



# Proyecto Final de Bases de Datos

Tarea presentada por:

Edgar Montiel Ledesma 317317794  
Carlos Daniel Cortes Jimenez 420004846

Facultad de Ciencias  
Universidad Nacional Autónoma de México  
Fecha de Entrega: 11 de Diciembre de 2023

# Base de Datos: Sistema de Administración de Tienda de Videojuegos en Línea

## 1. Lista de Requerimientos

### 1. Registro y Gestión de Clientes:

- Permitir la creación, actualización y eliminación de registros de clientes.
- Almacenar información relevante: nombre, dirección, contacto, etc.

### 2. Gestión de Empleados:

- Mantener un registro de empleados con sus roles y detalles de contacto.
- Permitir agregar, editar y eliminar empleados.

### 3. Catálogo de Juegos:

- Almacenar información detallada sobre los juegos disponibles, como título, género, plataforma.
- Mantener un control del stock disponible para cada juego.

### 4. Gestión de Pedidos:

- Permitir a los clientes realizar pedidos.
- Seguir el estado de los pedidos (pendiente, enviado, entregado).
- Mantener un registro histórico de los pedidos.

### 5. Control de Inventario:

- Actualizar automáticamente el inventario al realizar ventas o recibir nuevos juegos.

- Notificar cuando el stock de un juego esté bajo para reabastecimiento.

#### 6. Registro de Ventas:

- Registrar todas las transacciones de ventas, incluyendo detalles como el empleado que realizó la venta, el cliente y el monto total.

#### 7. Sistema de Venta de Juegos por Clientes:

- Permitir a los clientes vender juegos de su propiedad.
- Establecer un sistema para valorar los juegos y otorgar un saldo al cliente por la venta.
- Registrar las transacciones de venta de juegos por parte de los clientes.

#### 8. Saldo para Clientes:

- Mantener un saldo para cada cliente, reflejando el dinero obtenido por la venta de juegos.
- Permitir a los clientes utilizar este saldo para comprar otros juegos en la tienda.

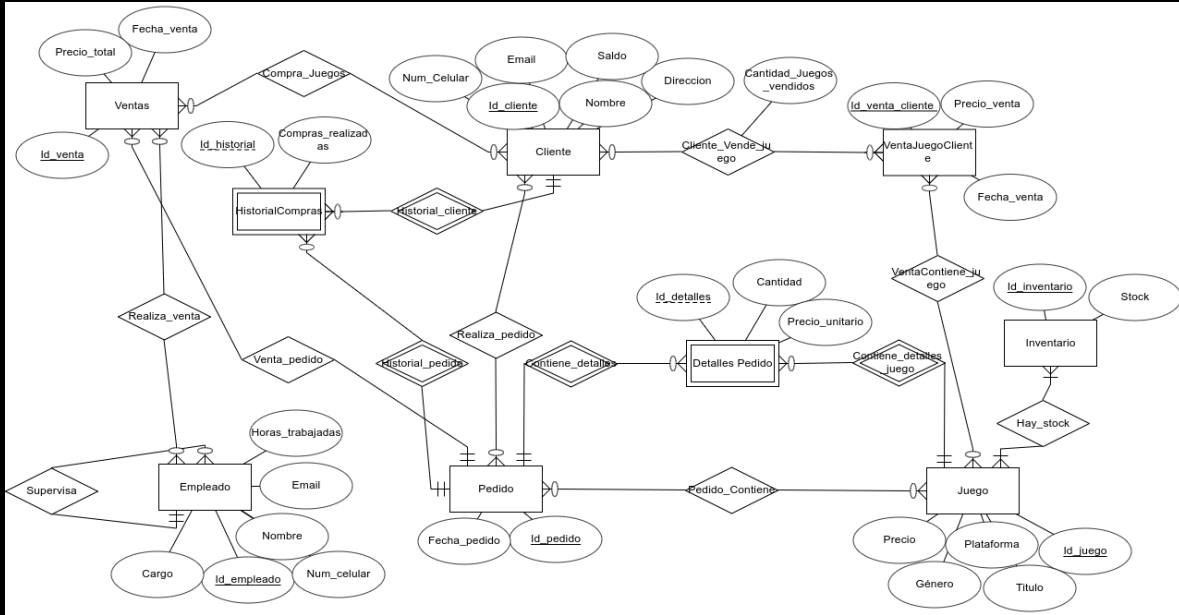
#### 9. Proceso de Compra con Saldo:

- Habilitar la opción para que los clientes utilicen su saldo disponible al comprar juegos.
- Actualizar el saldo del cliente después de cada compra utilizando el saldo disponible como método de pago.

