Memoria Técnica Práctica 3
Bases de Datos 2
Curso 2024/2025
Universidad de Zaragoza

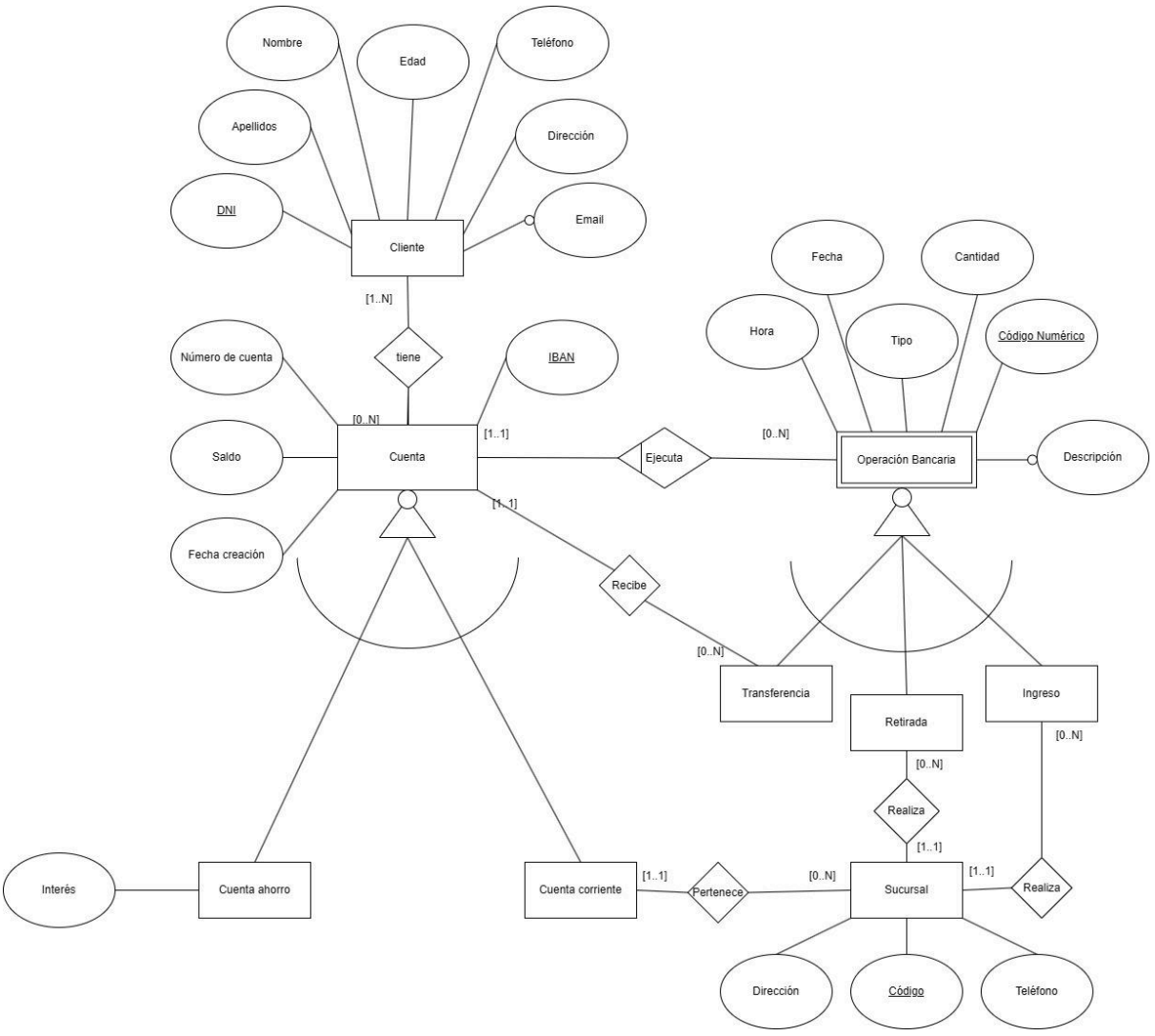
Diego Mateo Lorente - 873338
Daniel Simón Gayán - 870984
Carlos Solana Melero - 872815

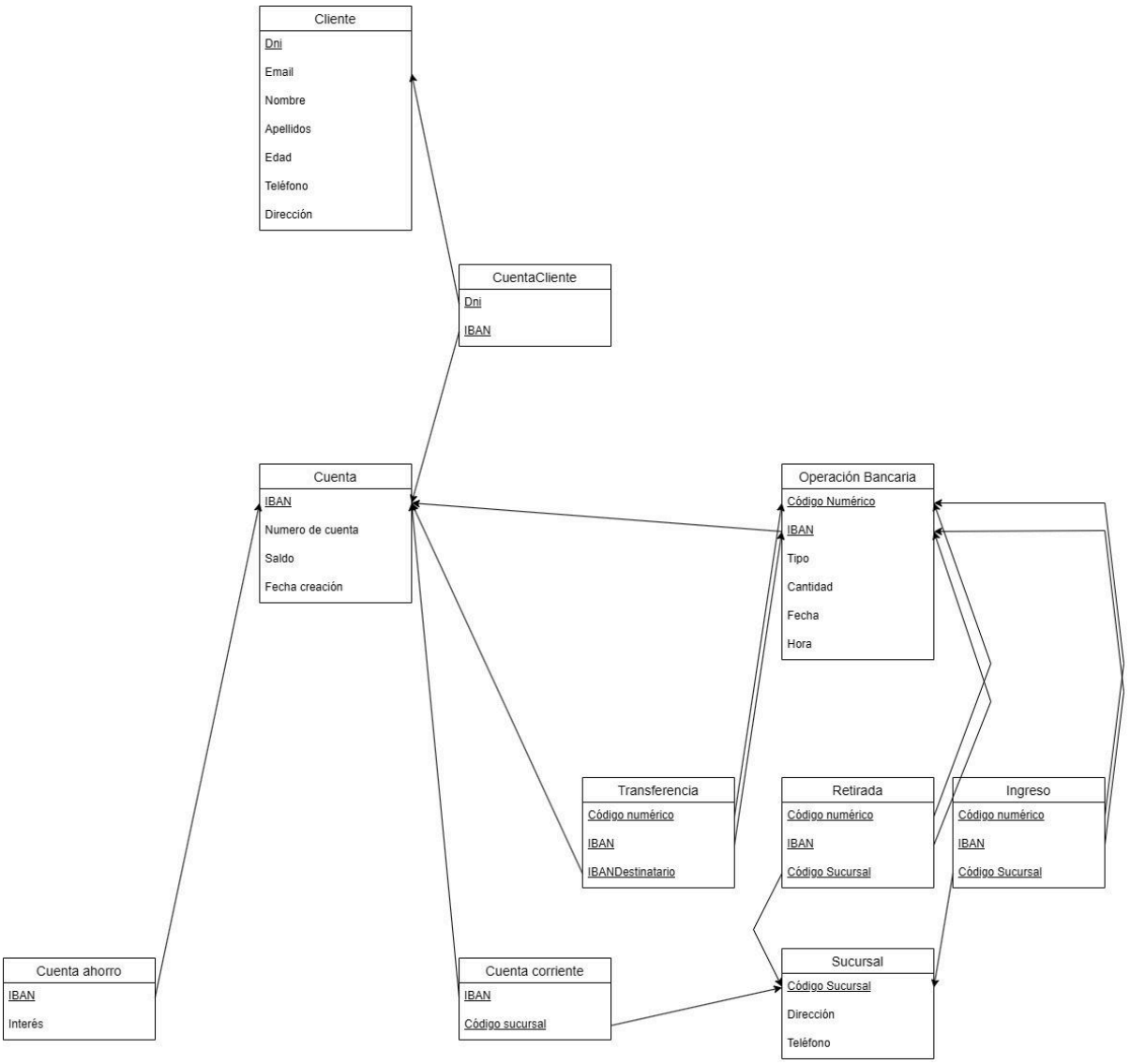
Esfuerzos invertidos

Horas invertidas	Diego	Carlos	Daniel	Total
Parte 1	4	18	6	28
Parte 2	15	0	8	23

Parte 1

Esquema conceptual y lógico de la base de datos relacional diseñada en la práctica anterior





Este es nuestro esquema lógico de la práctica anterior, hemos detectado varios fallos que se podrían haber evitado. Por ejemplo, edad no sería adecuada y habría que poner fecha de nacimiento, y también retirada y ingreso podrían unificarse en una sola tabla ya que comparten todos los campos. Estos fallos los hemos solucionado de cara al esquema global.

DOMINIOS

tpOpBancaria = {Transferencia, Retirada, Ingreso};
tpDate = Date;
tpSaldo= 0..
tpDni = 1-9*a-z

ESQUEMA DE RELACIÓN

Cliente = (
DNI: tpDni;
Email: cadena(50); **NO NULO**;
Nombre: tpName; **NO NULO**;
Apellidos: cadena(50); **NO NULO**;
Edad: number; **NO NULO**;
Teléfono: number; **NO NULO**;
Dirección: cadena(50); **NO NULO**;
);

Cuenta = (
IBAN: cadena(34); **NO NULO**;
Numero_Cuenta: cadena(20); **NO NULO**;
Saldo: tpSaldo; **NO NULO**;
Fecha_Creación: tpDate;
);

CuentaCliente = (
DNI: tpDni;
IBAN: cadena(20);
Dni clave ajena de cliente
Iban clave ajena de cuenta
);

Ahorro = (
IBAN: cadena(34); **NO NULO**;
Interés: number; **NO NULO**;
Iban es clave ajena de cuenta
);

Corriente = (
IBAN: cadena(34); **NO NULO**;
Código_Sucursal: number; **NO NULO**;
Iban es clave ajena de cuenta
);

Sucursal = (
Código_Sucursal: number; **NO NULO**;
Dirección: cadena(100); **NO NULO**;
Teléfono: cadena(20); **NO NULO**;
);

OperacionBancaria = (
Código_Numerico: number; **NO NULO**;
IBAN: cadena(34); **NO NULO**;
Cantidad: number;

Tipo: tpOpBancaria;
Fecha: tpDate;
Hora: timestamp;
Iban es clave ajena de cuenta
);
Retirada = (
Código_Numerico: number; **NO NULO**;
Código_Sucursal: number; **NO NULO**;
IBAN: cadena(34); **NO NULO**;
Iban y codigo numérico son clave ajena de operacion
Codigo sucursal es clave ajena de sucursal
);

Ingreso = (
Código_Numerico: number; **NO NULO**;
Código_Sucursal: number; **NO NULO**;
IBAN: cadena(34); **NO NULO**;
Iban y codigo numérico son clave ajena de operacion
Codigo sucursal es clave ajena de sucursal
);

Transferencia = (
Código_Numerico: number; **NO NULO**;
IBAN_Emisor: cadena(34); **NO NULO**;
IBAN_Receptor: cadena(34); **NO NULO**;
Iban Emisor y codigo numérico son clave ajena de operacion
Iban_Receptor es clave ajena de cuenta
);
Restricciones:
El iban tiene que tener un mínimo de 5 dígitos.

Las cuentas de ahorro deben contar obligatoriamente con un valor de interés, mientras que las cuentas corrientes deben tener asignado un código de sucursal.

Todas las operaciones deben ser de un tipo.

Todas las cuentas deben ser de un tipo.

El teléfono debe tener 9 dígitos.

