Universidad Autónoma de Nuevo León

Facultad de Ciencias Fisico-Matematicas

Seguridad en Base de Datos

Producto Integrador de Aprendizaje

Grupo: 062

Docente: Melissa Jacqueline Mendieta González

Alumnos:

Jesús Eduardo Rodríguez García - 2058524

Aldo De Jesús Luna Maldonado - 2087570

Christian Domínguez Villanueva - 2118864

Israel Sánchez Hilario - 2094993

Darien Alexander Ñañez Torres – 2119054

Abril Azeneth Alcocer Piñero - 2048108

Contenido

DESCRIPCION DE LA EMPRESA

Banco Capibaras_MX

Tipo de empresa: Institución financiera dedicada a ofrecer servicios bancarios a personas y pequeñas empresas.

Descripción general**:** Banco Capibaras_MX es una entidad financiera mexicana que ofrece una amplia gama de productos y servicios orientados a la administración y crecimiento del patrimonio de sus clientes. Su principal objetivo es brindar soluciones seguras, confiables y accesibles que faciliten las operaciones financieras del día a día, tanto de particulares como de pequeños negocios.

Servicios y operaciones principales**:** Apertura y manejo de cuentas: permite a los clientes disponer de cuentas de ahorro o cheques para realizar depósitos, retiros y transferencias.

Créditos y préstamos**:** ofrece créditos personales y empresariales para cubrir necesidades económicas o impulsar proyectos.

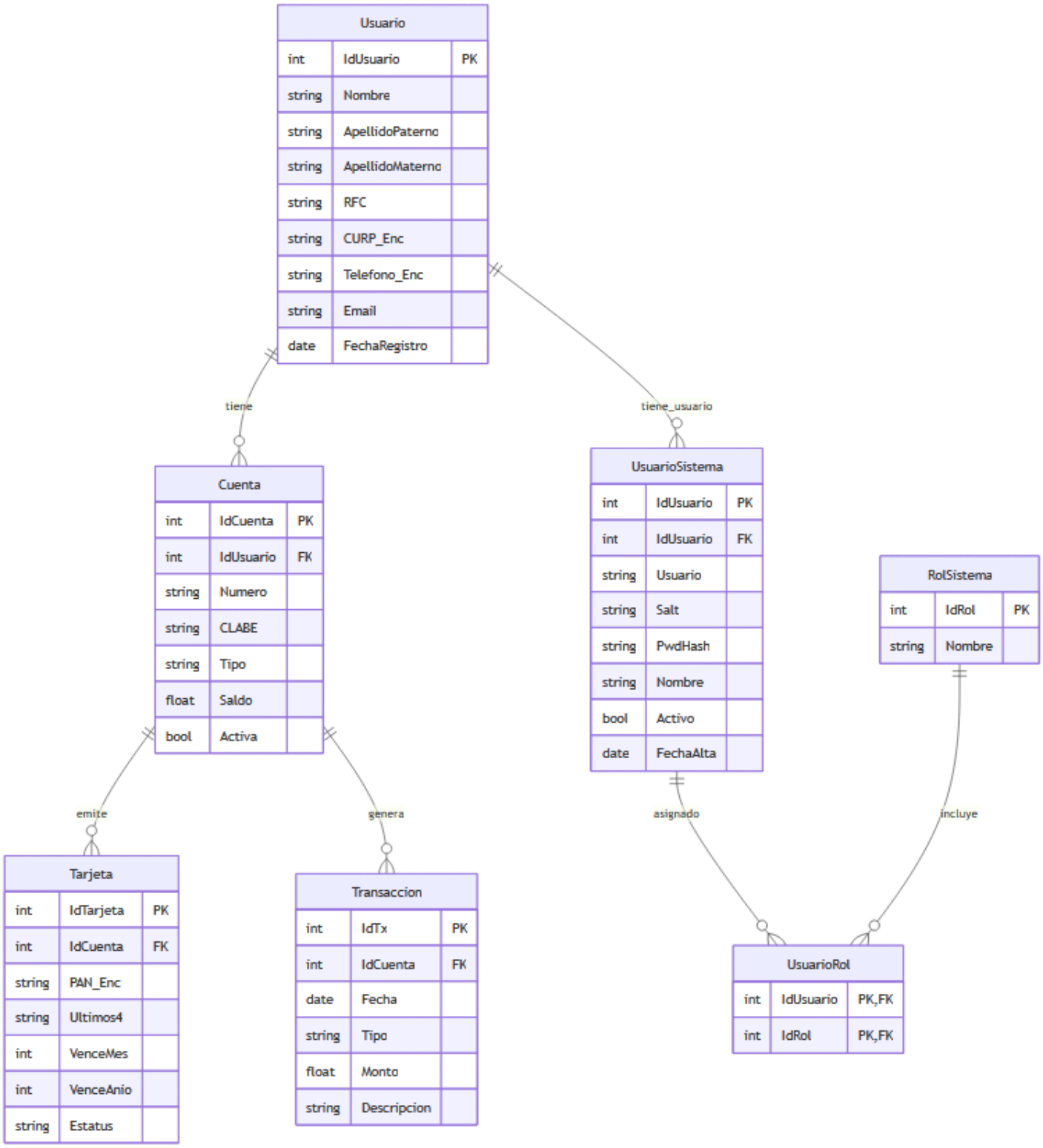
Tarjetas bancarias**:** proporciona tarjetas de débito y crédito para realizar compras en línea y en establecimientos físicos.