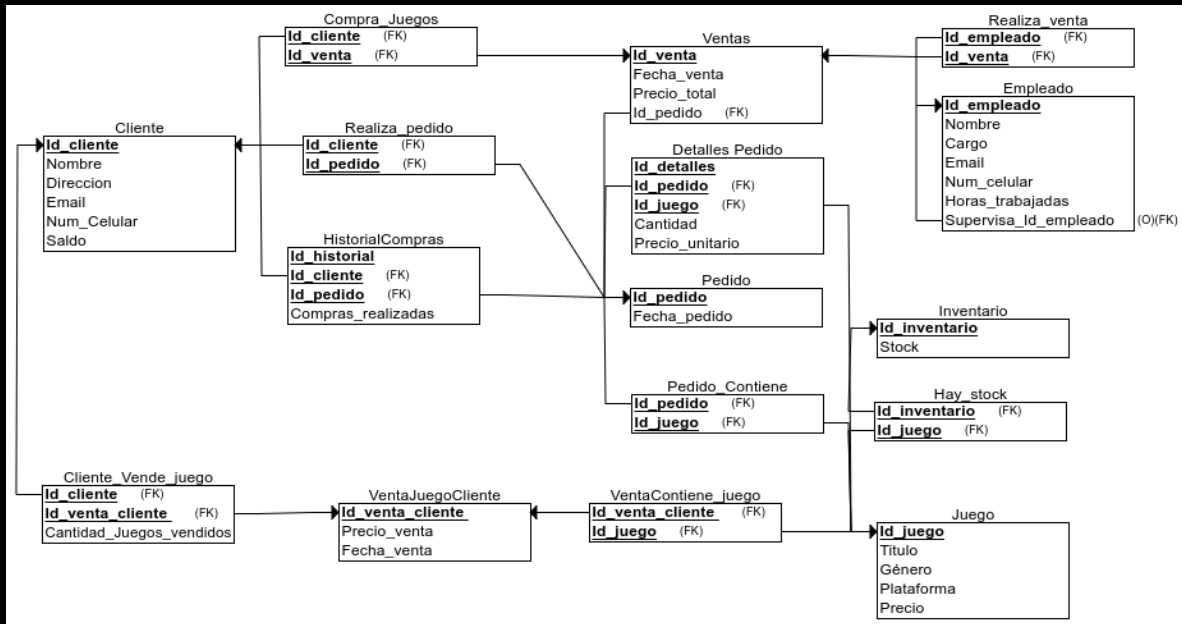
#### 10. Gestión de Saldo y Transacciones::

- Mantener un registro detallado de las transacciones de saldo para cada cliente.
- Permitir a los clientes verificar su saldo actual en la plataforma.

## 2. Modelo Conceptual (Notación de Peter Chen)



### 3. Modelo Relacional



### 4. Script Completo para Crear la Base de Datos

```

1 CREATE TABLE Cliente
2 (
3     Id_cliente INT NOT NULL,
4     NombreC VARCHAR(30) NOT NULL,
5     Direccion VARCHAR(30),
6     EmailC VARCHAR(255) NOT NULL,
7     Num_celularC VARCHAR(15) NOT NULL,
8     Saldo DECIMAL(10, 2) NOT NULL,
9     PRIMARY KEY (Id_cliente)
10 );
11
12 CREATE TABLE Empleado
13 (
14     Id_empleado INT NOT NULL,
15     NombreE VARCHAR(30) NOT NULL,
16     Cargo VARCHAR(15) NOT NULL,
17     EmailE VARCHAR(255) NOT NULL,
18     Num_celularE VARCHAR(15) NOT NULL,
19     Horas_trabajadas INT NOT NULL,
20     Supervisa_Id_empleado INT,
21     PRIMARY KEY (Id_empleado)
22 );

```

```
23
24 ALTER TABLE Empleado
25 ADD CONSTRAINT fk_supervisa_empleado
26 FOREIGN KEY (Supervisa_Id_empleado) REFERENCES Empleado(
    ↳ Id_empleado);
27
28
29 CREATE TABLE Juego
30 (
31     Id_juego INT NOT NULL,
32     Titulo VARCHAR(100) NOT NULL,
33     Genero VARCHAR(50) NOT NULL,
34     Plataforma VARCHAR(50) NOT NULL,
35     Precio DECIMAL(10, 2) NOT NULL,
36     PRIMARY KEY (Id_juego)
37 );
38
39 CREATE TABLE Inventario
40 (
41     Id_inventario INT NOT NULL,
42     Stock INT NOT NULL,
43     PRIMARY KEY (Id_inventario)
44 );
45
46 CREATE TABLE Pedido
47 (
48     Id_pedido INT NOT NULL,
49     Fecha_pedido DATE NOT NULL,
50     Estado_pedido VARCHAR(50) NOT NULL,
51     PRIMARY KEY (Id_pedido)
52 );
53
54 CREATE TABLE Ventas
55 (
56     Id_venta INT NOT NULL,
57     Fecha_venta DATE NOT NULL,
58     Precio_total DECIMAL(10, 2) NOT NULL,
59     Id_pedido INT NOT NULL,
60     PRIMARY KEY (Id_venta),
61     FOREIGN KEY (Id_pedido) REFERENCES Pedido(Id_pedido) ON
        ↳ DELETE CASCADE ON UPDATE CASCADE
62 );
63
```

```

64 CREATE TABLE VentaJuegoCliente
65 (
66     Id_venta_cliente INT NOT NULL,
67     Estado_venta VARCHAR(50) NOT NULL,
68     Precio_venta DECIMAL(10, 2) NOT NULL,
69     Fecha_venta DATE NOT NULL,
70     PRIMARY KEY (Id_venta_cliente)
71 );
72
73
74 CREATE TABLE HistorialCompras
75 (
76     Compras_realizadas INT NOT NULL,
77     Id_historial INT NOT NULL,
78     Id_cliente INT NOT NULL,
79     Id_pedido INT NOT NULL,
80     PRIMARY KEY (Id_historial, Id_cliente, Id_pedido),
81     FOREIGN KEY (Id_cliente) REFERENCES Cliente(Id_cliente)
      ↳ ON DELETE CASCADE ON UPDATE CASCADE,
82     FOREIGN KEY (Id_pedido) REFERENCES Pedido(Id_pedido) ON
      ↳ DELETE CASCADE ON UPDATE CASCADE
83 );
84
85 CREATE TABLE Realiza_pedido
86 (
87     Id_cliente INT NOT NULL,
88     Id_pedido INT NOT NULL,
89     PRIMARY KEY (Id_cliente, Id_pedido),
90     FOREIGN KEY (Id_cliente) REFERENCES Cliente(Id_cliente)
      ↳ ON DELETE CASCADE ON UPDATE CASCADE,
91     FOREIGN KEY (Id_pedido) REFERENCES Pedido(Id_pedido) ON
      ↳ DELETE CASCADE ON UPDATE CASCADE
92 );
93
94 CREATE TABLE Pedido_Contiene
95 (
96     Id_pedido INT NOT NULL,
97     Id_juego INT NOT NULL,
98     PRIMARY KEY (Id_pedido, Id_juego),
99     FOREIGN KEY (Id_pedido) REFERENCES Pedido(Id_pedido) ON
      ↳ DELETE CASCADE ON UPDATE CASCADE,
100    FOREIGN KEY (Id_juego) REFERENCES Juego(Id_juego) ON
      ↳ DELETE CASCADE ON UPDATE CASCADE