Determinación del esquema lógico y conceptual de la base de datos Oracle a integrar

Para descubrir el esquema global lo se han ejecutado los siguientes comandos:

```
SELECT cons.constraint_name, cons.table_name, cols.column_name
FROM user_constraints@SCHEMA2BD2 cons
JOIN user_cons_columns@SCHEMA2BD2 cols
  ON cons.constraint_name = cols.constraint_name
WHERE cons.constraint_type = 'P'
ORDER BY cons.table_name, cons.constraint_name;
```

De esta manera se obtienen todas las tablas y sus claves primarias

```
SELECT cons.constraint_name, cons.table_name, cols.column_name,
rcons.table_name
FROM user_constraints@SCHEMA2BD2 cons
JOIN user_cons_columns@SCHEMA2BD2 cols
  ON cons.constraint_name = cols.constraint_name
JOIN user_constraints@SCHEMA2BD2 rcons
  ON cons.r_constraint_name = rcons.constraint_name
JOIN user_cons_columns@SCHEMA2BD2 rcols
  ON rcons.constraint_name = rcols.constraint_name
WHERE cons.constraint_type = 'R'
ORDER BY cons.table_name, cons.constraint_name;
```

De esta manera hemos descubierto las claves ajenas de cada tabla, de esta manera hemos descubierto partes extrañas, como por ejemplo que OPEFECTIVO, pese a tener un campo llamado SUCURSAL_CODOFICINA, que coincide con la clave primaria pero no está expresamente relacionado y marcado como foreign key.

```
SELECT cons.constraint_name, cons.table_name, cons.search_condition
FROM user_constraints@SCHEMA2BD2 cons
WHERE cons.constraint_type = 'C'
ORDER BY cons.table_name, cons.constraint_name;
```

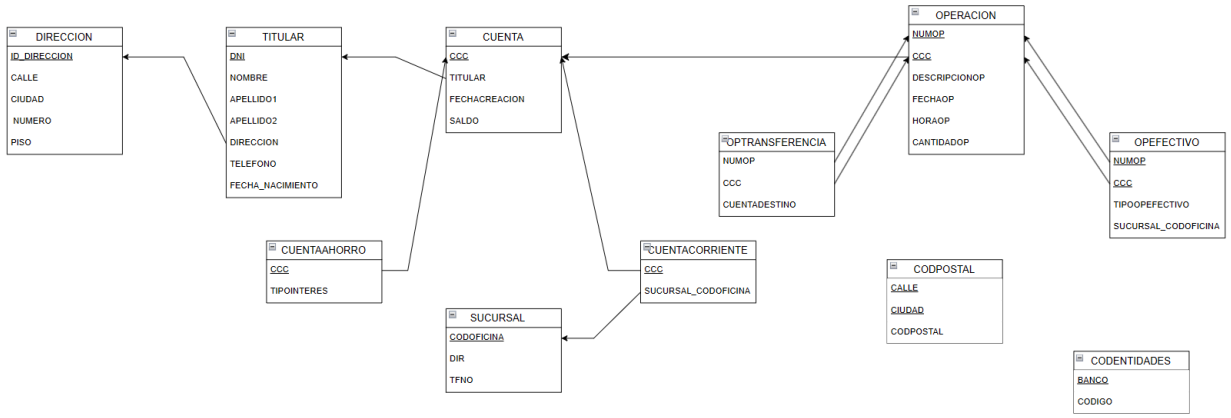
Por último esta consulta muestra las restricciones,que son únicamente NOT NULL en estos campos:

SQL> SELECT cons.constraint_name, cons.table_name, cons.search_condition
FROM user_constraints@SCHEMA2BD2 cons
WHERE cons.constraint_type = 'C'
ORDER BY cons.table_name, cons.constraint_name; 2 3 4

Constraint Name	Table Name	Search Condition
SYS_C009471	CODENTIDADES	"BANCO" IS NOT NULL
SYS_C009472	CODENTIDADES	"CODIGO" IS NOT NULL
SYS_C009459	CODPOSTAL	"CALLE" IS NOT NULL

SYS_C009460	CODPOSTAL	N/A
N/A	N/A	"CIUDAD" IS NOT NULL
SYS_C009461	CODPOSTAL	"CODPOSTAL" IS NOT NULL
SYS_C009474	N/A	N/A
N/A	CUENTA	"CCC" IS NOT NULL
SYS_C009475	CUENTA	"FECHACREACION" IS NOT NULL
SYS_C009476	CUENTA	"SALDO" IS NOT NULL
SYS_C009477	CUENTA	"TITULAR" IS NOT NULL
SYS_C009480	CUENTAAHORRO	"TIPOINTERES" IS NOT NULL
SYS_C009486	CUENTACORRIENTE	N/A
SYS_C009454	DIRECCION	"CALLE" IS NOT NULL
SYS_C009455	N/A	"SUCURSAL_CODOFICINA" IS NOT NULL
N/A	DIRECCION	"NUMERO" IS NOT NULL
SYS_C009456	DIRECCION	"PISO" IS NOT NULL
SYS_C009457	DIRECCION	"CIUDAD" IS NOT NULL
SYS_C009495	OPEFECTIVO	"TIPOOPEFECTIVO" IS NOT NULL
SYS_C009496	OPEFECTIVO	"SUCURSAL_CODOFICINA" IS NOT NULL
SYS_C009490	OPERACION	N/A
SYS_C009491	OPERACION	"HORAOP" IS NOT NULL
SYS_C009492	OPERACION	"FECHAOP" IS NOT NULL
N/A	OPERACION	"CANTIDADOP" IS NOT NULL
SYS_C009499	OPTRANSFERENCIA	"CUENTADESTINO" IS NOT NULL
SYS_C009483	SUCURSAL	"DIR" IS NOT NULL
SYS_C009484	SUCURSAL	"TFNO" IS NOT NULL
SYS_C009463	TITULAR	"DNI" IS NOT NULL
SYS_C009464	TITULAR	N/A
SYS_C009465	TITULAR	"APELLIDO1" IS NOT NULL
SYS_C009466	TITULAR	"NOMBRE" IS NOT NULL

N/A	TITULAR	"DIRECCION" IS NOT NULL
SYS_C009467	TITULAR	"TELEFONO" IS NOT NULL



Este el esquema obtenido a partir de las consultas , las principales diferencias respecto al nuestro son:

- La dirección se guarda en otra entidad llamada DIRECCION que es referenciada mediante DIRECCION en titular, que es un id.
- Existe opción para dos apellidos
- No hay entidad intermedia Cuenta Cliente por lo que no se permite que una cuenta tenga varios titulares, lo que es una debilidad comparado con nuestro modelo
- En operaciones la herencia se gestiona igual pero se han unificado las tablas de ingresos y retiradas en OPEFECTIVO con un tipo para distinguir.

Mejoras sugeridas para la base de datos a integrar

1. Normalización y tratamiento de direcciones

1. Unificar tablas relacionadas con la dirección

    - Actualmente se observa la tabla **`DIRECCION`** (ID_DIRECCION, CALLE, CIUDAD, NUMERO, PISO) y otra tabla **`CODPOSTAL`** (CALLE, CIUDAD, CODPOSTAL).

    - Hay cierta duplicidad o solapamiento de información: la ciudad aparece tanto en `DIRECCION` como en `CODPOSTAL`. Si el objetivo de `CODPOSTAL` es mantener un listado de códigos postales y asociarlos a calles/ciudades, haría falta una relación clara con `DIRECCION`.

    - **Posible mejora**:

        - Tener una tabla `CODPOSTAL` (CODPOSTAL, CIUDAD, PROVINCIA, etc.) y hacer que `DIRECCION` apunte a ella mediante una columna `CODPOSTAL` como FK en lugar de almacenar la ciudad en ambas tablas.

2. Separar correctamente dirección de titular

    - Se ve que en la tabla `TITULAR` hay un campo “DIRECCION” que parece apuntar al campo “ID_DIRECCION” de la tabla `DIRECCION`. Sería bueno que el nombre de la columna que actúa como FK (por ejemplo, “ID_DIRECCION”) sea el mismo tanto en la tabla `TITULAR` como en la tabla `DIRECCION` para mayor claridad.

---

2. Tabla “Cuenta” y subtipos de cuenta

- Como se requiere la posibilidad de varios titulares para una misma cuenta, habría que crear una tabla intermedia `CUENTA_TITULAR` para permitir esa relación N:M (p. ej. cuentas con cotitulares).

---

3 . Tabla “Sucursal” y su relación con “CuentaCorriente”

1. Tabla “CODENTIDADES”

    - Se ve que contiene `BANCO` y `CODIGO`. Si “CODIGO” corresponde a un código único de banco, podría relacionarse con “SUCURSAL” para indicar a qué banco pertenece la sucursal.

---

4. Tablas de “Operaciones”

1. Estructura de hora y fecha

- Posiblemente `FECHAOP` podría incluir fecha y hora en un solo campo de tipo `timestamp`.

---

5. Nomenclatura y consistencia

1. Nombres de columnas

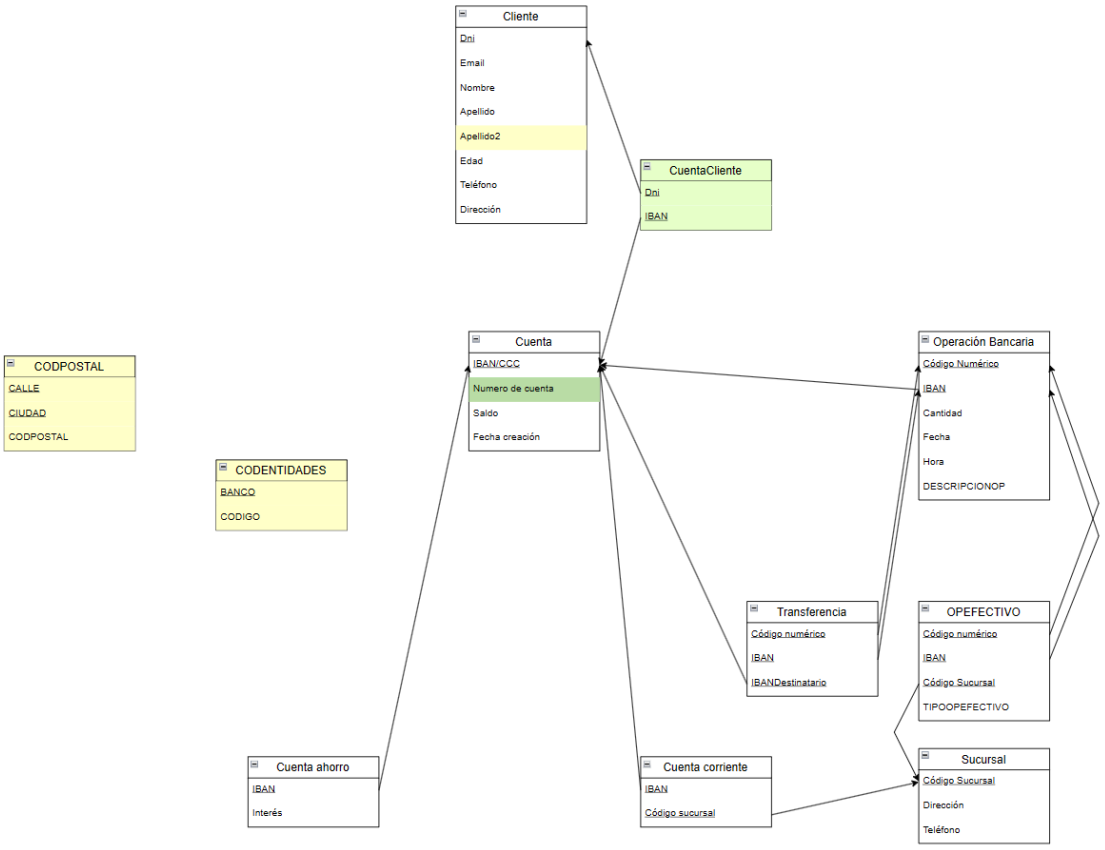
    - Evitar nombres ambiguos como `DIR` (en `SUCURSAL`). Se podría llamar `DIRECCION_SUCURSAL`.

---

6. Tabla OPEFECTIVO

1. Falta relación
    - En la tabla opefectivo hay un campo llamado SUCURSAL_CODOFICINA que guarda la FK de sucursal pero no hay ninguna relación que las conecte directamente por lo que se podría referenciar a sucursales que no existan.

Definición e implementación del esquema global en Oracle



En base a esto se ha creado este esquema global para juntar ambas BD. Los cambios principales son:

- En cliente se añade el apellido2
- Se elimina la tabla direcciones y se guarda en cliente la dirección completa
- Se mantiene cuentaCliente por lo que una cuenta puede tener varios propietarios
- En cuenta se añade el número de cuenta
- Se unifican las tablas de ingreso y retirada en una sola
- Opefectivo referencia a sucursal

En el esquema se han marcado de amarillo las partes que solo estaban en el esquema de Banquete y en verde aquellas solo presentes en Banquito.

