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 Méixco
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.

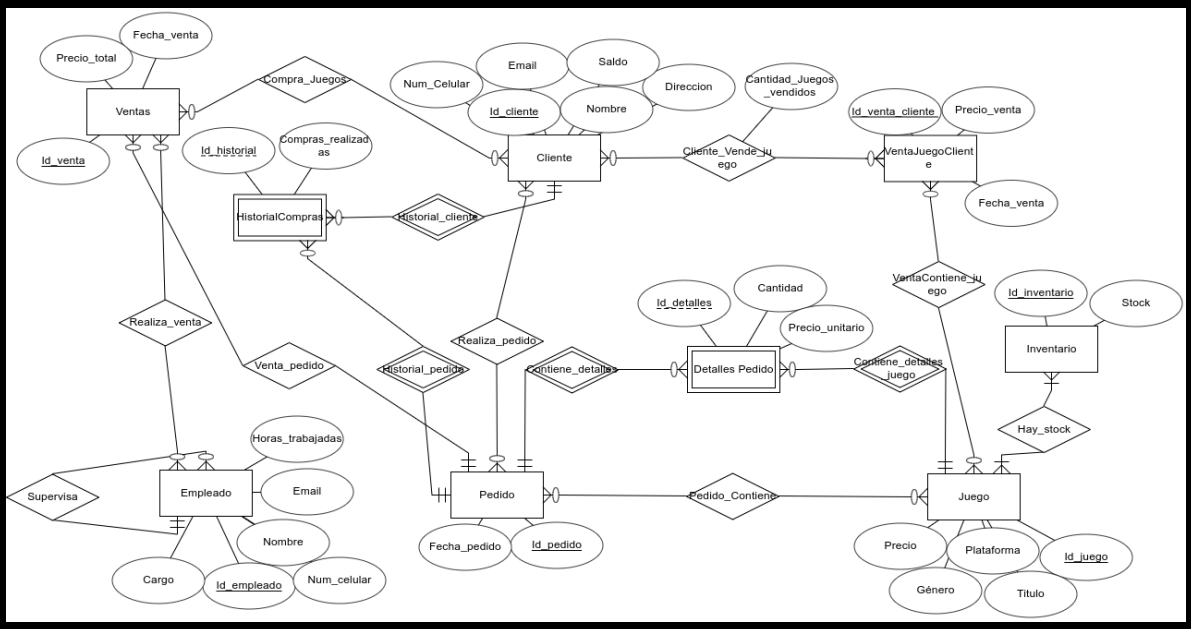
9. Proceso de Compra con Saldo:
    - Habilitar la opción para que los clientes utilicen su saldo disponible al comprar juegos.
    - Actualizar el saldo del cliente después de cada compra utilizando el saldo disponible como método de pago.

10. Gestión de Saldo y Transacciones::

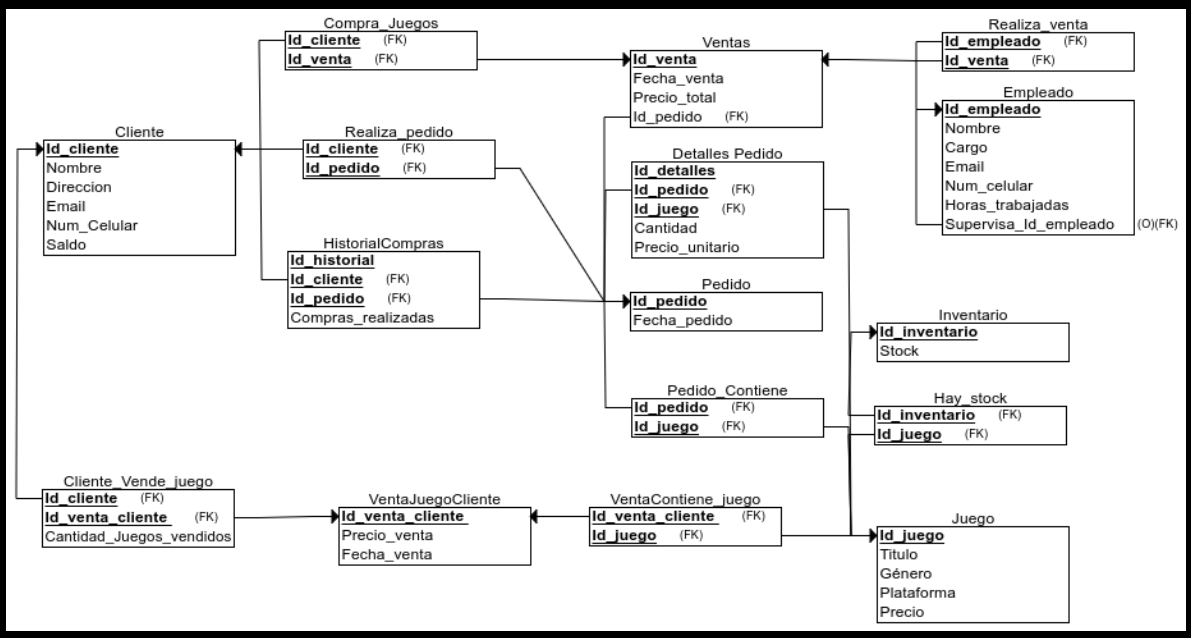
- Mantener un registro detallado de las transacciones de saldo para cada cliente.