```

```

101 );
102
103 CREATE TABLE Hay_stock
104 (
105     Id_inventario INT NOT NULL,
106     Id_juego INT NOT NULL,
107     PRIMARY KEY (Id_inventario, Id_juego),
108     FOREIGN KEY (Id_inventario) REFERENCES Inventario(
109         ↪ Id_inventario) ON DELETE CASCADE ON UPDATE CASCADE,
110     FOREIGN KEY (Id_juego) REFERENCES Juego(Id_juego) ON
111         ↪ DELETE CASCADE ON UPDATE CASCADE
112 );
113
114 CREATE TABLE Cliente_Vende_juego
115 (
116     Cantidad_Juegos_vendidos INT NOT NULL,
117     Id_cliente INT NOT NULL,
118     Id_venta_cliente_ INT NOT NULL,
119     PRIMARY KEY (Id_cliente, Id_venta_cliente_),
120     FOREIGN KEY (Id_cliente) REFERENCES Cliente(Id_cliente)
121         ↪ ON DELETE CASCADE ON UPDATE CASCADE,
122     FOREIGN KEY (Id_venta_cliente_) REFERENCES
123         ↪ VentaJuegoCliente(Id_venta_cliente_) ON DELETE
124         ↪ CASCADE ON UPDATE CASCADE
125 );
126
127 CREATE TABLE VentaContiene_juego
128 (
129     Id_venta_cliente_ INT NOT NULL,
130     Id_juego INT NOT NULL,
131     PRIMARY KEY (Id_venta_cliente_, Id_juego),
132     FOREIGN KEY (Id_venta_cliente_) REFERENCES
133         ↪ VentaJuegoCliente(Id_venta_cliente_) ON DELETE
134         ↪ CASCADE ON UPDATE CASCADE,
135     FOREIGN KEY (Id_juego) REFERENCES Juego(Id_juego) ON
136         ↪ DELETE CASCADE ON UPDATE CASCADE
137 );
138
139 CREATE TABLE Detalles_Pedido
140 (
141     Id_detalle INT NOT NULL,
142     Cantidad INT NOT NULL,
143     Precio_unitario DECIMAL(10, 2) NOT NULL,

```



```

136     Id_pedido INT NOT NULL,
137     Id_juego INT NOT NULL,
138     PRIMARY KEY (Id_detalle, Id_pedido, Id_juego),
139     FOREIGN KEY (Id_pedido) REFERENCES Pedido(Id_pedido) ON
        ↳ DELETE CASCADE ON UPDATE CASCADE,
140     FOREIGN KEY (Id_juego) REFERENCES Juego(Id_juego) ON
        ↳ DELETE CASCADE ON UPDATE CASCADE
141 );
142
143 CREATE TABLE Compra_Juegos
144 (
145     Id_cliente INT NOT NULL,
146     Id_venta INT NOT NULL,
147     PRIMARY KEY (Id_cliente, Id_venta),
148     FOREIGN KEY (Id_cliente) REFERENCES Cliente(Id_cliente)
        ↳ ON DELETE CASCADE ON UPDATE CASCADE,
149     FOREIGN KEY (Id_venta) REFERENCES Ventas(Id_venta) ON
        ↳ DELETE CASCADE ON UPDATE CASCADE
150 );
151
152 CREATE TABLE Realiza_venta
153 (
154     Id_empleado INT NOT NULL,
155     Id_venta INT NOT NULL,
156     PRIMARY KEY (Id_empleado, Id_venta),
157     FOREIGN KEY (Id_empleado) REFERENCES Empleado(Id_empleado
        ↳ ) ON DELETE CASCADE ON UPDATE CASCADE,
158     FOREIGN KEY (Id_venta) REFERENCES Ventas(Id_venta) ON
        ↳ DELETE CASCADE ON UPDATE CASCADE
159 );

```

## 5. Script de Inserción de Datos (para 100 registros)

La inserción de datos, se encuentra en el archivo `Base_de_Datos.sql`, no se colocaron los 100 registros para la tabla Empleado ya que tenemos una tienda en línea, además de que no puede haber demasiados cargos de los que se realizaron en el script.

## 6. Evidencia de Restricciones de Integridad Referencial

Evidencia 1 de integridad referencial:

6.a Tablas involucradas en la restricción.

- Realiza\_venta.
- Empleado.

6.b FK de la tabla que referencia y PK de la tabla referenciada.

- Realiza\_venta.Id\_empleado (FK) hace referencia a Empleado.Id\_empleado (PK)

6.c Justificación del trigger de integridad referencial elegido.

