u.Membresia_idMembresia =
m.idMembresia
        """
        cursor.execute(query)

        resultados = cursor.fetchall()
        if not resultados:
            print("No hay usuarios registrados.")
            return

        # Se muestran los resultados
        print("Usuarios registrados:")
        print("-" * 65)  # Línea de separación
        print("{:<5} {:<30} {:<15} {:<15} {:<15}".format("ID", "Email", "Nombre",
"Apellido", "Tipo de Membresia"))
        print("-" * 65)  # Línea de separación
        for fila in resultados:
            print("{:<5} {:<30} {:<15} {:<15} {:<15}".format(fila[0], fila[1], fila[2], fila[3],
fila[4]))

    except mysql.connector.Error as err:
        print(f"Error al mostrar usuarios: {err}")
    finally:
        cursor.close()
```

6.2 Funcionalidades Implementadas

- **Gestión de Juegos y Categorías**: Mantenimiento de juegos y categorías, incluyendo validación de datos.
- **Carrito de Compras**: Sistema para que el usuario pueda seleccionar juegos y realizar la compra final.
- **Transacciones con Descuentos**: Aplicación de descuentos automáticos según el tipo de cuenta.
- **Visualización de Historial de Compras**: Permite a los usuarios y administradores consultar el historial de transacciones.

6.3 Estructura de la Base de Datos ventadejuegos

La base de datos ventadejuegos está diseñada para gestionar toda la información relevante de una plataforma de venta de juegos. Esta estructura permite un manejo efectivo de los usuarios, juegos, categorías, carritos de compra, y transacciones.

Tablas Principales

1. **usuario**
    - **idUsuario**: int (PK) — Identificador único para cada usuario.
    - **email**: varchar(45) — Correo electrónico del usuario (debe ser único).
    - **password**: varchar(255) — Contraseña del usuario.
    - **nombre**: varchar(45) — Nombre del usuario.
    - **apellido**: varchar(45) — Apellido del usuario.
    - **Membresia_idMembresia**: int (FK) — Referencia a la membresía del usuario (estándar o plus).
    - **fecha_creacion**: datetime — Fecha y hora de creación del registro.
    - **fecha_modificacion**: datetime — Fecha y hora de la última modificación del registro.
2. **membresia**
    - **idMembresia**: int (PK) — Identificador único de la membresía.
    - **tipo**: varchar(45) — Tipo de membresía (ej., estándar o plus).

    - **descuento**: decimal(5,2) — Porcentaje de descuento que aplica la membresía.
3. **categoria**
    - **idCategoria**: int (PK) — Identificador único de la categoría.
    - **tipo**: varchar(45) — Nombre o tipo de la categoría (ej., Acción, Terror, etc.).
4. **juego**
    - **idJuego**: int (PK) — Identificador único del juego.
    - **nombre**: varchar(45) — Nombre del juego.
    - **precio**: decimal(10,2) — Precio del juego.
    - **Categoria_idCategoria**: int (FK) — Referencia a la categoría a la que pertenece el juego.
5. **carrito**
    - **idCarrito**: int (PK) — Identificador único para el carrito de cada usuario.
    - **Usuario_idUsuario**: int (FK) — Referencia al usuario dueño del carrito.
    - **fecha_creacion**: datetime — Fecha de creación del carrito.
    - **fecha_modificacion**: datetime — Fecha de la última modificación del carrito.
6. **carrito_has_juego**
    - **Carrito_idCarrito**: int (PK, FK) — Identificador del carrito.
    - **Juego_idJuego**: int (PK, FK) — Identificador del juego agregado al carrito.
    - **fecha_adicion**: datetime — Fecha de adición del juego al carrito.
7. **compra**
    - **idCompra**: int (PK) — Identificador único de la compra.
    - **Usuario_idUsuario**: int (FK) — Identificador del usuario que realizó la compra.
    - **Juego_idJuego**: int (FK) — Identificador del juego comprado.
    - **fecha**: datetime — Fecha de la compra.
    - **total**: decimal(10,2) — Total de la compra, con descuento aplicado si corresponde.
8. **compra_detalle**
    - **idDetalle**: int (PK) — Identificador único del detalle de la compra.
    - **Compra_idCompra**: int (FK) — Identificador de la compra a la que pertenece el detalle.
    - **Juego_idJuego**: int (FK) — Identificador del juego adquirido.
    - **precio**: decimal(10,2) — Precio del juego al momento de la compra.

Relaciones entre Tablas

- **usuario - membresia**: Relación uno a muchos donde un usuario tiene una membresía que define posibles descuentos.
- **juego - categoria**: Relación uno a muchos, donde cada juego pertenece a una sola categoría.
- **carrito - usuario**: Relación uno a uno donde cada usuario tiene un carrito asociado.
- **carrito_has_juego**: Relación muchos a muchosque asocia juegos a carritos.
- **compra - usuario**: Relación uno a muchos, donde un usuario puede tener múltiples compras.
- **compra_detalle - compra**: Relación uno a muchos, asociando detalles de juegos específicos a cada compra.

7. Conclusiones

Este proyecto ha permitido la creación de un sistema de venta digital eficiente, se cubren aspectos de gestión de usuarios, inventario de juegos y transacciones (carrito de compras y gestión de compras).

La modularidad del código facilita su mantenimiento y posible expansión.

La implementación de descuentos para cuentas Plus proporciona valor adicional a los usuarios y permite explorar futuras mejoras en la experiencia del cliente.