Creación de las vistas del esquema global

Para crear las vistas se hace con este archivo.sql

```
-- Eliminar vistas si existen
BEGIN
   EXECUTE IMMEDIATE 'DROP VIEW Titular_Unificado';
EXCEPTION
   WHEN OTHERS THEN NULL;
END;
/

BEGIN
   EXECUTE IMMEDIATE 'DROP VIEW VistaCliente';
EXCEPTION
   WHEN OTHERS THEN NULL;
END;
/

BEGIN
   EXECUTE IMMEDIATE 'DROP VIEW VistaCuenta';
EXCEPTION
   WHEN OTHERS THEN NULL;
END;
/

BEGIN
   EXECUTE IMMEDIATE 'DROP VIEW VistaCuentaCliente';
EXCEPTION
   WHEN OTHERS THEN NULL;
END;
/

BEGIN
   EXECUTE IMMEDIATE 'DROP VIEW VistaCuentaAhorro';
EXCEPTION
   WHEN OTHERS THEN NULL;
END;
/

BEGIN
   EXECUTE IMMEDIATE 'DROP VIEW VistaCuentaCorriente';
EXCEPTION
```

```
   WHEN OTHERS THEN NULL;
END;
/

BEGIN
   EXECUTE IMMEDIATE 'DROP VIEW VistaSucursal';
EXCEPTION
   WHEN OTHERS THEN NULL;
END;
/

BEGIN
   EXECUTE IMMEDIATE 'DROP VIEW VistaOperacionBancaria';
EXCEPTION
   WHEN OTHERS THEN NULL;
END;
/

BEGIN
   EXECUTE IMMEDIATE 'DROP VIEW VistaOpefectivo';
EXCEPTION
   WHEN OTHERS THEN NULL;
END;
/

BEGIN
   EXECUTE IMMEDIATE 'DROP VIEW VistaTransferencia';
EXCEPTION
   WHEN OTHERS THEN NULL;
END;
/

BEGIN
   EXECUTE IMMEDIATE 'DROP VIEW CODPOSTAL';
EXCEPTION
   WHEN OTHERS THEN NULL;
END;
/

BEGIN
   EXECUTE IMMEDIATE 'DROP VIEW CODENTIDADES';
EXCEPTION
   WHEN OTHERS THEN NULL;
```

```
END;
/

-------------------------------------------------------------------------
-- Creaci◆n de las vistas
-------------------------------------------------------------------------

-- Vista para Titular_Unificado (accediendo v◆a dblink)
CREATE OR REPLACE VIEW Titular_Unificado AS
SELECT
    t.DNI,
    t.NOMBRE,
    t.APELLIDO1,
    t.APELLIDO2,
    d.CALLE || ', ' || d.CIUDAD || ', N◆ ' || d.NUMERO || ', Piso ' ||
d.PISO AS DIRECCION_COMPLETA,
    t.TELEFONO,
    t.FECHA_NACIMIENTO
FROM TITULAR@SCHEMA2BD2 t
JOIN DIRECCION@SCHEMA2BD2 d
    ON t.DIRECCION = d.ID_DIRECCION;

-- Vista unificada para la tabla CLIENTE
CREATE VIEW VistaCliente AS
SELECT
    COALESCE(TO_CHAR(b.dni), TO_CHAR(a.dni)) AS dni,
    COALESCE(b.nombre, a.nombre) AS nombre,
    COALESCE(CAST((2025 - b.edad) AS VARCHAR2(10)),
TO_CHAR(a.fecha_nacimiento, 'DD/MM/YYYY')) AS fecha_nacimiento,
    COALESCE(a.APELLIDO1 || ' ' || a.APELLIDO2, b.apellidos) AS
APELLIDOS,
    COALESCE(b.telefono, a.telefono) AS telefono,
    b.email AS email,
    COALESCE(b.direccion, a.DIRECCION_COMPLETA) AS direccion
FROM
    Titular_Unificado a
    FULL OUTER JOIN Cliente b ON TO_CHAR(a.dni) = TO_CHAR(b.dni);

-- Vista unificada para la tabla CUENTA
CREATE VIEW VistaCuenta AS
SELECT
    COALESCE(b.iban, a.CCC) AS iban,
    b.numero_cuenta,
```

```
    COALESCE(b.saldo, a.saldo) AS saldo,
    COALESCE(b.fecha_creacion, a.FECHACREACION) AS fecha_creacion
FROM
    Cuenta@SCHEMA2BD2 a
    FULL OUTER JOIN Cuenta b ON a.CCC = b.iban;

-- Vista unificada para la tabla CUENTACLIENTE
CREATE VIEW VistaCuentaCliente AS
SELECT CAST(a.dni AS VARCHAR2(20)) AS dni, CAST(a.iban AS VARCHAR2(50))
AS iban
FROM CuentaCliente a
UNION
SELECT CAST(b.TITULAR AS VARCHAR2(20)) AS dni, CAST(b.CCC AS
VARCHAR2(50)) AS iban
FROM Cuenta@SCHEMA2BD2 b;

-- Vista unificada para la tabla AHORRO
CREATE VIEW VistaCuentaAhorro AS
SELECT
    b.CCC AS iban,
    b.TIPOINTERES AS interes
FROM
    CUENTAAHORRO@SCHEMA2BD2 b
UNION
SELECT
    a.iban,
    a.interes
FROM
    Ahorro a;

-- Vista unificada para la tabla CORRIENTE
CREATE VIEW VistaCuentaCorriente AS
SELECT
    b.CCC  AS iban,
    b.SUCURSAL_CODOFICINA AS codigo_sucursal
FROM
    CUENTACORRIENTE@SCHEMA2BD2 b
UNION
SELECT
    a.iban AS iban,
    a.codigo_sucursal AS codigo_sucursal
FROM
    Corriente a;
```

```sql
-- Vista unificada para la tabla SUCURSAL
CREATE VIEW VistaSucursal AS
SELECT
    b.CODOFICINA,
    b.dir,
    NULL AS tfno
FROM
    Sucursal@SCHEMA2BD2 b
UNION
SELECT
    a.codigo_sucursal,
    a.direccion,
    a.telefono
FROM
    Sucursal a;

-- Vista unificada para la tabla OPERACIONBANCARIA
CREATE VIEW VistaOperacionBancaria AS
SELECT
    numop AS codigo_numerico,
    CCC AS iban,
    cantidadop AS cantidad,
    fechaop AS fecha,
    horaop AS hora,
    descripcionOP AS descripcionOP
FROM OPERACION@SCHEMA2BD2
UNION
SELECT
    codigo_numerico,
    iban,
    cantidad,
    fecha,
    TO_CHAR(hora, 'HH24:MI:SS') AS hora,
    NULL AS descripcionOP
FROM OperacionBancaria;

-- Vista unificada para la tabla OPefectivo (Retirada e Ingreso)
CREATE VIEW VistaOpefectivo AS
SELECT
    a.codigo_numerico AS codigo_numerico,
    a.iban AS iban,
    a.codigo_sucursal AS codigo_sucursal,
```

```
    'retirada' AS tipoopefectivo
FROM
    Retirada a
UNION
SELECT
    a.codigo_numerico AS codigo_numerico,
    a.iban AS iban,
    a.codigo_sucursal AS codigo_sucursal,
    'ingreso' AS tipoopefectivo
FROM
    Ingreso a
UNION
SELECT
    b.NUMOP AS codigo_numerico,
    b.CCC AS iban,
    b.SUCURSAL_CODOFICINA AS codigo_sucursal,
    b.TIPOOPEFECTIVO AS tipoopefectivo
FROM
    OPEFECTIVO@SCHEMA2BD2 b;

-- Vista unificada para la tabla TRANSFERENCIA
CREATE VIEW VistaTransferencia AS
SELECT
    b.NUMOP AS codigo_numerico,
    b.CCC AS iban_emisor,
    b.CUENTADESTINO AS iban_receptor
FROM
    OPTRANSFERENCIA@SCHEMA2BD2 b
UNION
SELECT
    a.codigo_numerico AS codigo_numerico,
    a.iban_emisor AS iban_emisor,
    a.iban_receptor AS iban_receptor
FROM
    Transferencia a;

-- Vistas para CODPOSTAL y CODENTIDADES (acceso remoto)
CREATE VIEW CODPOSTAL AS
SELECT * FROM CODPOSTAL@SCHEMA2BD2;

CREATE VIEW CODENTIDADES AS
SELECT * FROM CODENTIDADES@SCHEMA2BD2;
```

```sql
-------------------------------------------------------------------------
-- Sentencias SELECT para verificar el contenido de cada vista
-------------------------------------------------------------------------

-- Verificar Titular_Unificado
SELECT * FROM Titular_Unificado;

-- Verificar VistaCliente
SELECT * FROM VistaCliente;

-- Verificar VistaCuenta
SELECT * FROM VistaCuenta;

-- Verificar VistaCuentaCliente
SELECT * FROM VistaCuentaCliente;

-- Verificar VistaCuentaAhorro
SELECT * FROM VistaCuentaAhorro;

-- Verificar VistaCuentaCorriente
SELECT * FROM VistaCuentaCorriente;

-- Verificar VistaSucursal
SELECT * FROM VistaSucursal;

-- Verificar VistaOperacionBancaria
SELECT * FROM VistaOperacionBancaria;

-- Verificar VistaOpefectivo
SELECT * FROM VistaOpefectivo;

-- Verificar VistaTransferencia
SELECT * FROM VistaTransferencia;

-- Verificar CODPOSTAL
SELECT * FROM CODPOSTAL;

-- Verificar CODENTIDADES
SELECT * FROM CODENTIDADES;
```

Explicación de las funciones usadas para cada vista:

1. VistaTitular_Unificado

Se utiliza la concatenación de cadenas para formar el campo `DIRECCION_COMPLETA`:

```
d.CALLE || ', ' || d.CIUDAD || ', N° ' || d.NUMERO || ', Piso ' ||
d.PISO AS DIRECCION_COMPLETA
```

- Esto junta varios campos (calle, ciudad, número y piso) separados por comas y etiquetas, permitiendo juntar todos los campos de la tabla de direcciones de banquete para cambiar el ID_DIRECCION por la dirección completa, tal y como está en banquito. Esta solución, aunque menos normalizada desde el punto de vista de las formas normales, es más sencilla que insertar los datos de dirección en la tabla de direcciones, ya que podría haber direcciones repetidas y los campos de dirección en banquito podrían ser mucho más heterogéneos y no adecuarse bien al formato de la tabla de direcciones de banquete.

---

2. VistaCliente

Esta vista une la información del titular unificado, que es la de banquito con la dirección ya transformada con la tabla local `Cliente`. Se usan varias funciones para normalizar y formatear los datos:

- COALESCE:
  Se emplea para seleccionar el primer valor no nulo de dos posibles columnas. Ejemplos:

Para el `dni`:

```
COALESCE(TO_CHAR(b.dni), TO_CHAR(a.dni)) AS dni
```

  - Se convierte a cadena (con `TO_CHAR`) y se elige el valor disponible, ya sea el de `Cliente` (`b.dni`) o el de `Titular_Unificado` (`a.dni`).

  - Para `nombre`, `telefono` y `direccion` se sigue el mismo patrón, permitiendo que si la información de una fuente es nula, se tome la otra.

Conversión de datos y formateo de fechas:
Se usa `TO_CHAR` para formatear la fecha de nacimiento de `Titular_Unificado`:

```
TO_CHAR(a.fecha_nacimiento, 'DD/MM/YYYY')
```

- Esto transforma una fecha en una cadena con el formato día/mes/año.

CAST y operaciones aritméticas:
Para los registros de `Cliente` que traen un campo `edad` en lugar de una fecha de nacimiento, se estima la fecha de nacimiento:

```
CAST((2025 - b.edad) AS VARCHAR2(10))
```

- Aquí se resta el valor de `edad` del año 2025 (año de referencia) y luego se convierte el resultado a una cadena, permitiendo homogeneizar nuestro formato con edad, con banquete.

Concatenación de apellidos:
Se unen dos columnas (`APELLIDO1` y `APELLIDO2`) para formar el campo `APELLIDOS`:

```
COALESCE(a.APELLIDO1 || ' ' || a.APELLIDO2, b.apellidos) AS APELLIDOS
```

- Esto permite que, si en el registro unificado no se tienen ambos apellidos por separado, se recurra al campo único `apellidos` de la tabla `Cliente`.

Resumen en VistaCliente:
La combinación de estas funciones garantiza que, sin importar la fuente (remota o local), se obtenga un conjunto coherente de datos, con formatos uniformes y sin valores nulos donde se espera información.

---

3. VistaCuenta

En esta vista se unifican datos de cuentas provenientes de dos fuentes, y se utilizan funciones para seleccionar correctamente los datos:

- COALESCE para selección de campos:

Para el campo `iban` se usa:

```
COALESCE(b.iban, a.CCC) AS iban
```

  - Esto significa que, si en la tabla local (b) existe el campo `iban`, se utiliza ese valor; de lo contrario, se toma el valor CCC de la base remota (a).

  - Se emplea COALESCE de forma similar para `saldo` y `fecha_creacion`.

- Nota:
  No se aplican conversiones o formateos especiales en este caso, ya que los datos numéricos y de fecha son compatibles entre ambas fuentes.

---

4. VistaCuentaCliente

Esta vista consolida la relación entre clientes y cuentas usando:

CAST:
Se convierte explícitamente el `dni` y el `iban` a cadenas (VARCHAR2) para asegurar que ambas fuentes tengan el mismo tipo de dato:

```
CAST(a.dni AS VARCHAR2(20)) AS dni, CAST(a.iban AS VARCHAR2(50)) AS iban
```

De igual forma para la segunda parte de la unión:

```
CAST(b.TITULAR AS VARCHAR2(20)) AS dni, CAST(b.CCC AS VARCHAR2(50)) AS iban
```

- Esto permite que la unión (`UNION`) se realice sin problemas de incompatibilidad de tipos.

---

5. VistaCuentaAhorro y VistaCuentaCorriente

Estas vistas usan funciones de selección mediante UNION:

VistaCuentaAhorro:
Combina el `CCC` y `TIPOINTERES` de la fuente remota con los campos equivalentes (`iban` y `interes`) de la tabla local. No se aplican transformaciones adicionales, solo se renombran los campos para tener homogeneidad.

VistaCuentaCorriente:
De forma similar, se utiliza:

```
b.CCC  AS iban,
b.SUCURSAL_CODOFICINA AS codigo_sucursal
```

para banquete y se equipara con:

```
a.iban AS iban,
```

```
a.codigo_sucursal AS codigo_sucursal
```

en banquito. La unión de ambos conjuntos se realiza sin transformaciones extra, ya que los campos ya se encuentran en el formato deseado.

---

6. VistaSucursal

Aquí se combinan datos de sucursales de dos fuentes:

Selección directa y asignación de valores nulos:
En la parte remota se selecciona:

```
b.CODOFICINA,
b.dir,
NULL AS tfno
```

- indicando que en esa fuente no se tiene un campo para teléfono, y se iguala a `NULL` para mantener la estructura. En la tabla local se extraen `codigo_sucursal`, `direccion` y `telefono` directamente.