Si un empleado es eliminado de la tabla Empleado, esta restricción asegura que se eliminen automáticamente los registros correspondientes en Realiza\_venta para mantener la integridad de la base de datos.

6.d Instrucción UPDATE o DELETE que permita evidenciar que la restricción está funcionando.

```
1 DELETE FROM Empleado WHERE Id_empleado = 203;
```

```
evidenciastiendav=# DELETE FROM Empleado WHERE Id_empleado = 203;
DELETE 1
evidenciastiendav=# █
```

6.e Captura de pantalla con el resultado de la instrucción que muestre que la restricción está funcionando.

**Antes de aplicar DELETE:**

```
evidenciastiendav=# SELECT * FROM realiza_venta;
 id_empleado | id_venta
-----+-----
          203 |         601
          203 |         602
          203 |         603
          207 |         605
          206 |         606
          210 |         607
          207 |         608
          203 |         609
          211 |         610
          207 |         604
(10 filas)
```

Después de aplicar DELETE:

```
evidenciastiendav=# DELETE FROM Empleado WHERE Id_empleado = 203;
DELETE 1
evidenciastiendav=# SELECT * FROM realiza_venta;
 id_empleado | id_venta
-----+-----
          207 |         605
          206 |         606
          210 |         607
          207 |         608
          211 |         610
          207 |         604
(6 filas)
evidenciastiendav=# █
```

## Evidencia 2 de integridad referencial:

6.a Tablas involucradas en la restricción.

- Pedido\_Contiene.
- Pedido.

6.b FK de la tabla que referencia y PK de la tabla referenciada.

- Pedido\_Contiene.Id\_pedido (FK) hace referencia a Pedido.Id\_pedido (PK)

6.c Justificación del trigger de integridad referencial elegido.

Para mantener la integridad de los pedidos contenidos en la tabla Pedido\_Contiene, se desea que al eliminar un pedido de la tabla Pedido, los registros asociados en Pedido\_Contiene también se eliminen.

6.d Instrucción UPDATE o DELETE que permita evidenciar que la restricción está funcionando.

```
1 DELETE FROM Pedido WHERE Id_pedido = 501;
```

```
evidenciastiendav=# DELETE FROM Pedido WHERE Id_pedido = 501;
DELETE 1
```

- 6.e Captura de pantalla con el resultado de la instrucción que muestre que la restricción está funcionando.

Antes de aplicar DELETE:

```
evidenciastiendav=# SELECT * FROM pedido_contiene;
 id_pedido | id_juego
-----+-----
          501 |          301
          502 |          302
          503 |          303
          504 |          304
          505 |          305
          506 |          306
          507 |          307
          508 |          308
          509 |          309
          510 |          310
(10 filas)
```

Después de aplicar DELETE:

```
evidenciastiendav=# DELETE FROM Pedido WHERE Id_pedido = 501;
DELETE 1
evidenciastiendav=# SELECT * FROM pedido_contiene;
 id_pedido | id_juego
-----+-----
          502 |          302
          503 |          303
          504 |          304
          505 |          305
          506 |          306
          507 |          307
          508 |          308
          509 |          309
          510 |          310
(9 filas)
```

### Evidencia 3 de integridad referencial:

6.a Tablas involucradas en la restricción.

- VentaContiene\_juego.
- Juego.

6.b FK de la tabla que referencia y PK de la tabla referenciada.

- VentaContiene\_juego.Id\_juego (FK) hace referencia a Juego.Id\_juego (PK)

6.c Justificación del trigger de integridad referencial elegido.

Se busca mantener la integridad de la tabla VentaContiene\_juego. Si el ID de un juego se modifica en la tabla Juego, debe reflejarse automáticamente en la tabla VentaContiene\_juego.

6.d Instrucción UPDATE o DELETE que permita evidenciar que la restricción está funcionando.

```
1 UPDATE Juego SET Id_juego = 350 WHERE Id_juego = 301;
```

```
evidenciastiendav=# UPDATE Juego SET Id_juego = 350 WHERE Id_juego = 301;
UPDATE 1
```

6.e Captura de pantalla con el resultado de la instrucción que muestre que la restricción está funcionando.

**Antes de aplicar UPDATE:**

```
evidenciastiendav=# SELECT * FROM ventacontiene_juego;
 id_venta_cliente | id_juego
-----+-----
          701 |      301
          702 |      302
          703 |      303
          704 |      304
          705 |      305
          706 |      306
          707 |      307
          708 |      308
          709 |      309
          710 |      310
(10 filas)
```

Después de aplicar UPDATE:

```
evidenciastiendav=# UPDATE Juego SET Id_juego = 350 WHERE Id_juego = 301;
UPDATE 1
evidenciastiendav=# SELECT * FROM ventacontiene_juego;
 id_venta_cliente | id_juego
-----+-----
                702 |      302
                703 |      303
                704 |      304
                705 |      305
                706 |      306
                707 |      307
                708 |      308
                709 |      309
                710 |      310
                701 |      350
(10 filas)
```

Evidencia 4 de integridad referencial:

6.a Tablas involucradas en la restricción.

- Cliente\_Vende\_juego.
- Cliente.

6.b FK de la tabla que referencia y PK de la tabla referenciada.

- Cliente\_Vende\_juego.Id\_cliente (FK) hace referencia a Cliente.Id\_cliente (PK)

6.c Justificación del trigger de integridad referencial elegido.