Pagos y transferencias electrónicas**:** posibilita el pago de servicios, envío de dinero entre cuentas y operaciones interbancarias.

Atención al cliente**:** cuenta con asesoría personalizada en sucursales y canales digitales para resolver dudas, consultas y gestiones financieras.

Banca digital**:** brinda acceso a una plataforma en línea y aplicación móvil que permite consultar saldos, realizar operaciones y controlar los movimientos financieros de forma práctica y segura.

Diagrama ER


Usuario
int IdUsuario PK
string Nombre
string ApellidoPaterno
string ApellidoMaterno
string RFC
string CURP_Enc
string Telefono_Enc
string Email
date FechaRegistro
tiene
tiene_usuario
Cuenta
int IdCuenta PK
int IdUsuario FK
string Numero
string CLABE
string Tipo
float Saldo
bool Activa
UsuarioSistema
int IdUsuario PK
int IdUsuario FK
string Usuario
string Salt
string PwdHash
string Nombre
bool Activo
date FechaAlta
RolSistema
int IdRol PK
string Nombre
emite
genera
asignado
incluye
Tarjeta
int IdTarjeta PK
int IdCuenta FK
string PAN_Enc
string Ultimos4
int VenceMes
int VenceAnio
string Estatus
Transaccion
int IdTx PK
int IdCuenta FK
date Fecha
string Tipo
float Monto
string Descripcion
UsuarioRol
int IdUsuario PK,FK
int IdRol PK,FK

Normalizacion

1FN (Primera Forma Normal)

- Todas las tablas tienen atributos atómicos, sin grupos repetidos ni multivalorados.
- Los cifrados siguen siendo atómicos a nivel lógico.

2FN (Segunda Forma Normal)

- Todas las tablas tienen **PK simple**, excepto UsuarioRol pero no tiene otros atributos no-clave.

3FN (Tercera Forma Normal)

- **Cliente**: no hay atributos no-clave que determinen a otros no-clave.
- **Cuenta**: atributos dependen de la clave; no dependen de otro no-clave.
- **Tarjeta**: Ultimos4 depende funcionalmente de PAN_Enc , lo que viola 3FN/BCNF en sentido estricto; sin embargo, es una desnormalización controlada aceptable por motivos operativos de banca. Si quisieras 3FN pura, elimina Ultimos4 y calcúlalo al vuelo.
- **Transaccion**: todo depende de IdTx.
- **UsuarioSistema / RolSistema / UsuarioRol**: en 3FN; Usuario y Nombre ya son únicos.

Investigación de dos ataques de base de datos

Ataque 1: Inyección SQL (SQL Injection)

La inyección SQL es uno de los ataques más comunes y peligrosos en bases de datos. Este consiste en insertar código SQL malicioso dentro de entradas que deberían ser simples textos, como campos de usuario, contraseñas, filtros o parámetros enviados por aplicaciones web y móviles.

Si la aplicación concatena directamente lo que recibe del usuario dentro de una consulta, el motor de SQL interpreta esa entrada como parte del comando, lo que permite alterar o manipular la consulta original.