No se utilizan funciones de transformación avanzadas aquí, salvo la asignación explícita de `NULL` para homogeneizar la estructura de ambas partes.

---

7. VistaOperacionBancaria

Esta vista unifica operaciones bancarias usando funciones para formatear y homogeneizar la información:

TO_CHAR para formateo de hora:
En la parte local se transforma la hora a un formato de 24 horas:
`TO_CHAR(hora, 'HH24:MI:SS') AS hora`

- Esto asegura que, sin importar el tipo de dato de origen (que es de TIMESTAMP), la hora se presente de forma consistente como una cadena con el formato deseado.

- COALESCE en general:
  Aunque en esta vista no se utiliza COALESCE de forma directa entre columnas, se hace una unión (`UNION`) de registros de dos fuentes, garantizando que ambas partes tengan los mismos nombres de columnas para poder combinarse sin conflicto.

---

8. VistaOpefectivo

Esta vista consolida operaciones en efectivo de tres fuentes:

Asignación directa de literales:
En las dos primeras partes se usa:

```
'retirada' AS tipoopefectivo
y

'ingreso' AS tipoopefectivo
```

De esta manera se juntan todos los datos de la tabla de retirada de banquito con todos los datos de la tabla de ingreso de banquito y se asignan los tipos correspondientes.

- Uso de UNION:
  Se combinan las filas de las tablas `Retirada`, `Ingreso` y la tabla remota `OPEFECTIVO`, asegurándose que la estructura resultante tenga los mismos campos:

  - `codigo_numerico`, `iban`, `codigo_sucursal` y `tipoopefectivo`.
    La función principal aquí es la asignación de valores constantes (literal strings) para asignar los tipos.

---

9. VistaTransferencia

En la vista de transferencias se hace una unión de dos fuentes:

- Unión sin transformación:
  La selección de campos es directa, sin funciones de conversión. Sin embargo, se debe destacar que los nombres de columnas son diferentes por lo que simplemente hay que cambiar el nombre con `AS`, lo que permite que el `UNION` combine los datos sin conflictos.

---

10. Vistas CODPOSTAL y CODENTIDADES

Estas vistas se crean de forma directa usando el asterisco (`*`) para seleccionar todos los campos de las tablas remotas:

- Sin transformaciones:
  Como estas tablas sólo están presentes en banquete se añaden sin transformar.

Parte 2

Enunciado de un problema de diseño de bases de datos

Una empresa internacional gestiona una plataforma digital dedicada a la distribución, análisis y seguimiento del uso de videojuegos en múltiples consolas. Esta plataforma centraliza información sobre compañías del sector, videojuegos disponibles, consolas comercializadas, usuarios registrados, tipos de suscripción y la relación entre los usuarios y los videojuegos que poseen.

El sistema debe ser capaz de gestionar compañías que pueden actuar como desarrolladoras de videojuegos, fabricantes de consolas, o ambas. De cada compañía se debe almacenar su nombre, director general, país de origen, fecha de fundación y una licencia asociada a su rol. Los videojuegos, por su parte, deben estar identificados por un nombre único, precio, fecha de lanzamiento, puntuación media y uno o varios géneros. Cada videojuego es desarrollado por una única compañía y se lanza para una consola concreta.

Las consolas, también conocidas como plataformas, están asociadas a una empresa fabricante y disponen de información como nombre, fecha de lanzamiento, volumen de ventas y generación tecnológica a la que pertenecen (por ejemplo, octava o novena generación).

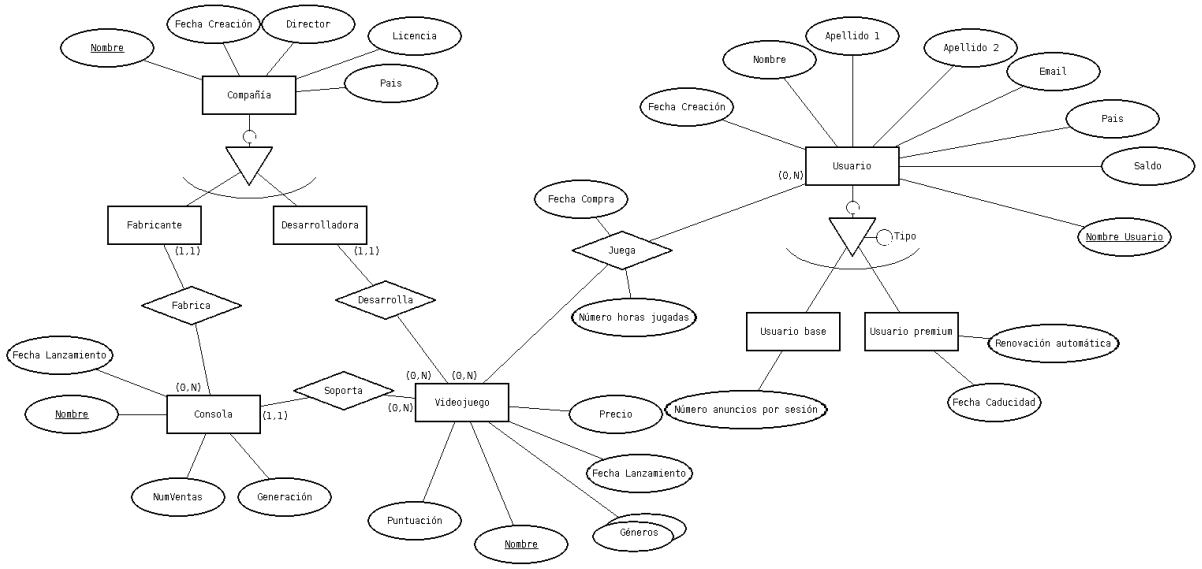
Los usuarios de la plataforma pueden registrarse y disponer de una cuenta que puede ser estándar o premium. Toda cuenta debe contener información como nombre de usuario, nombre completo, país, correo electrónico, saldo y fecha de creación. En el caso de los usuarios premium, se debe guardar además la fecha de expiración de la membresía y si está activada la renovación automática. Los usuarios base (no premium) no tienen el privilegio de no visualizar anuncios, por lo que se guarda la cantidad de anuncios por sesión que estos deben consumir antes de poder comenzar a jugar.

Por último, la plataforma registra qué videojuegos posee cada usuario, junto con la fecha de compra y el número total de horas jugadas.

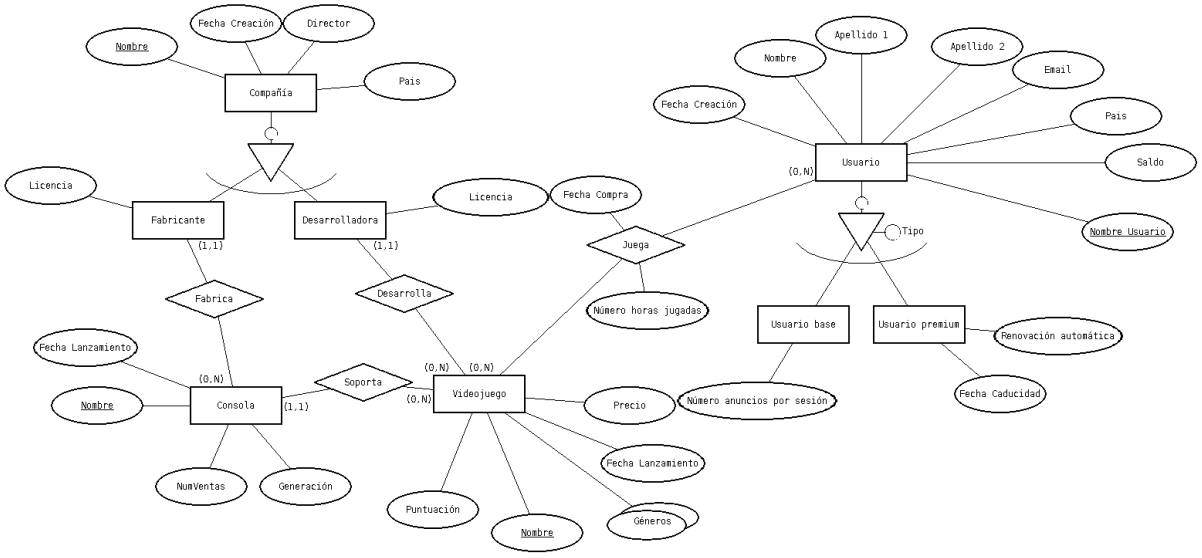
Esquema conceptual para el problema enunciado

Dado el enunciado del apartado anterior, por razones de mera comprensión lectora, hay un aspecto el cual ha provocado cierto debate en el grupo. Ese aspecto en concreto es la gestión de la licencia de la compañía. Por un lado, un compañero entendía que la licencia asociada al rol suponía crear un atributo “Licencia” en cada una de las entidades que heredan de “Compañía”, mientras que otro pensaba que la mejor opción era tener el atributo de la licencia en la entidad padre. Por ello, se han desarrollado 2 versiones distintas del diagrama E/R modificando únicamente ese aspecto. Los diagramas resultantes quedan de esta manera:

Primera versión:



Segunda versión:



La entidad **Compañía** almacena información básica como su nombre (clave primaria), director, país y fecha de creación. Esta entidad se especializa en dos subtipos: **Fabricante** y **Desarrolladora**, cada uno con un atributo específico de licencia (versión 2). Se ha modelado esta especialización como una generalización con disyunción total (una compañía puede ser fabricante, desarrolladora o ambas).

La entidad **Fabricante** está relacionada con la entidad **Consola** mediante la relación **Fabrica**, en la que se establece que una consola solo puede ser fabricada por una única compañía, pero una compañía puede fabricar varias consolas (relación 1:N). Las consolas contienen información sobre su nombre, fecha de lanzamiento, volumen de ventas y generación a la que pertenecen.

La entidad **Desarrolladora** se relaciona con la entidad **Videojuego** mediante la relación **Desarrolla**, también de tipo 1:N, ya que un videojuego es desarrollado por una única desarrolladora, mientras que una desarrolladora puede desarrollar múltiples videojuegos.

Por su parte, cada videojuego (entidad **Videojuego**) tiene atributos como nombre, precio, fecha de lanzamiento, puntuación y uno o varios géneros (atributo multivaluado). Los videojuegos también están relacionados con las consolas a través de la relación **Soporta**, indicando en qué consola se ejecuta cada juego. Un juego sólo está disponible para una única consola, y una consola puede soportar múltiples juegos (relación 0,N a 1,1).

La entidad **Usuario** contiene todos los datos personales del usuario, como su nombre de usuario (clave primaria), nombre, apellidos, email, país, fecha de creación y saldo. Esta entidad se especializa en dos subtipos (identificados mediante el atributo Tipo de la generalización): **Usuario base** y **Usuario premium**, de forma disjunta y total, de forma que solo puede pertenecer a uno de los 2 subtipos. Los usuarios base tienen un atributo adicional que representa el número de anuncios mostrados por sesión, mientras que los usuarios premium cuentan con atributos específicos como fecha de caducidad de la suscripción y si tienen activada la renovación automática.

Por último, se ha modelado la relación **Juega** entre las entidades Usuario y Videojuego para representar la posesión o actividad de juego. Esta relación incluye atributos como la fecha de compra y el número total de horas jugadas. La multiplicidad de esta relación permite reflejar que un usuario puede jugar a varios videojuegos y que cada videojuego puede ser jugado por múltiples usuarios.

Esquemas Lógicos para el problema enunciado

Durante el desarrollo del sistema se han diseñado **dos versiones del esquema relacional** que representan la misma realidad, pero aplicando distintas estrategias de modelado. Ambas versiones son compatibles a nivel semántico y están pensadas para facilitar una futura integración federada.

Esquema lógico 1 para el problema enunciado

A la hora de diseñar la primera versión esquema relacional que da soporte al sistema de gestión de videojuegos, se han tomado varias decisiones con el objetivo de mantener la coherencia, normalización y claridad del modelo. A continuación, se resumen los principales criterios adoptados durante el proceso de modelado:

- **Idioma castellano**: Se han usado nombres en castellano para atributos y dominios, facilitando la comprensión del modelo en un entorno académico hispanohablante.
- **Jerarquía explícita**: Se han mantenido las entidades generales (**Compañía**, **Usuario**) junto con sus subentidades para reflejar mejor los distintos roles y tipos de usuarios o empresas.
- **Tablas separadas para subtipos**: Se ha optado por dividir **Usuario** en **UsuarioPremium** y **UsuarioCorriente**, lo que permite controlar fácilmente qué atributos pertenecen a cada tipo.
- **Atributo multivaluado separado**: El atributo “Géneros” se ha movido a una tabla **GeneroVideojuego** para cumplir la 1ª Forma Normal y permitir múltiples géneros por juego.
- **Dominios definidos**: Se han definido tipos personalizados (como **tpNombre**, **tpEmail**, **tpFecha**...) para unificar criterios de formato y facilitar cambios futuros.