Garantizar que si el ID de un cliente cambia en la tabla Cliente, se actualicen automáticamente los registros correspondientes en la tabla Cliente\_Vende\_juego

6.d Instrucción UPDATE o DELETE que permita evidenciar que la restricción está funcionando.

```
1 UPDATE Cliente SET Id_cliente = 150 WHERE Id_cliente =
   ↪ 101;
```

```
evidenciastiendav=# UPDATE Cliente SET Id_cliente = 150 WHERE Id_cliente = 101;
UPDATE 1
```

6.e Captura de pantalla con el resultado de la instrucción que muestre que la restricción está funcionando.

Antes de aplicar UPDATE:

```
evidenciastiendav=# SELECT * FROM cliente_vende_juego;
cantidad_juegos_vendidos | id_cliente | id_venta_cliente
-----+-----+-----
5 | 101 | 701
2 | 102 | 702
2 | 103 | 703
8 | 104 | 704
4 | 105 | 705
1 | 106 | 706
1 | 107 | 707
1 | 108 | 708
9 | 109 | 709
10 | 110 | 710
(10 filas)
```

Después de aplicar UPDATE:

```
evidenciastiendav=# UPDATE Cliente SET Id_cliente = 150 WHERE Id_cliente = 101;
UPDATE 1
evidenciastiendav=# SELECT * FROM cliente_vende_juego;
cantidad_juegos_vendidos | id_cliente | id_venta_cliente
-----+-----+-----
2 | 102 | 702
2 | 103 | 703
8 | 104 | 704
4 | 105 | 705
1 | 106 | 706
1 | 107 | 707
1 | 108 | 708
9 | 109 | 709
10 | 110 | 710
5 | 150 | 701
(10 filas)
```

## 7. Evidencia de Restricciones CHECK

Evidencia 1 CHECK:

7.a Tablas involucradas en la restricción.

- Cliente.

7.b Atributo elegido.

- Num\_celularC

7.c Breve descripción de la restricción.

La restricción CHECK asegura que el número de celular tenga exactamente 10 dígitos.

7.d Instrucción para la creación de la restricción.

```
1      ALTER TABLE Cliente
2      ADD CONSTRAINT check_length_Num_celularC
3      CHECK (LENGTH(Num_celularC) = 10);
```

```
evidenciastiendav=# ALTER TABLE Cliente
ADD CONSTRAINT check_length_Num_celularC CHECK (LENGTH(Num_celularC) = 10);
ALTER TABLE
```

7.e Instrucción que permita evidenciar que la restricción esta funcionando.

```
1      INSERT INTO Cliente (Id_cliente, NombreC, Direccion
    ↪      , EmailC, Num_celularC, Saldo)
2      VALUES (111, 'Ejemplo', 'Calle Ejemplo', '
    ↪      ejemplo@email.com', '123456738972', 150.00);
```

```
evidenciastiendav=# INSERT INTO Cliente (Id_cliente, NombreC, Direccion, EmailC, Num_celularC, Saldo)
VALUES (111, 'Ejemplo', 'Calle Ejemplo', 'ejemplo@email.com', '123456738972', 150.00);
```

7.f Captura de pantalla con el resultado de la instrucción que muestre que la restricción está funcionando.

```
evidenciastiendav=# INSERT INTO Cliente (Id_cliente, NombreC, Direccion, EmailC, Num_celularC, Saldo)
VALUES (111, 'Ejemplo', 'Calle Ejemplo', 'ejemplo@email.com', '123456738972', 150.00);
ERROR: el nuevo registro para la relación «cliente» viola la restricción «check» «check_length_num_celularc»
DETALLE: La fila que falla contiene (111, Ejemplo, Calle Ejemplo, ejemplo@email.com, 123456738972, 150.00).
```

## Evidencia 2 CHECK:

7.a Tablas involucradas en la restricción.

- Empleado.



7.b Atributo elegido.

- Horas\_trabajadas

7.c Breve descripción de la restricción.

La restricción CHECK asegura que las horas trabajadas estén entre 0 y 50.

7.d Instrucción para la creación de la restricción.

```
1 ALTER TABLE Empleado
2 ADD CONSTRAINT chk_Horas_trabajadas_range
3 CHECK (Horas_trabajadas BETWEEN 0 AND 50);
```

```
evidenciastiendav=# ALTER TABLE Empleado
ADD CONSTRAINT chk_Horas_trabajadas_range CHECK (Horas_trabajadas BETWEEN 0 AND 50);
ALTER TABLE
```

7.e Instrucción que permita evidenciar que la restricción esta funcionando.

```
1 UPDATE Empleado SET Horas_trabajadas = 60 WHERE
   ↳ Id_empleado = 201;
```

```
evidenciastiendav=# UPDATE Empleado SET Horas_trabajadas = 60 WHERE Id_empleado = 201;
```

7.f Captura de pantalla con el resultado de la instrucción que muestre que la restricción está funcionando.

```
evidenciastiendav=# UPDATE Empleado SET Horas_trabajadas = 60 WHERE Id_empleado = 201;
ERROR:  el nuevo registro para la relación «empleado» viola la restricción «check» «chk_horas_trabajadas_range»
DETALLE:  La fila que falla contiene (201, Gabriel Garcia, Gerente, gabriel@email.com, 2345678901, 60, null).
```

### Evidencia 3 CHECK:

7.a Tablas involucradas en la restricción.

- Juego.

7.b Atributo elegido.

- Precio.

7.c Breve descripción de la restricción.

La restricción CHECK asegura que el precio esté entre 0 y 100.