- 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

```sql
CREATE TABLE Cliente
(
  Id_cliente INT NOT NULL,
  NombreC VARCHAR(30) NOT NULL,
  Direccion VARCHAR(30),
  EmailC VARCHAR(255) NOT NULL,
  Num_celularC VARCHAR(15) NOT NULL,
  Saldo DECIMAL(10, 2) NOT NULL,
  PRIMARY KEY (Id_cliente)
);

CREATE TABLE Empleado
(
  Id_empleado INT NOT NULL,
  NombreE VARCHAR(30) NOT NULL,
  Cargo VARCHAR(15) NOT NULL,
  EmailE VARCHAR(255) NOT NULL,
  Num_celularE VARCHAR(15) NOT NULL,
  Horas_trabajadas INT NOT NULL,
  Supervisa_Id_empleado INT,
  PRIMARY KEY (Id_empleado)
);
```

```
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
```

```sql
CREATE TABLE VentaJuegoCliente
(
  Id_venta_cliente INT NOT NULL,
  Estado_venta VARCHAR(50) NOT NULL,
  Precio_venta DECIMAL(10, 2) NOT NULL,
  Fecha_venta DATE NOT NULL,
  PRIMARY KEY (Id_venta_cliente)
);


CREATE TABLE HistorialCompras
(
  Compras_realizadas INT NOT NULL,
  Id_historial INT NOT NULL,
  Id_cliente INT NOT NULL,
  Id_pedido INT NOT NULL,
  PRIMARY KEY (Id_historial, Id_cliente, Id_pedido),
  FOREIGN KEY (Id_cliente) REFERENCES Cliente(Id_cliente)
    ↪ ON DELETE CASCADE ON UPDATE CASCADE,
  FOREIGN KEY (Id_pedido) REFERENCES Pedido(Id_pedido) ON
    ↪ DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE Realiza_pedido
(
  Id_cliente INT NOT NULL,
  Id_pedido INT NOT NULL,
  PRIMARY KEY (Id_cliente, Id_pedido),
  FOREIGN KEY (Id_cliente) REFERENCES Cliente(Id_cliente)
    ↪ ON DELETE CASCADE ON UPDATE CASCADE,
  FOREIGN KEY (Id_pedido) REFERENCES Pedido(Id_pedido) ON
    ↪ DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE Pedido_Contiene
(
  Id_pedido INT NOT NULL,
  Id_juego INT NOT NULL,
  PRIMARY KEY (Id_pedido, Id_juego),
  FOREIGN KEY (Id_pedido) REFERENCES Pedido(Id_pedido) ON
    ↪ DELETE CASCADE ON UPDATE CASCADE,
  FOREIGN KEY (Id_juego) REFERENCES Juego(Id_juego) ON
    ↪ DELETE CASCADE ON UPDATE CASCADE
```

```
);

CREATE TABLE Hay_stock
(
  Id_inventario INT NOT NULL,
  Id_juego INT NOT NULL,
  PRIMARY KEY (Id_inventario, Id_juego),
  FOREIGN KEY (Id_inventario) REFERENCES Inventario(
    ↪ Id_inventario) ON DELETE CASCADE ON UPDATE CASCADE,
  FOREIGN KEY (Id_juego) REFERENCES Juego(Id_juego) ON
    ↪ DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE Cliente_Vende_juego
(
  Cantidad_Juegos_vendidos INT NOT NULL,
  Id_cliente INT NOT NULL,
  Id_venta_cliente_ INT NOT NULL,
  PRIMARY KEY (Id_cliente, Id_venta_cliente_),
  FOREIGN KEY (Id_cliente) REFERENCES Cliente(Id_cliente)
    ↪ ON DELETE CASCADE ON UPDATE CASCADE,
  FOREIGN KEY (Id_venta_cliente_) REFERENCES
    ↪ VentaJuegoCliente(Id_venta_cliente_) ON DELETE
    ↪ CASCADE ON UPDATE CASCADE
);

CREATE TABLE VentaContiene_juego
(
  Id_venta_cliente_ INT NOT NULL,
  Id_juego INT NOT NULL,
  PRIMARY KEY (Id_venta_cliente_, Id_juego),
  FOREIGN KEY (Id_venta_cliente_) REFERENCES
    ↪ VentaJuegoCliente(Id_venta_cliente_) ON DELETE
    ↪ CASCADE ON UPDATE CASCADE,
  FOREIGN KEY (Id_juego) REFERENCES Juego(Id_juego) ON
    ↪ DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE Detalles_Pedido
(
  Id_detalles INT NOT NULL,
  Cantidad INT NOT NULL,
  Precio_unitario DECIMAL(10, 2) NOT NULL,
```

```
136   Id_pedido INT NOT NULL,
137   Id_juego INT NOT NULL,
138   PRIMARY KEY (Id_detalles, 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 linea, ademas 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.

```sql
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.

```
ALTER TABLE Empleado
ADD CONSTRAINT chk_Horas_trabajadas_range
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.

```
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 García, 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

9. Evidencia de Restricciones para Tuplas

10. Consultas Relevantes

11. Vistas Relevantes