- **Restricciones de no nulidad**: Atributos esenciales como nombres, fechas o precios no permiten nulos para garantizar la consistencia de los datos.
- **Claves primarias claras**: Se han usado claves simples o compuestas según el caso, como en **PosesionVideojuego**, para reflejar relaciones sin ambigüedad.

DOMINIOS

tpEsPremium = Boolean;
tpNombre = cadena(100);
tpEmail = cadena (100);
tpApellido = cadena(50);
tpFecha = Date;
tpSaldo = Number;
tpPrecio = Number;
tpPais = cadena(100);
tpNumAnuncios: 0..10;
tpGenero: cadena(50)
tpRenovacion = Boolean;
tpNumHoras = Number;
tpNumVentas = Number;
tpGeneracion = cadena(50);
tpLicencia = cadena(500);
tpPuntuacion = 0..10;

ESQUEMA DE RELACIÓN

Compañia = (
Nombre: tpNombre;
Director: tpNombre **NO NULO;**
FechaCreacion: tpFecha **NO NULO;**
Pais: tpPais **NO NULO;**
);

Desarrolladora = (
Nombre: tpNombre;
LicenciaCopyright: tpLicencia **NO NULO;**
Nombre clave ajena de Compañia;
);

Fabricante = (
Nombre: tpNombre;
LicenciaFabricacion: tpLicencia **NO NULO;**
Nombre clave ajena de Compañia;
);
Consola = (
Nombre: tpNombre;
FechaLanzamiento: tpFecha **NO NULO;**
NumVentas: tpNumVentas **NO NULO;**
Fabricante: tpNombre **NO NULO;**
Generacion: tpGeneracion **NO NULO;**

Fabricante clave ajena de Fabricante;

);

Videojuego = (
Nombre: tpNombre;
Desarrolladora: tpNombre **NO NULO;**
Consola: tpNombre **NO NULO;**
Precio: tpPrecio **NO NULO;**
FechaLanzamiento: tpFecha **NO NULO;**
Puntuacion: tpPuntuacion **NO NULO;**
Desarrolladora clave ajena de Desarrolladora;
Consola clave ajena de Consola;
);

GeneroVideojuego = (
Videojuego: tpNombre;
Genero: tpGenero;
Videojuego clave ajena de Videojuego;
);

Usuario = (
NombreUsuario: tpNombre;
FechaCreacion: tpFecha **NO NULO;**
Nombre: tpNombre **NO NULO;**
Apellido1: tpApellido **NO NULO;**
Apellido2: tpApellido **NO NULO;**
Email: tpEmail **NO NULO;**
Pais: tpPais **NO NULO;**
Saldo: tpSaldo **NO NULO;**
EsPremium: tpEsPremium **NO NULO;**
);

UsuarioPremium = (
NombreUsuario: tpNombre;
FechaCaducidad: tpFecha **NO NULO;**
RenovacionAutomatica: **tpRenovacion NO NULO;**
NombreUsuario clave ajena de Usuario;
);

UsuarioCorriente = (
NombreUsuario: tpNombre;
NumeroAnuncios: tpNumAnuncios **NO NULO;**
NombreUsuario clave ajena de Usuario;
);

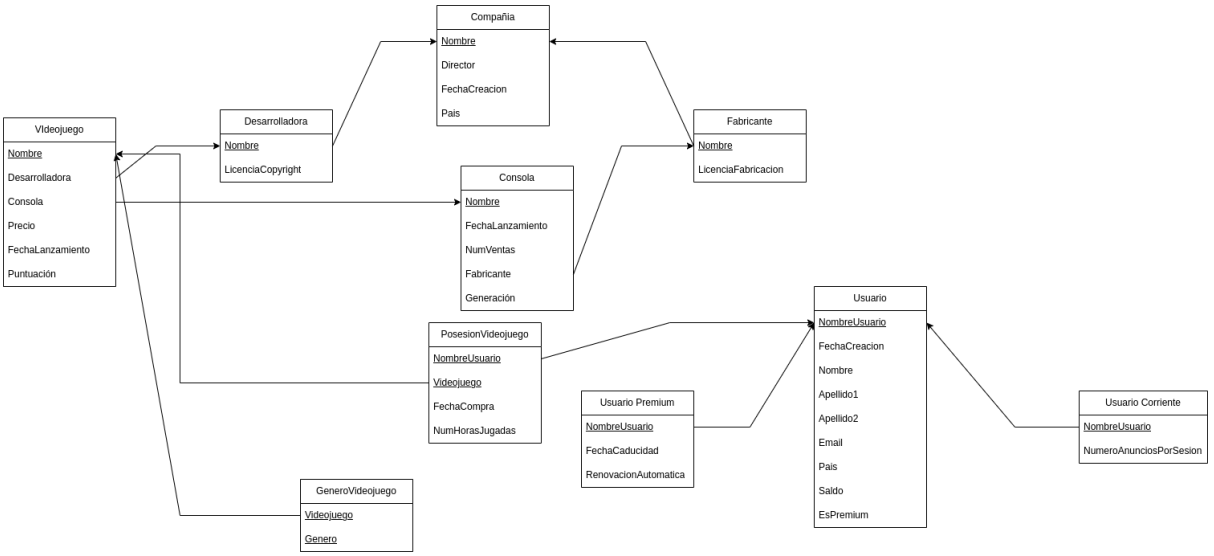
PosesionVideojuego = (
NombreUsuario: tpNombre;

Videojuego: tpNombre **NO NULO;**
FechaCompra; tpFecha **NO NULO;**
NumHorasJugadas; tpNumHoras **NO NULO;**
NombreUsuario clave ajena de Usuario;
);

RESTRICCIONES

1. La puntuación de un videojuego debe estar en el rango [0,10]
2. El saldo del usuario no debe ser negativo, al igual que el precio de una consola o videojuego.
3. El tipo de compañía debe ser **'Desarrolladora'** o **'Fabricante'**.
4. El tipo de suscripción debe ser **'premium'** o **'base'**.
5. Las cuentas premium deben tener 0 anuncios por sesión.
6. Las cuentas base no deben tener fecha de caducidad ni renovación automática.
7. Si se borra un usuario o videojuego, también se eliminan sus registros de propiedad.
8. Una desarrolladora o fabricante debe existir previamente como compañía.

El diagrama con las tablas queda así:



Esquema lógico 2 para el problema enunciado

En esta segunda versión del modelo lógico se ha optado por una estructura más simple y compacta, en la que se han unificado varias entidades y atributos que en la primera versión estaban separados. El objetivo principal ha sido reducir el número de tablas y simplificar el diseño, manteniendo la información esencial del sistema. A continuación se describen las principales decisiones tomadas en esta versión respecto a la anterior:

- **Uso de una única tabla Company**: Se ha reunido la información de desarrolladoras y fabricantes en una sola tabla, usando un campo **Type** para distinguir su función.
- **Usuarios en una sola tabla Account**: Todos los usuarios se gestionan en una única tabla, diferenciando los tipos de cuenta con el atributo **MembershipType**.

- **Atributos en lugar de subentidades**: En lugar de crear tablas separadas para los tipos de usuario, se han incluido todos los campos en la misma tabla y se controlan con restricciones.
- **Géneros como texto en una cadena**: Los géneros de los videojuegos no se han separado en otra tabla, sino que se guardan como texto en un solo campo.
- **Nombres en inglés**: Se ha utilizado el inglés para nombrar tablas y atributos, buscando una estructura más estándar o internacional.
- **Menor número de tablas y relaciones**: Al juntar más información en menos tablas, el modelo final es más compacto y directo.

DOMINIOS

tpMembershipType = {premium, base};
tpName = cadena(100);
tpEmail = cadena (100);
tpDate = Date;
tpWallet = Number;
tpPrice = Number;
tpCountry = cadena(100);
tpNumAds: 0..10;
tpGenres: cadena(250)
tpRenew = Boolean;
tpNumHours = Number;
tpSales = Number;
tpGeneration = cadena(50);
tpLicense = cadena(500);
tpRating = 0..10;
tpGenres = cadena(250)
tpType = {developer, manufacturer};

ESQUEMA DE RELACIÓN

Company = (
Name: tpName;
Director: tpName **NO NULO;**
CreatedAt: tpDate **NO NULO;**
Country: tpCountry **NO NULO;**
License: tpLicense **NO NULO;**
Type: tpType **NO NULO;**
);

Platform = (
Name: tpName;
ReleaseDate: tpDate **NO NULO;**
SalesVolume: tpSales **NO NULO;**
Manufacturer: tpName **NO NULO;**
Generation: tpGeneration **NO NULO;**
Manufacturer clave ajena de Company;

);

Videogame = (
Name: tpName;
Developer: tpName **NO NULO;**
Console: tpName **NO NULO;**
Price: tpPrice **NO NULO;**
ReleaseDate: tpDate **NO NULO;**
Rating: tpRating **NO NULO;**
Genres: tpGenres **NO NULO;**
Developer clave ajena de Company;
Console clave ajena de Console;
);

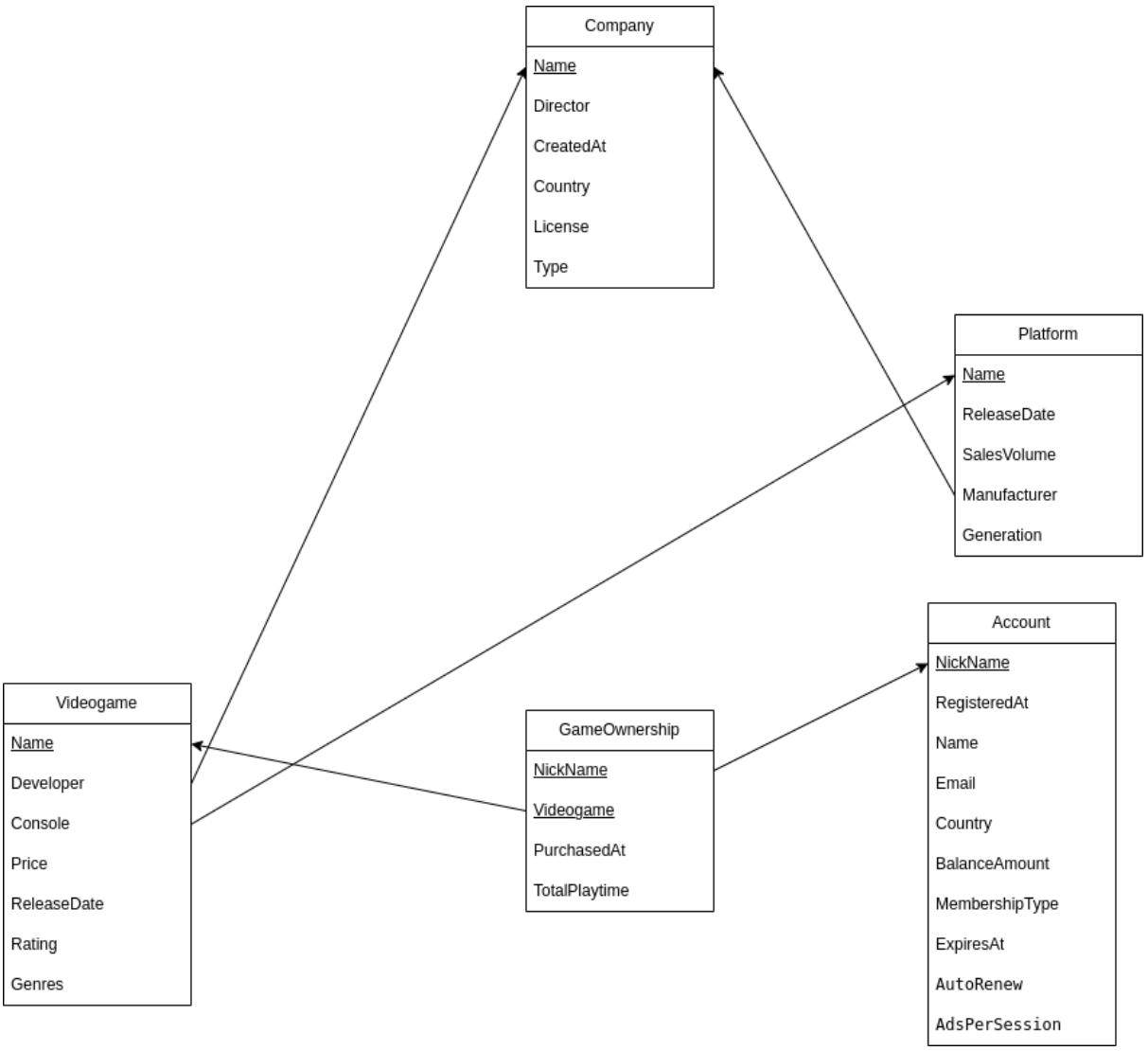
Account = (
NickName: tpName;
RegisteredAt: tpDate **NO NULO;**
Name: tpName **NO NULO;**
Email: tpEmail **NO NULO;**
Country: tpCountry **NO NULO;**
BalanceAmount: tpWallet **NO NULO;**
MembershipType: tpMembershipType **NO NULO;**
ExpiresAt; tpDate **NO NULO;**
AutoRenew; **tpRenew NO NULO;**
AdsPerSession; tpNumAds **NO NULO;**
);

GameOwnership = (
NickName: tpName;
Videogame: tpName **NO NULO;**
PurchasedAt; tpDate **NO NULO;**
TotalPlaytime; tpNumHours **NO NULO;**
NickName clave ajena de Account;
Videogame clave ajena de Videogame;
);

RESTRICCIONES

1. La puntuación de un videojuego debe estar en el rango [0,10]
2. El saldo del usuario no debe ser negativo, al igual que el precio de una consola o videojuego.
3. El tipo de compañía debe ser **'Desarrolladora'** o **'Fabricante'**.
4. El tipo de suscripción debe ser **'premium'** o **'base'**.
5. Las cuentas premium deben tener 0 anuncios por sesión.
6. Las cuentas base no deben tener fecha de caducidad ni renovación automática.
7. Si se borra un usuario o videojuego, también se eliminan sus registros de propiedad.