7.d Instrucción para la creación de la restricción.

```
1      ALTER TABLE Juego
2      ADD CONSTRAINT chk_Precio_range
3      CHECK (Precio BETWEEN 0 AND 100);
```

```
evidenciastiendav=# ALTER TABLE Juego
ADD CONSTRAINT chk_Precio_range CHECK (Precio BETWEEN 0 AND 100);
ALTER TABLE
```

7.e Instrucción que permita evidenciar que la restricción esta funcionando.

```
1      UPDATE Juego SET Precio = 120 WHERE Id_juego = 302;
```

```
evidenciastiendav=# UPDATE Juego SET Precio = 120 WHERE Id_juego = 302;
```

7.f Captura de pantalla con el resultado de la instrucción que muestre que la restricción está funcionando.

```
evidenciastiendav=# UPDATE Juego SET Precio = 120 WHERE Id_juego = 302;
ERROR:  el nuevo registro para la relación «juego» viola la restricción «check» «chk_precio_range»
DETALLE:  La fila que falla contiene (302, Red Dead Redemption 2, Acción-Aventura, PlayStation, 120.00).
```

## 8. Evidencia de Dominios Personalizados

### Evidencia 1 DOMAIN:

8.a Tabla elegida.

- Empleado.

8.b Atributo elegido.

- Horas\_trabajadas

8.c Breve descripción del dominio y de la restricción check propuesta.

Se establecerá un límite mínimo y máximo de horas trabajadas para los empleados. Esto asegurará que las horas reportadas sean razonables y estén dentro de un rango aceptable.

8.d Instrucción para la creación del dominio personalizado.

```
1      CREATE DOMAIN HorasTrabajadasDomain INT CHECK (
      ↪ VALUE BETWEEN 0 AND 50);
2      ALTER TABLE Empleado ALTER COLUMN
      ↪ Horas_trabajadas SET DATA TYPE
      ↪ HorasTrabajadasDomain;
```

```
evidenciastlendav=# CREATE DOMAIN HorasTrabajadasDomain INT CHECK (VALUE BETWEEN 0 AND 50);
ALTER TABLE Empleado ALTER COLUMN Horas_trabajadas SET DATA TYPE HorasTrabajadasDomain;
CREATE DOMAIN
ALTER TABLE
```

8.e Captura de pantalla de la estructura de la tabla donde se muestre el dominio personalizado en uso.

```
1      INSERT INTO Empleado (Id_empleado, NombreE,
      ↪ Cargo, EmailE, Num_celularE,
      ↪ Horas_trabajadas, Supervisa_Id_empleado)
2      VALUES (240, 'Rene Mendez', 'Supervisor de
      ↪ Ventas', 'ReneM@email.com', '1312345678',
      ↪ 55, 201);
```

```
evidenciastlendav=# INSERT INTO Empleado (Id_empleado, NombreE, Cargo, EmailE, Num_celularE, Horas_trabajadas, Supervisa_Id_empleado) VALUES
(240, 'Rene Méndez', 'Supervisor de Ventas', 'ReneM@email.com', '1312345678', 55, 201);
ERROR:  el valor para el dominio horastrabajadasdomain viola la restricción «check» «horastrabajadasdomain_check»
```

## Evidencia 2 DOMAIN:

8.a Tabla elegida.

- Cliente.

8.b Atributo elegido.

- Saldo

8.c Breve descripción del dominio y de la restricción check propuesta.

Limitar el saldo de los clientes para que no pueda ser negativo. Esto garantiza que no se registren saldos negativos en las cuentas de los clientes.

8.d Instrucción para la creación del dominio personalizado.

```

1      CREATE DOMAIN PrecioDomain DECIMAL(10, 2) CHECK
      ↪ (VALUE >= 0);
2      ALTER TABLE Juego ALTER COLUMN Precio SET DATA
      ↪ TYPE PrecioDomain;

```

```

evidenciastiendav=# CREATE DOMAIN SaldoDomain DECIMAL(10, 2) CHECK (VALUE >= 0);
ALTER TABLE Cliente ALTER COLUMN Saldo SET DATA TYPE SaldoDomain;
CREATE DOMAIN
ALTER TABLE

```

8.e Captura de pantalla de la estructura de la tabla donde se muestre el dominio personalizado en uso.

```

1      INSERT INTO Cliente (Id_cliente, NombreC,
      ↪ Direccion, EmailC, Num_celularC, Saldo)
2      VALUES (120, 'Fernando Perez', 'Calle 31,
      ↪ Ciudad 3K', 'fernandP@email.com', '
      ↪ 6624113323', -10.00);

```

```

evidenciastiendav=# INSERT INTO Cliente (Id_cliente, NombreC, Direccion, EmailC, Num_celularC, Saldo) VALUES (120, 'Fernando Perez', 'Calle
31, Ciudad 3K', 'fernandP@email.com', '6624113323', -10.00);
ERROR:  el valor para el dominio saldodomain viola la restricción «check» «saldodomain_check»

```

### Evidencia 3 DOMAIN:

8.a Tabla elegida.

- Juego.

8.b Atributo elegido.

- Precio

8.c Breve descripción del dominio y de la restricción check propuesta.

Limitar el precio de un juego para que no sea inferior a cierto valor establecido. Esto asegura que los precios de los juegos sean razonables.