Un ejemplo de cómo podría ocurrir en un sistema bancario es el siguiente:

SELECT * FROM Cliente

WHERE Usuario = 'usuarioIngresado'

AND Password = 'contraseñaIngresada';

Si no hay validación ni parametrización, un atacante podría introducir:

Usuario: admin' --

Contraseña: (cualquier valor)

La consulta pasaría a:

SELECT * FROM Cliente

WHERE Usuario = 'admin' --'

AND Password = 'cualquier'

El símbolo -- comenta el resto de la instrucción, permitiendo el acceso sin conocer la contraseña.

Esto compromete datos sensibles, cuentas, transacciones y todo el sistema de autenticación.

Ataque 2: Escalada de privilegios

La escalada de privilegios ocurre cuando un atacante o usuario obtiene permisos superiores a los que le corresponden, ya sea por mala configuración, asignación incorrecta de roles o vulnerabilidades internas.
Un usuario con demasiados privilegios puede ejecutar comandos peligrosos como:

- Eliminar tablas.
- Consultar información confidencial de clientes.
- Alterar transacciones para ocultar movimientos fraudulentos.
- Modificar roles, permisos o auditorías.

En una institución bancaria, este tipo de ataque puede tener consecuencias extremadamente graves, como manipulación de saldos, eliminación de evidencia, creación de usuarios ilegítimos y alteración de procesos críticos.

Este ataque suele originarse cuando:

- La aplicación utiliza un usuario con permisos excesivos para conectarse a la base.
- No existe una separación clara entre funciones (operación, auditoría y administración).

El control de privilegios es esencial para minimizar el impacto de cualquier ataque interno o externo.

Propuesta de contramedidas

Consultas parametrizadas

Las consultas parametrizadas separan explícitamente los valores proporcionados por el usuario del código SQL.
Esto significa que el motor de SQL Server interpreta las entradas únicamente como datos, sin importar el contenido, evitando que puedan transformarse en instrucciones ejecutables.

Ejemplo aplicado (inicio de sesión seguro):

```
CREATE PROCEDURE dbo.usp_LoginCliente
   @Usuario   NVARCHAR(50),
   @Password  NVARCHAR(100)
AS
BEGIN
   SET NOCOUNT ON;

   SELECT IdCliente, Nombre, Estatus
   FROM Cliente
   WHERE Usuario = @Usuario
     AND PasswordHash = HASHBYTES('SHA2_256', @Password);
END;
GO
```

Las formas en que este procedimiento protege son las siguientes:

- El servidor nunca concatena texto para construir la consulta.
- Reduce la exposición del sistema ante ataques automáticos o manuales de inyección.

Uso de roles mínimos (Principio de privilegios mínimos)

El principio de privilegios mínimos consiste en otorgar a cada usuario, servicio o aplicación solo los permisos estrictamente necesarios para realizar su función, sin conceder acceso adicional.

Implementarlo correctamente reduce el impacto de cualquier ataque, ya que incluso si una cuenta es comprometida, el atacante solo podrá operar dentro de un ámbito limitado.

Creación de un rol seguro para la aplicación de clientes:

```
CREATE ROLE rolClienteApp;
GO


GRANT EXECUTE ON dbo.usp_LoginCliente TO rolClienteApp;
GRANT EXECUTE ON dbo.usp_RegistrarTransaccion TO rolClienteApp;
GO
```

Con esto, la aplicación:

- No puede leer tablas completas.
- No puede modificar datos directamente.
- Solo puede ejecutar procedimientos específicos que ya están validados.

Vistas seguras

Las vistas seguras permiten mostrar únicamente la información estrictamente necesaria a los usuarios o aplicaciones, evitando exponer datos sensibles o estructuras internas de la base de datos. Funcionan como una capa de abstracción que regula qué columnas pueden ser consultadas y bajo qué condiciones, sin otorgar acceso directo a las tablas reales.

En un entorno bancario, las vistas son esenciales para proteger información personal, financiera o confidencial que no debe ser visible para usuarios operativos, aplicaciones externas o servicios con permisos limitados.