El diagrama con las tablas queda así:



Esquema global y su implementación en PostgreSQL

Una vez definidos los diagramas relacionales de ambas versiones, lo que toca ahora integrar son las tablas en SQL. El lenguaje donde se va a realizar es PostgreSQL. Así, la creación de tablas de la primera base de datos es la siguiente:

```sql
-- Crear esquema si no existe
CREATE SCHEMA IF NOT EXISTS esquema_videojuegos;

-- Borrar tablas existentes en orden de dependencias
DROP TABLE IF EXISTS esquema_videojuegos.PosesionVideojuego CASCADE;
DROP TABLE IF EXISTS esquema_videojuegos.UsuarioCorriente CASCADE;
DROP TABLE IF EXISTS esquema_videojuegos.UsuarioPremium CASCADE;
DROP TABLE IF EXISTS esquema_videojuegos.Usuario CASCADE;
DROP TABLE IF EXISTS esquema_videojuegos.GeneroVideojuego CASCADE;
DROP TABLE IF EXISTS esquema_videojuegos.Videojuego CASCADE;
DROP TABLE IF EXISTS esquema_videojuegos.Consola CASCADE;
```

```sql
DROP TABLE IF EXISTS esquema_videojuegos.Fabricante CASCADE;
DROP TABLE IF EXISTS esquema_videojuegos.Desarrolladora CASCADE;
DROP TABLE IF EXISTS esquema_videojuegos.Compania CASCADE;

-- Crear tablas

-- Compañía
CREATE TABLE esquema_videojuegos.Compania (
   Nombre VARCHAR(100) PRIMARY KEY,
   Director VARCHAR(100) NOT NULL,
   FechaCreacion DATE NOT NULL,
   Pais VARCHAR(100) NOT NULL
);

-- Desarrolladora
CREATE TABLE esquema_videojuegos.Desarrolladora (
   Nombre VARCHAR(100) PRIMARY KEY,
   LicenciaCopyright VARCHAR(500) NOT NULL,
   FOREIGN KEY(Nombre) REFERENCES esquema_videojuegos.Compania(Nombre)
);

-- Fabricante
CREATE TABLE esquema_videojuegos.Fabricante (
   Nombre VARCHAR(100) PRIMARY KEY,
   LicenciaFabricacion VARCHAR(500) NOT NULL,
   FOREIGN KEY(Nombre) REFERENCES esquema_videojuegos.Compania(Nombre)
);

-- Consola
CREATE TABLE esquema_videojuegos.Consola (
   Nombre VARCHAR(100) PRIMARY KEY,
   FechaLanzamiento DATE NOT NULL,
   NumVentas INTEGER NOT NULL,
   Fabricante VARCHAR(100) NOT NULL,
   Generacion VARCHAR(50) NOT NULL,
   FOREIGN KEY(Fabricante) REFERENCES
esquema_videojuegos.Fabricante(Nombre)
);

-- Videojuego
CREATE TABLE esquema_videojuegos.Videojuego (
   Nombre VARCHAR(100) PRIMARY KEY,
   Desarrolladora VARCHAR(100) NOT NULL,
```

```
    Consola VARCHAR(100) NOT NULL,
    Precio NUMERIC(10,2) NOT NULL,
    FechaLanzamiento DATE NOT NULL,
    Puntuacion NUMERIC(3,1) NOT NULL,
    FOREIGN KEY(Desarrolladora) REFERENCES
esquema_videojuegos.Desarrolladora(Nombre),
    FOREIGN KEY(Consola) REFERENCES esquema_videojuegos.Consola(Nombre),
    CHECK (Puntuacion >= 0 AND Puntuacion <= 10),
    CHECK (Precio >= 0)
);

-- GeneroVideojuego
CREATE TABLE esquema_videojuegos.GeneroVideojuego (
    Videojuego VARCHAR(100) NOT NULL,
    Genero VARCHAR(50) NOT NULL,
    PRIMARY KEY (Videojuego, Genero),
    FOREIGN KEY(Videojuego) REFERENCES
esquema_videojuegos.Videojuego(Nombre)
);

-- Usuario
CREATE TABLE esquema_videojuegos.Usuario (
    NombreUsuario VARCHAR(100) PRIMARY KEY,
    FechaCreacion DATE NOT NULL,
    Nombre VARCHAR(100) NOT NULL,
    Apellido1 VARCHAR(50) NOT NULL,
    Apellido2 VARCHAR(50) NOT NULL,
    Email VARCHAR(100) NOT NULL,
    Pais VARCHAR(100) NOT NULL,
    Saldo NUMERIC(10,2) NOT NULL,
    EsPremium BOOLEAN NOT NULL
);

-- Usuario Premium
CREATE TABLE esquema_videojuegos.UsuarioPremium (
    NombreUsuario VARCHAR(100) PRIMARY KEY,
    FechaCaducidad DATE NOT NULL,
    RenovacionAutomatica BOOLEAN NOT NULL,
    FOREIGN KEY(NombreUsuario) REFERENCES
esquema_videojuegos.Usuario(NombreUsuario)
);

-- Usuario Corriente
```

```sql
CREATE TABLE esquema_videojuegos.UsuarioCorriente (
   NombreUsuario VARCHAR(100) PRIMARY KEY,
   NumeroAnunciosPorSesion INTEGER NOT NULL,
   FOREIGN KEY(NombreUsuario) REFERENCES
esquema_videojuegos.Usuario(NombreUsuario)
);

-- PosesionVideojuego
CREATE TABLE esquema_videojuegos.PosesionVideojuego (
   NombreUsuario VARCHAR(100) NOT NULL,
   Videojuego VARCHAR(100) NOT NULL,
   FechaCompra DATE NOT NULL,
   NumHorasJugadas INTEGER NOT NULL,
   PRIMARY KEY (NombreUsuario, Videojuego),
   FOREIGN KEY(NombreUsuario) REFERENCES
esquema_videojuegos.Usuario(NombreUsuario),
   FOREIGN KEY(Videojuego) REFERENCES
esquema_videojuegos.Videojuego(Nombre)
);

-- Insertar datos en la tabla 'Compania'
INSERT INTO esquema_videojuegos.Compania (Nombre, Director,
FechaCreacion, Pais) VALUES
('Ubisoft', 'Yves Guillemot', '1986-03-28', 'Francia'),
('Electronic Arts', 'Andrew Wilson', '1982-05-28', 'Estados Unidos'),
('Naughty Dog', 'Evan Wells', '1984-01-01', 'Estados Unidos'),
('Sony', 'Kenichiro Yoshida', '1946-05-07', 'Japón'),
('Microsoft', 'Satya Nadella', '1975-04-04', 'Estados Unidos');

-- Insertar datos en la tabla 'Desarrolladora'
INSERT INTO esquema_videojuegos.Desarrolladora (Nombre,
LicenciaCopyright) VALUES
('Ubisoft', 'Copyright 2025 Ubisoft Entertainment'),
('Electronic Arts', 'Copyright 2025 Electronic Arts'),
('Naughty Dog', 'Copyright 2025 Naughty Dog Studios');

-- Insertar datos en la tabla 'Fabricante'
INSERT INTO esquema_videojuegos.Fabricante (Nombre,
LicenciaFabricacion) VALUES
('Sony', 'Copyright 2025 Sony Interactive Entertainment'),
('Microsoft', 'Copyright 2025 Microsoft Corp.');

-- Insertar datos en la tabla 'Consola'
```

```sql
INSERT INTO esquema_videojuegos.Consola (Nombre, FechaLanzamiento,
NumVentas, Fabricante, Generacion) VALUES
('PlayStation 5', '2020-11-12', 50000000, 'Sony', 'Novena'),
('Xbox Series X', '2020-11-10', 40000000, 'Microsoft', 'Novena');

-- Insertar datos en la tabla 'Videojuego'
INSERT INTO esquema_videojuegos.Videojuego (Nombre, Desarrolladora,
Consola, Precio, FechaLanzamiento, Puntuacion) VALUES
('The Last of Us Part II', 'Naughty Dog', 'PlayStation 5', 59.99,
'2020-06-19', 9.8),
('FIFA 21', 'Electronic Arts', 'PlayStation 5', 59.99, '2020-10-09',
8.5),
('Assassins Creed Valhalla', 'Ubisoft', 'PlayStation 5', 59.99,
'2020-11-10', 8.7);

-- Insertar datos en la tabla 'GeneroVideojuego'
INSERT INTO esquema_videojuegos.GeneroVideojuego (Videojuego, Genero)
VALUES
('The Last of Us Part II', 'Aventura'),
('The Last of Us Part II', 'Acción'),
('The Last of Us Part II', 'Historia'),
('FIFA 21', 'Deportes'),
('Assassins Creed Valhalla', 'Acción');

-- Insertar datos en la tabla 'Usuario'
INSERT INTO esquema_videojuegos.Usuario (NombreUsuario, FechaCreacion,
Nombre, Apellido1, Apellido2, Email, Pais, Saldo, EsPremium) VALUES
('user1', '2021-01-01', 'Juan', 'Pérez', 'García', 'juan@example.com',
'España', 100.00, TRUE),
('user2', '2021-02-15', 'Ana', 'López', 'Martínez', 'ana@example.com',
'México', 50.00, FALSE);

-- Insertar datos en la tabla 'UsuarioPremium'
INSERT INTO esquema_videojuegos.UsuarioPremium (NombreUsuario,
FechaCaducidad, RenovacionAutomatica) VALUES
('user1', '2023-01-01', TRUE);

-- Insertar datos en la tabla 'UsuarioCorriente'
INSERT INTO esquema_videojuegos.UsuarioCorriente (NombreUsuario,
NumeroAnunciosPorSesion) VALUES
('user2', 5);

-- Insertar datos en la tabla 'PosesionVideojuego'
```

```
INSERT INTO esquema_videojuegos.PosesionVideojuego (NombreUsuario,
Videojuego, FechaCompra, NumHorasJugadas) VALUES
('user1', 'The Last of Us Part II', '2021-01-05', 10),
('user1', 'Assassins Creed Valhalla', '2022-11-10', 25),
('user2', 'The Last of Us Part II', '2021-02-15', 15),
('user2', 'FIFA 21', '2021-03-01', 5);

SELECT * FROM esquema_videojuegos.Compania;
SELECT * FROM esquema_videojuegos.Desarrolladora;
SELECT * FROM esquema_videojuegos.Fabricante;
SELECT * FROM esquema_videojuegos.Consola;
SELECT * FROM esquema_videojuegos.Videojuego;
SELECT * FROM esquema_videojuegos.GeneroVideojuego;
SELECT * FROM esquema_videojuegos.Usuario;
SELECT * FROM esquema_videojuegos.UsuarioPremium;
SELECT * FROM esquema_videojuegos.UsuarioCorriente;
SELECT * FROM esquema_videojuegos.PosesionVideojuego;
```