8.d Instrucción para la creación del dominio personalizado.

```

1      CREATE DOMAIN PrecioDomain DECIMAL(10, 2) CHECK
      ↪ (VALUE >= 0);
2      ALTER TABLE Juego ALTER COLUMN Precio SET DATA
      ↪ TYPE PrecioDomain;

```

```

evidenciastiendav=# CREATE DOMAIN PrecioDomain DECIMAL(10, 2) CHECK (VALUE >= 0);
ALTER TABLE Juego ALTER COLUMN Precio SET DATA TYPE PrecioDomain;
CREATE DOMAIN
ALTER TABLE

```

8.e Captura de pantalla de la estructura de la tabla donde se muestre el dominio personalizado en uso.

```

1      INSERT INTO Juego (Id_juego, Titulo, Genero,
      ↪ Plataforma, Precio)
2      VALUES (320, 'The last faith', 'Metroidvania',
      ↪ 'Nintendo Switch', -14.99);

```

```

evidenciastiendav=# INSERT INTO Juego (Id_juego, Titulo, Genero, Plataforma, Precio) VALUES (320, 'The last faith', 'Metroidvania', 'Nintendo Switch', -14.99);
ERROR:  el valor para el dominio preciodomain viola la restricción «check» «preciodomain_check»

```

## 9. Evidencia de Restricciones para Tuplas

### Evidencia 1 Tuplas:

9.a Tabla elegida.

- Cliente.

9.b Breve descripción de la restricción.

- La restricción **NOT NULL** se aplica al campo 'Direccion' en la tabla Cliente para asegurarse de que este campo no esté vacío.

9.c Instrucción para la creación de la restricción.

```

1      ALTER TABLE Cliente
2      ALTER COLUMN Direccion SET NOT NULL;

```

```

ALTER COLUMN Direccion SET NOT NULL;
ALTER TABLE

```

9.d Instrucción “Insert” o “Update” que permita evidenciar que la restricción esta funcionando.

```

1      INSERT INTO Cliente (Id_cliente, NombreC,
      ↪ Direccion, EmailC, Num_celularC, Saldo)
2      VALUES (115, 'Jose Lopez', NULL, 'JoseLO@email.
      ↪ com', '4321567890', 150.00);

```

```

evidenciastiendav=# INSERT INTO Cliente (Id_cliente, NombreC, Direccion, EmailC, Num_celularC, Saldo) VALUES (115, 'Jose Lopez', NULL, 'Jose
LO@email.com', '4321567890', 150.00);

```

9.e Captura de pantalla con el resultado de la instrucción que muestre que la restricción está funcionando.

```

evidenciastiendav=# INSERT INTO Cliente (Id_cliente, NombreC, Direccion, EmailC, Num_celularC, Saldo) VALUES (115, 'Jose Lopez', NULL, 'Jose
LO@email.com', '4321567890', 150.00);
ERROR:  el valor nulo en la columna «direccion» de la relación «cliente» viola la restricción de no nulo
DETALLE: La fila que falla contiene (115, Jose Lopez, null, JoseLO@email.com, 4321567890, 150.00).

```

## Evidencia 2 Tuplas:

9.a Tabla elegida.

- Empleado.

9.b Breve descripción de la restricción.

- La restricción **UNIQUE** garantiza que el campo Correo en la tabla Empleado no tenga valores duplicados, es decir, cada dirección de correo electrónico debe ser única en la tabla.

9.c Instrucción para la creación de la restricción.

```

1      ALTER TABLE Empleado
2      ADD CONSTRAINT unique_email
3      UNIQUE (EmailE);

```

```

evidenciastiendav=# ALTER TABLE Empleado
ADD CONSTRAINT unique_email
UNIQUE (EmailE);
ALTER TABLE

```

9.d Instrucción “Insert” o “Update” que permita evidenciar que la restricción esta funcionando.

```

1      INSERT INTO Empleado (Id_empleado, NombreE,
      ↪ Cargo, EmailE, Num_celularE,
      ↪ Horas_trabajadas, Supervisa_Id_empleado)
2      VALUES (216, 'Victoria Cortes', 'Vendedor', '
      ↪ victoria@email.com', '7654890123', 40,
      ↪ 213);

```

```

evidenciastlendav=# INSERT INTO Empleado (Id_empleado, NombreE, Cargo, EmailE, Num_celularE, Horas_trabajadas, Supervisa_Id_empleado) VALUES
(203, 'Victoria Martinez', 'Vendedor', 'victoria@email.com', '4567890123', 42, 202);

```

- 9.e Captura de pantalla con el resultado de la instrucción que muestre que la restricción está funcionando.

```

evidenciastlendav=# INSERT INTO Empleado (Id_empleado, NombreE, Cargo, EmailE, Num_celularE, Horas_trabajadas, Supervisa_Id_empleado) VALUES
(203, 'Victoria Martinez', 'Vendedor', 'victoria@email.com', '4567890123', 42, 202);
ERROR: llave duplicada viola restricción de unicidad «unique_email»
DETALLE: Ya existe la llave (email)=(victoria@email.com).

```

## 10. Consultas Relevantes

### Consulta 1:

- 10.a Redacción clara de la consulta.

Obtener el nombre y el saldo de los clientes que han realizado compras y cuyo saldo sea mayor a 100.

- 10.b Código en lenguaje SQL de la consulta.

```

1      SELECT NombreC, Saldo
2      FROM Cliente
3      WHERE Id_cliente IN (
4      SELECT DISTINCT Id_cliente
5      FROM Compra_Juegos) AND Saldo > 100;

```

```

tiendavideojuegos=# SELECT NombreC, Saldo
FROM Cliente
WHERE Id_cliente IN (
    SELECT DISTINCT Id_cliente
    FROM Compra_Juegos
) AND Saldo > 100;

```

- 10.c Ejecutar la consulta en Postgres e incluir una captura de pantalla con el resultado de la consulta.