Ejemplo aplicado (Vista segura para exponer solo datos no sensibles del cliente):

```
CREATE VIEW vw_ClientePublico
AS
SELECT
    IdCliente,
    Nombre,
    ApellidoPaterno,
    ApellidoMaterno,
    UltimaConexion,
    Estatus
FROM Cliente
WHERE Estatus = 'Activo';
GO
```

Esta vista restringe el acceso a información sensible como el número completo de cuenta, hashes de contraseña, direcciones, correos protegidos o identificadores confidenciales.

Las vistas seguras aportan los siguientes beneficios:

- Limitar la exposición de datos sensibles.
- Reducir el riesgo en caso de accesos indebidos.
- Cumplir normativas de protección de datos.
- Otorgar acceso controlado sin comprometer la estructura interna.

Encriptación

La encriptación proporciona una capa adicional de protección al cifrar información crítica almacenada en la base de datos. Incluso si un atacante obtiene acceso a los archivos físicos, respaldos o realiza una extracción no autorizada, los datos cifrados resultan ilegibles sin las llaves correspondientes.

En SQL Server se emplean distintos mecanismos de cifrado relevantes en un sistema bancario:

1. Cifrado de columnas (Column Encryption)

Útil para proteger valores como números de cuenta, identificaciones, tokens o datos personales.

Ejemplo aplicado:

```
-- Creación de llaves de cifrado

CREATE MASTER KEY ENCRYPTION BY PASSWORD =
'ClaveFuerte_2025*';


CREATE CERTIFICATE Certificado_Cliente

WITH SUBJECT = 'Cifrado de datos sensibles de clientes';
```

```
CREATE SYMMETRIC KEY Llave_Cliente
WITH ALGORITHM = AES_256
ENCRYPTION BY CERTIFICATE Certificado_Cliente;
GO
-- Cifrado de un campo sensible
OPEN SYMMETRIC KEY Llave_Cliente
DECRYPTION BY CERTIFICATE Certificado_Cliente;

UPDATE Cliente
SET NumeroCuentaCifrado =
EncryptByKey(Key_GUID('Llave_Cliente'), NumeroCuenta);

CLOSE SYMMETRIC KEY Llave_Cliente;
```

2. Hashing de contraseñas

Evita almacenar contraseñas en texto plano, utilizando algoritmos seguros como SHA-256.

3. Transparent Data Encryption (TDE)

Cifra toda la base de datos en reposo, protegiendo archivos .mdf, .ldf y respaldos.

Beneficios generales de la encriptación:

- Previene fuga de datos sensibles.
- Protege respaldos robados o comprometidos.
- Reduce impactos de ataques internos.
- Asegura que datos críticos permanezcan ilegibles sin autorización.

Breve análisis del impacto si no se hubieran aplicado las contramedidas

La ausencia de las contramedidas propuestas (consultas parametrizadas, uso de roles mínimos, vistas seguras y encriptación) expondría al sistema bancario a riesgos graves con consecuencias operativas, legales y financieras.

1. Explotación mediante SQL Injection

Sin parametrización, un atacante podría:

- Acceder a cuentas ajenas
- Extraer o modificar información financiera.
- Realizar transacciones no autorizadas.
- Eliminar o alterar registros críticos.

Esto comprometería el funcionamiento del banco y pondría en riesgo los fondos de los clientes.

2. Escalada de privilegios sin control

Sin roles mínimos:

- Usuarios internos podrían obtener permisos administrativos.
- Un atacante podría modificar tablas, registros, saldos o evidencias.
- Se perdería el control sobre la integridad del sistema.

3. Exposición de información sensible

Sin vistas seguras:

- Aplicaciones normales podrían ver datos confidenciales.
- Errores de programación podrían exponer información protegida.

4. Datos robados en texto claro

Sin encriptación:

- Respaldo o archivos robados revelarían datos financieros y personales.
- Se expondrían números de cuenta, identificaciones y movimientos.

Impacto general

No aplicar estas medidas podría ocasionar:

- Pérdida financiera directa.
- Daño reputacional severo.
- Incumplimiento legal y sanciones.
- Riesgo de fraude, robo y manipulación de información.
- Pérdida de confianza por parte de los clientes.

Estas contramedidas fortalecen la seguridad, reducen el riesgo y garantizan la protección de los datos críticos del banco.