La creación de tablas de la segunda base de datos queda así:

```
-- Crear esquema si no existe
CREATE SCHEMA IF NOT EXISTS esquema_videojuegos_en;

-- Borrar tablas existentes en orden de dependencias
DROP TABLE IF EXISTS esquema_videojuegos_en.GameOwnership CASCADE;
DROP TABLE IF EXISTS esquema_videojuegos_en.Videogame CASCADE;
DROP TABLE IF EXISTS esquema_videojuegos_en.Platform CASCADE;
DROP TABLE IF EXISTS esquema_videojuegos_en.Account CASCADE;
DROP TABLE IF EXISTS esquema_videojuegos_en.Company CASCADE;

-- Crear tablas

-- Company
CREATE TABLE esquema_videojuegos_en.Company (
   Name VARCHAR(100) PRIMARY KEY,
   Director VARCHAR(100) NOT NULL,
   CreatedAt DATE NOT NULL,
   Country VARCHAR(100) NOT NULL,
   License VARCHAR(500) NOT NULL,
   Type VARCHAR(50) NOT NULL,
   CONSTRAINT chk_company_type CHECK (Type IN ('Desarrolladora',
'Fabricante'))
```

```
);

-- Platform
CREATE TABLE esquema_videojuegos_en.Platform (
   Name VARCHAR(100) PRIMARY KEY,
   ReleaseDate DATE NOT NULL,
   SalesVolume INT NOT NULL,
   Manufacturer VARCHAR(100) NOT NULL,
   Generation VARCHAR(50) NOT NULL,
   FOREIGN KEY (Manufacturer) REFERENCES
esquema_videojuegos_en.Company(Name)
);

-- Videogame
CREATE TABLE esquema_videojuegos_en.Videogame (
   Name VARCHAR(100) PRIMARY KEY,
   Developer VARCHAR(100) NOT NULL,
   Console VARCHAR(100) NOT NULL,
   Price NUMERIC(10, 2) NOT NULL,
   ReleaseDate DATE NOT NULL,
   Rating INT NOT NULL,
   Genres VARCHAR(250) NOT NULL,
   FOREIGN KEY(Developer) REFERENCES
esquema_videojuegos_en.Company(Name) ON DELETE CASCADE,
   FOREIGN KEY(Console) REFERENCES
esquema_videojuegos_en.Platform(Name) ON DELETE CASCADE,
   CONSTRAINT chk_rating CHECK (Rating BETWEEN 0 AND 10),
   CONSTRAINT chk_price CHECK (Price >= 0)
);

-- Account
CREATE TABLE esquema_videojuegos_en.Account (
   NickName VARCHAR(100) PRIMARY KEY,
   RegisteredAt DATE NOT NULL,
   Name VARCHAR(100) NOT NULL,
   Email VARCHAR(100) NOT NULL,
   Country VARCHAR(100) NOT NULL,
   BalanceAmount NUMERIC(10, 2) NOT NULL,
   MembershipType VARCHAR(100) NOT NULL,
   ExpiresAt DATE,
   AutoRenew BOOLEAN,
   AdsPerSession INTEGER NOT NULL,
```

```
    CONSTRAINT chk_membership_type CHECK (MembershipType IN ('premium',
'base')),
    CONSTRAINT chk_ads_for_premium CHECK (
        (MembershipType = 'premium' AND AdsPerSession = 0) OR
MembershipType != 'premium'
    ),
    CONSTRAINT chk_base_account CHECK (
        (MembershipType = 'base' AND ExpiresAt IS NULL AND AutoRenew IS
NULL) OR MembershipType != 'base'
    )
);

-- GameOwnership
CREATE TABLE esquema_videojuegos_en.GameOwnership (
    NickName VARCHAR(100),
    Videogame VARCHAR(100),
    PurchasedAt DATE NOT NULL,
    TotalPlaytime INT NOT NULL,
    PRIMARY KEY (NickName, Videogame),
    FOREIGN KEY (NickName) REFERENCES
esquema_videojuegos_en.Account(NickName) ON DELETE CASCADE,
    FOREIGN KEY (Videogame) REFERENCES
esquema_videojuegos_en.Videogame(Name) ON DELETE CASCADE
);

-- Insertar datos en la tabla 'Company'
INSERT INTO esquema_videojuegos_en.Company (Name, Director, CreatedAt,
Country, License, Type) VALUES
('Ubisoft', 'Yves Guillemot', '1986-03-28', 'France', 'Copyright 2025
Ubisoft Entertainment', 'Desarrolladora'),
('Electronic Arts', 'Andrew Wilson', '1982-05-28', 'USA', 'Copyright
2025 Electronic Arts', 'Desarrolladora'),
('Naughty Dog', 'Evan Wells', '1984-01-01', 'USA', 'Copyright 2025
Naughty Dog Studios', 'Desarrolladora'),
('Sony', 'Kenichiro Yoshida', '1946-05-07', 'Japan', 'Copyright 2025
Sony', 'Fabricante'),
('Microsoft', 'Satya Nadella', '1975-04-04', 'USA', 'Copyright 2025
Microsoft', 'Fabricante'),
('Activision', 'Bobby Kotick', '1979-10-01', 'USA', 'Copyright 2025
Activision Blizzard', 'Desarrolladora'),
('Rockstar Games', 'Sam Houser', '1998-12-01', 'USA', 'Copyright 2025
Rockstar Games', 'Desarrolladora'),
```

```
('Valve Corporation', 'Gabe Newell', '1996-08-24', 'USA', 'Copyright
2025 Valve Corporation', 'Desarrolladora'),
('Nintendo', 'Shuntaro Furukawa', '1889-09-23', 'Japan', 'Copyright
2025 Nintendo Co., Ltd.', 'Desarrolladora'),
('CD Projekt Red', 'Adam Kicinski', '2002-05-01', 'Poland', 'Copyright
2025 CD Projekt', 'Desarrolladora'),
('Epic Games', 'Tim Sweeney', '1991-07-01', 'USA', 'Copyright 2025 Epic
Games', 'Desarrolladora');

-- Insertar datos en la tabla 'Platform'
INSERT INTO esquema_videojuegos_en.Platform (Name, ReleaseDate,
SalesVolume, Manufacturer, Generation) VALUES
('PlayStation 5', '2020-11-12', 50000000, 'Sony', 'Ninth'),
('Nintendo Switch', '2017-03-03', 100000000, 'Nintendo', 'Eighth'),
('Xbox One', '2013-11-22', 50000000, 'Microsoft', 'Eighth'),
('PlayStation 4', '2013-11-15', 116000000, 'Sony', 'Eighth'),
('Xbox Series S', '2020-11-10', 20000000, 'Microsoft', 'Ninth'),
('Xbox Series X', '2020-11-10', 40000000, 'Microsoft', 'Ninth');

-- Insertar datos en la tabla 'Videogame'
INSERT INTO esquema_videojuegos_en.Videogame (Name, Developer, Console,
Price, ReleaseDate, Rating, Genres) VALUES
('The Last of Us Part II', 'Naughty Dog', 'PlayStation 5', 59.99,
'2020-06-19', 9.8, 'Adventure'),
('FIFA 21', 'Electronic Arts', 'PlayStation 5', 59.99, '2020-10-09',
8.5, 'Sports'),
('Mario Kart 8 Deluxe', 'Nintendo', 'Nintendo Switch', 59.99,
'2017-04-28', 9.5, 'Racing'),
('Assassins Creed Valhalla', 'Ubisoft', 'PlayStation 5', 59.99,
'2020-11-10', 8.7, 'Action');

-- Insertar datos en la tabla 'Account'
INSERT INTO esquema_videojuegos_en.Account (NickName, RegisteredAt,
Name, Email, Country, BalanceAmount, MembershipType, ExpiresAt,
AutoRenew, AdsPerSession) VALUES
('user1', '2021-01-01', 'Juan', 'juan@example.com', 'Spain', 100.00,
'premium', '2023-01-01', TRUE, 0),
('user3', '2024-12-11', 'Cristiano Ronaldo dos Santos Aveiro',
'cr7@example.com', 'Portugal', 60.15, 'premium', '2024-01-01', TRUE,
0),
('user2', '2021-02-15', 'Ana', 'ana@example.com', 'Mexico', 50.00,
'base', NULL, NULL, 5);
```

```sql
-- Insertar datos en la tabla 'GameOwnership'
INSERT INTO esquema_videojuegos_en.GameOwnership (NickName, Videogame,
PurchasedAt, TotalPlaytime) VALUES
('user1', 'The Last of Us Part II', '2021-01-05', 10),
('user1', 'FIFA 21', '2021-01-10', 20),
('user1', 'Mario Kart 8 Deluxe', '2021-01-15', 15),
('user3', 'Mario Kart 8 Deluxe', '2024-12-12', 15),
('user3', 'The Last of Us Part II', '2024-12-13', 5),
('user2', 'Assassins Creed Valhalla', '2021-03-01', 30),
('user2', 'FIFA 21', '2021-03-01', 5);

SELECT * FROM esquema_videojuegos_en.Videogame;
SELECT * FROM esquema_videojuegos_en.Platform;
SELECT * FROM esquema_videojuegos_en.Company;
SELECT * FROM esquema_videojuegos_en.Account;
SELECT * FROM esquema_videojuegos_en.GameOwnership;
```

Por último, se va a definir el esquema global que englobe a ambas bases de datos. Para ello, el proceso consta de dos fases:

- Traducción: Por razones humanas, es casi imposible que una base de datos sea exactamente igual a la otra, ya sea por razones de mera nomenclatura o por el empleo de otro formato de tablas. Por ello, antes de realizar una integración de las bases de datos en un esquema global, es importante que ambas bases de datos contengan un formato que sea similar al otro para no tener problemas de integración.
- Integración: Una vez traducido el formato de ambas bases de datos y se ha logrado que ambas bases de datos (en este caso vistas ya que lo que son las bases de datos locales no se tocan) mantengan un mismo formato, toca la fase de integración, donde se crea la vista global uniendo todas las filas de una base de datos y todas las filas de la otra.