nombrec	saldo
Maria García	200.00
Ana Martínez	120.00
Sofia Ramírez	180.00
Laura Castillo	210.00
Pedro Díaz	170.00
Gabriela Vargas	130.00
Jorge Ruiz	190.00
Fernanda Gómez	160.00
Andrés Castro	140.00
Valentina Navarro	220.00
Ricardo Mendoza	115.00
Daniela Torres	205.00
Luis Rodríguez	175.00
Isabel Silva	185.00
Martín Gutiérrez	195.00
Carmen Reyes	175.00
Hugo Sánchez	165.00
Silvia Pérez	225.00
Raúl Hernández	140.00
Carolina Martínez	200.00
Roberto Gómez	155.00
Lucía Díaz	210.00
Eduardo Vargas	180.00
Adriana Ruiz	170.00
Óscar Castro	190.00
Marisol Navarro	205.00
Gonzalo Mendoza	195.00
Valeria Torres	175.00
Mateo López	160.00
Renata González	185.00
Arturo Sánchez	170.00
Camila Rivera	195.00
Bruno Fernández	200.00
Gabriel Silva	185.00
Mariana Gutiérrez	195.00
Ricardo Reyes	175.00
Fernanda Sánchez	165.00
Santiago Pérez	225.00
Luciana Hernández	140.00
Diego Martínez	200.00
Valentina Gómez	155.00
Matías Díaz	210.00
Isabella Vargas	180.00
Julián Ruiz	170.00
Ana Castro	190.00
Manuel Navarro	205.00
Carla Mendoza	195.00
Maximiliano Torres	175.00
Luna López	160.00
Simón González	185.00
Alicia Sánchez	170.00
Lucas Rivera	195.00



Carla Mendoza	195.00
Maximiliano Torres	175.00
Luna López	160.00
Simón González	185.00
Alicia Sánchez	170.00
Lucas Rivera	195.00
Valentino Fernández	200.00
Martina Silva	185.00
Luciano Gutiérrez	195.00
Gabriela Reyes	175.00
Javier Sánchez	165.00
Selena Pérez	225.00
Matias Hernández	140.00
Ariana Martínez	200.00
Facundo Gómez	155.00
Abril Díaz	210.00
Tomás Vargas	180.00
Isadora Ruiz	170.00
Santiago Castro	190.00
Catalina Navarro	205.00
Gael Mendoza	195.00
Martina Torres	175.00
Benjamín López	160.00
Aitana González	185.00
Facundo Sánchez	170.00
Martina Rivera	195.00
Valentina Fernández	200.00
Mateo Silva	185.00
Valeria Gutiérrez	195.00
Joaquín Reyes	175.00
Daniela Sánchez	165.00
Leonardo Pérez	225.00
Isabella Hernández	140.00
Juan Martínez	200.00
Abril Gómez	155.00
Hugo Díaz	210.00
Camila Vargas	180.00
Lautaro Ruiz	170.00
Emilia Castro	190.00
Benicio Navarro	205.00
Renata Mendoza	195.00
Diego Torres	175.00
Mía López	160.00
Santino González	185.00
Abril Sánchez	170.00
Amelia Rivera	195.00
Dante Fernández	200.00
Lucas Silva	185.00
Isabel Gutiérrez	195.00
Agustín Reyes	175.00
Marina Sánchez	165.00
Natalia Fernández	175.00
(98 filas)	

## Consulta 2:

10.a Redacción clara de la consulta.

Mostrar los juegos disponibles en la plataforma "Xbox" junto con su stock actual.

10.b Código en lenguaje SQL de la consulta.

```

1      SELECT J.Titulo , I.Stock
2      FROM Juego J

```

```

3      INNER JOIN Hay_stock H ON J.Id_juego = H.
      ↪ Id_juego
4      INNER JOIN Inventario I ON H.Id_inventario = I.
      ↪ Id_inventario
5      WHERE J.Plataforma = 'Xbox';

```

```

tiendavideojuegos=# SELECT J.Titulo, I.Stock
FROM Juego J
INNER JOIN Hay_stock H ON J.Id_juego = H.Id_juego
INNER JOIN Inventario I ON H.Id_inventario = I.Id_inventario
WHERE J.Plataforma = 'Xbox';

```

10.c Ejecutar la consulta en Postgres e incluir una captura de pantalla con el resultado de la consulta.

titulo	stock
FIFA 22	20
Assassins Creed Valhalla	25
Halo Infinite	15
Forza Horizon 5	30
Doom Eternal	90
Yakuza: Like a Dragon	30
Tom Clancy's Rainbow Six Siege	60
Dead by Daylight	70
Outer Wilds	15
Subnautica	85
The Outer Worlds	10
Mortal Kombat 11	75
Forza Motorsport 7	60
Need for Speed: Heat	25
Resident Evil 7: Biohazard	90
Monster Hunter: World	80
Celeste	50
Rogue Company	60
Cuphead	40
Hollow Knight	75
Ori and the Will of the Wisps	55
(21 filas)	

### Consulta 3:

10.a Redacción clara de la consulta.

Obtener el nombre de los empleado que tengas el cargo de "Supervisor de Ventas" y hayan trabajado mas de 40 horas.

10.b Código en lenguaje SQL de la consulta.

```
1      SELECT NombreE
2      FROM Empleado
3      WHERE Cargo = 'Supervisor de Ventas' AND
           ↳ Horas_trabajadas > 40;
```

```
tiendavideojuegos=# SELECT NombreE
FROM Empleado
WHERE Cargo = 'Supervisor de Ventas' AND Horas_trabajadas > 40;
```

- 10.c Ejecutar la consulta en Postgres e incluir una captura de pantalla con el resultado de la consulta.

```
      nombree
-----
Elena Ramírez
Natalia Vargas
Carolina Díaz
Tomás Méndez
(4 filas)
```

## 11. Vistas Relevantes