Una vez explicado esto, el código de la integración del esquema global será el siguiente:

```sql
-- Crear base de datos 'db_global' (esto no se puede hacer desde una
conexión a 'db_global' directamente, así que hazlo desde 'postgres')
-- Solo si no existe
DO $$
BEGIN
  IF NOT EXISTS (SELECT FROM pg_database WHERE datname = 'db_global')
THEN
     EXECUTE 'CREATE DATABASE db_global';
  END IF;
END
$$;
```

```
-- Cambiar de base de datos: asegúrate de conectarte a 'db_global'
antes de ejecutar lo siguiente

-- Habilitar extensión dblink
CREATE EXTENSION IF NOT EXISTS dblink;

-- Crear esquema
CREATE SCHEMA IF NOT EXISTS esquema_global;

-- Desconectar si había conexiones anteriores activas (por limpieza)
SELECT dblink_disconnect('db1_connection') WHERE
dblink_get_connections() @> ARRAY['db1_connection'];
SELECT dblink_disconnect('db2_connection') WHERE
dblink_get_connections() @> ARRAY['db2_connection'];

-- Crear conexiones dblink
SELECT dblink_connect('db1_connection', 'dbname=db1 user=postgres
password=admin host=localhost');
SELECT dblink_connect('db2_connection', 'dbname=db2 user=postgres
password=admin host=localhost');

-- Eliminar vistas y triggers si ya existían
DROP VIEW IF EXISTS esquema_global.GlobalCompany CASCADE;
DROP VIEW IF EXISTS esquema_global.GlobalVideogame CASCADE;
DROP VIEW IF EXISTS esquema_global.GlobalPlatform CASCADE;
DROP VIEW IF EXISTS esquema_global.GlobalUser CASCADE;
DROP VIEW IF EXISTS esquema_global.GlobalOwnership CASCADE;

-- Crear vista global unificada de compañías
CREATE OR REPLACE VIEW esquema_global.GlobalCompany AS

-- Parte 1: Desarrolladoras desde db1
SELECT
   db1_desarrolladora.Nombre AS CompanyName,
   db1_companies.Director,
   db1_companies.FechaCreacion AS CreationDate,
   db1_companies.Pais AS Country,
   db1_desarrolladora.LicenciaCopyright AS License,
   'Desarrolladora' AS Type,
   'db1' AS Source
FROM dblink('db1_connection', 'SELECT Nombre, LicenciaCopyright FROM
esquema_videojuegos.Desarrolladora')
```

```
AS db1_desarrolladora(Nombre VARCHAR(300), LicenciaCopyright
VARCHAR(300))
LEFT JOIN dblink('db1_connection', 'SELECT Nombre, Director,
FechaCreacion, Pais FROM esquema_videojuegos.Compania')
AS db1_companies(Nombre VARCHAR(300), Director VARCHAR(300),
FechaCreacion DATE, Pais VARCHAR(300))
ON db1_desarrolladora.Nombre = db1_companies.Nombre

UNION

-- Parte 2: Fabricantes desde db1
SELECT
   db1_fabricante.Nombre AS CompanyName,
   db1_companies.Director,
   db1_companies.FechaCreacion AS CreationDate,
   db1_companies.Pais AS Country,
   db1_fabricante.LicenciaFabricacion AS License,
   'Fabricante' AS Type,
   'db1' AS Source
FROM dblink('db1_connection', 'SELECT Nombre, LicenciaFabricacion FROM
esquema_videojuegos.Fabricante')
AS db1_fabricante(Nombre VARCHAR(300), LicenciaFabricacion
VARCHAR(300))
LEFT JOIN dblink('db1_connection', 'SELECT Nombre, Director,
FechaCreacion, Pais FROM esquema_videojuegos.Compania')
AS db1_companies(Nombre VARCHAR(300), Director VARCHAR(300),
FechaCreacion DATE, Pais VARCHAR(300))
ON db1_fabricante.Nombre = db1_companies.Nombre

UNION

-- Parte 3: Compañías desde db2
SELECT DISTINCT ON (en_company.Name)
   en_company.Name AS CompanyName,
   en_company.Director,
   en_company.CreatedAt AS CreationDate,
   en_company.Country,
   en_company.License AS License,
   en_company.Type AS Type,
   'db2' AS Source
FROM dblink('db2_connection', 'SELECT Name, Director, CreatedAt,
Country, License, Type FROM esquema_videojuegos_en.Company')
```

```
AS en_company(Name VARCHAR(300), Director VARCHAR(300), CreatedAt DATE,
Country VARCHAR(300), License VARCHAR(300), Type VARCHAR(100))
WHERE en_company.Name NOT IN (
   SELECT Nombre FROM dblink('db1_connection', 'SELECT Nombre FROM
esquema_videojuegos.Desarrolladora')
   AS devs(Nombre VARCHAR(300))
   UNION
   SELECT Nombre FROM dblink('db1_connection', 'SELECT Nombre FROM
esquema_videojuegos.Fabricante')
   AS fabs(Nombre VARCHAR(300))
);

CREATE OR REPLACE VIEW esquema_global.GlobalVideogame AS

-- Parte 1: Videojuegos desde db1
SELECT
   vj.Nombre AS Name,
   vj.Desarrolladora AS Developer,
   vj.Consola AS Console,
   vj.Precio AS Price,
   vj.FechaLanzamiento AS ReleaseDate,
   vj.Puntuacion AS Rating,
   gen.Generos AS Genres,
   'db1' AS Source
FROM dblink('db1_connection', '
   SELECT v.Nombre, v.Desarrolladora, v.Consola, v.Precio,
v.FechaLanzamiento, v.Puntuacion
   FROM esquema_videojuegos.Videojuego v
') AS vj(Nombre VARCHAR(300), Desarrolladora VARCHAR(300), Consola
VARCHAR(300), Precio NUMERIC(10,2), FechaLanzamiento DATE, Puntuacion
NUMERIC(3,1))
LEFT JOIN (
   SELECT
       gv.Videojuego,
       STRING_AGG(gv.Genero, ', ') AS Generos
   FROM dblink('db1_connection', '
       SELECT Videojuego, Genero FROM
esquema_videojuegos.GeneroVideojuego
   ') AS gv(Videojuego VARCHAR(300), Genero VARCHAR(300))
   GROUP BY gv.Videojuego
) AS gen ON vj.Nombre = gen.Videojuego

UNION
```

```
-- Parte 2: Videojuegos desde db2
SELECT
   vg.Name,
   vg.Developer,
   vg.Console,
   vg.Price,
   vg.ReleaseDate,
   vg.Rating,
   vg.Genres,
   'db2' AS Source
FROM dblink('db2_connection', '
   SELECT Name, Developer, Console, Price, ReleaseDate, Rating, Genres
FROM esquema_videojuegos_en.Videogame
') AS vg(Name VARCHAR(255), Developer VARCHAR(255), Console
VARCHAR(255), Price NUMERIC(10,2), ReleaseDate DATE, Rating INT, Genres
VARCHAR(255))
WHERE vg.Name NOT IN (
   SELECT Nombre FROM dblink('db1_connection', '
       SELECT Nombre FROM esquema_videojuegos.Videojuego
   ') AS sub(Nombre VARCHAR(300))
);

CREATE OR REPLACE VIEW esquema_global.GlobalPlatform AS

-- Parte 1: Consolas desde db1
SELECT
   c.Nombre AS Name,
   c.FechaLanzamiento AS ReleaseDate,
   c.NumVentas AS SalesVolume,
   c.Fabricante AS Manufacturer,
   c.Generacion AS Generation,
   'db1' AS Source
FROM dblink('db1_connection', '
   SELECT Nombre, FechaLanzamiento, NumVentas, Fabricante, Generacion
   FROM esquema_videojuegos.Consola
') AS c(Nombre VARCHAR(300), FechaLanzamiento DATE, NumVentas INTEGER,
Fabricante VARCHAR(300), Generacion VARCHAR(300))

UNION

-- Parte 2: Plataformas desde db2
SELECT
```

```
    p.Name,
    p.ReleaseDate,
    p.SalesVolume,
    p.Manufacturer,
    p.Generation,
    'db2' AS Source
FROM dblink('db2_connection', '
    SELECT Name, ReleaseDate, SalesVolume, Manufacturer, Generation
    FROM esquema_videojuegos_en.Platform
') AS p(Name VARCHAR(255), ReleaseDate DATE, SalesVolume INT,
Manufacturer VARCHAR(255), Generation VARCHAR(255))
WHERE p.Name NOT IN (
    SELECT Nombre FROM dblink('db1_connection', '
        SELECT Nombre FROM esquema_videojuegos.Consola
    ') AS sub(Nombre VARCHAR(300))
);

CREATE OR REPLACE VIEW esquema_global.GlobalUser AS

-- Parte 1: Usuario Premium (db1)
SELECT
    u.NombreUsuario AS NickName,
    u.FechaCreacion AS RegisteredAt,
    CONCAT(u.Nombre, ' ', u.Apellido1, ' ', u.Apellido2) AS Name,
    u.Email,
    u.Pais AS Country,
    u.Saldo AS BalanceAmount,
    'premium' AS MembershipType,
    up.FechaCaducidad AS ExpiresAt,
    up.RenovacionAutomatica AS AutoRenew,
    0 AS AdsPerSession,
    'db1' AS Source
FROM dblink('db1_connection', '
    SELECT NombreUsuario, FechaCreacion, Nombre, Apellido1, Apellido2,
Email, Pais, Saldo, EsPremium
    FROM esquema_videojuegos.Usuario WHERE EsPremium = true
') AS u(NombreUsuario VARCHAR(300), FechaCreacion DATE, Nombre
VARCHAR(300), Apellido1 VARCHAR(300), Apellido2 VARCHAR(300), Email
VARCHAR(300), Pais VARCHAR(300), Saldo NUMERIC(10,2), EsPremium
BOOLEAN)
JOIN dblink('db1_connection', '
    SELECT NombreUsuario, FechaCaducidad, RenovacionAutomatica FROM
esquema_videojuegos.UsuarioPremium
```

```
') AS up(NombreUsuario VARCHAR(300), FechaCaducidad DATE,
RenovacionAutomatica BOOLEAN)
ON u.NombreUsuario = up.NombreUsuario

UNION

-- Parte 2: Usuario Corriente (db1)
SELECT
   u.NombreUsuario AS NickName,
   u.FechaCreacion AS RegisteredAt,
   CONCAT(u.Nombre, ' ', u.Apellido1, ' ', u.Apellido2) AS Name,
   u.Email,
   u.Pais AS Country,
   u.Saldo AS BalanceAmount,
   'base' AS MembershipType,
   NULL AS ExpiresAt,
   NULL AS AutoRenew,
   uc.NumeroAnunciosPorSesion AS AdsPerSession,
   'db1' AS Source
FROM dblink('db1_connection', '
   SELECT NombreUsuario, FechaCreacion, Nombre, Apellido1, Apellido2,
Email, Pais, Saldo, EsPremium
   FROM esquema_videojuegos.Usuario WHERE EsPremium = false
') AS u(NombreUsuario VARCHAR(300), FechaCreacion DATE, Nombre
VARCHAR(300), Apellido1 VARCHAR(300), Apellido2 VARCHAR(300), Email
VARCHAR(300), Pais VARCHAR(300), Saldo NUMERIC(10,2), EsPremium
BOOLEAN)
JOIN dblink('db1_connection', '
   SELECT NombreUsuario, NumeroAnunciosPorSesion FROM
esquema_videojuegos.UsuarioCorriente
') AS uc(NombreUsuario VARCHAR(300), NumeroAnunciosPorSesion INTEGER)
ON u.NombreUsuario = uc.NombreUsuario

UNION

-- Parte 3: Cuentas desde db2
SELECT
   a.NickName,
   a.RegisteredAt,
   a.Name,
   a.Email,
   a.Country,
   a.BalanceAmount,
```

```
    a.MembershipType,
    a.ExpiresAt,
    a.AutoRenew,
    a.AdsPerSession,
    'db2' AS Source
FROM dblink('db2_connection', '
    SELECT NickName, RegisteredAt, Name, Email, Country, BalanceAmount,
MembershipType, ExpiresAt, AutoRenew, AdsPerSession
    FROM esquema_videojuegos_en.Account
') AS a(NickName VARCHAR(255), RegisteredAt DATE, Name VARCHAR(255),
Email VARCHAR(255), Country VARCHAR(255), BalanceAmount NUMERIC(10,2),
MembershipType VARCHAR(255), ExpiresAt DATE, AutoRenew BOOLEAN,
AdsPerSession INTEGER)
WHERE a.NickName NOT IN (
    SELECT NombreUsuario FROM dblink('db1_connection', '
        SELECT NombreUsuario FROM esquema_videojuegos.Usuario
    ') AS sub(NombreUsuario VARCHAR(300))
);

CREATE OR REPLACE VIEW esquema_global.GlobalOwnership AS

-- Parte 1: Posesiones desde db1
SELECT
    p.NombreUsuario AS NickName,
    p.Videojuego AS Videogame,
    p.FechaCompra AS PurchasedAt,
    p.NumHorasJugadas AS TotalPlaytime,
    'db1' AS Source
FROM dblink('db1_connection', '
    SELECT NombreUsuario, Videojuego, FechaCompra, NumHorasJugadas
    FROM esquema_videojuegos.PosesionVideojuego
') AS p(NombreUsuario VARCHAR(300), Videojuego VARCHAR(300),
FechaCompra DATE, NumHorasJugadas INTEGER)

UNION

-- Parte 2: Posesiones desde db2
SELECT
    o.NickName,
    o.Videogame,
    o.PurchasedAt,
    o.TotalPlaytime,
    'db2' AS Source
```

```
FROM dblink('db2_connection', '
   SELECT NickName, Videogame, PurchasedAt, TotalPlaytime
   FROM esquema_videojuegos_en.GameOwnership
') AS o(NickName VARCHAR(255), Videogame VARCHAR(255), PurchasedAt
DATE, TotalPlaytime INT)
WHERE (o.NickName, o.Videogame) NOT IN (
   SELECT NombreUsuario, Videojuego FROM dblink('db1_connection', '
       SELECT NombreUsuario, Videojuego
       FROM esquema_videojuegos.PosesionVideojuego
   ') AS sub(NombreUsuario VARCHAR(300), Videojuego VARCHAR(300))
);
```

Aspectos a considerar

- Gestión de géneros: Se ha decidido mantener la versión de la db2, donde todos los géneros se han mantenido en un solo atributo y separados por comas. Se ha considerado esta la opción más viable por ahorrar la creación de una vista adicional sólo para la gestión de géneros. Otra razón de peso es que, al no existir la necesidad de realizar consultas hacia un género en concreto de un videojuego en concreto, no hacía falta crear otra vista solo para guardar cada género de cada videojuego por separado.
- Gestión de usuarios: Se ha decidido eliminar la jerarquía de usuarios para reducir el número de tablas y simplificar las futuras operaciones de inserción en el esquema global. Por ello, se guarda el tipo de usuario, además de los atributos correspondientes a ese tipo de usuario, ignorando el resto de atributos que no correspondan al tipo. Además, se ha decidido eliminar los atributos Apellido1 y Apellido2 y añadir dichos atributos al Nombre, teniendo ahora en un solo atributo tanto el nombre como los apellidos del usuario.
- Gestión de compañías: Se ha decidido, al igual que en los usuarios, eliminar la jerarquía de compañías. Por ello, el tipo de compañía se gestiona mediante un atributo. Además, se ha decidido tener un solo atributo de licencia cuando en la versión 1 de la base de datos se tenía una licencia por cada subtipo de compañía. La decisión viene dada porque realmente el atributo como tal es el mismo, solo que lo que lo hace verse de una manera o de otra es su tipo (atributo ya gestionado en este esquema global), por lo que el problema desaparece.

Actualizaciones de datos sobre el esquema global

Actualizaciones de datos sobre el esquema global: inserciones y modificaciones mediante triggers INSTEAD OF en Oracle

En el contexto de bases de datos federadas o distribuidas, como la nuestra en Oracle con vistas globales que unifican datos de múltiples fuentes, surge una limitación importante: **no es posible realizar directamente operaciones de escritura (INSERT, UPDATE, DELETE) sobre vistas que integran datos de diferentes tablas o provenientes de esquemas remotos**. Para solventar esta restricción, Oracle ofrece la posibilidad de definir *triggers* del

tipo `INSTEAD OF`, los cuales permiten interceptar dichas operaciones sobre la vista y redirigirlas manualmente a las tablas físicas subyacentes del esquema local.

En esta práctica, se ha implementado un conjunto de *triggers INSTEAD OF* sobre diversas vistas del esquema global con el objetivo de permitir inserciones y actualizaciones de forma controlada y coherente, respetando tanto las restricciones semánticas como la lógica del dominio bancario modelado.

Dado que no se puede modificar directamente el contenido de una vista compleja (especialmente aquellas que combinan múltiples tablas o usan joins), los *triggers INSTEAD OF* resultan la herramienta adecuada y **necesaria** para permitir operaciones sobre dichas vistas. En este caso, el esquema define varias vistas como `VistaCuenta`, `VistaSucursal`, `VistaCuentaCliente`, `VistaOperacionBancaria`, `VistaCuentaCorriente` y `VistaCuentaAhorro`, cada una de las cuales representa una abstracción de datos provenientes de distintas tablas base del modelo bancario relacional.

Mediante estos *triggers*, se consigue:

- Permitir la modificación de datos desde el nivel lógico (vistas globales), facilitando la interoperabilidad del sistema.
- Encapsular la lógica de validación y enrutamiento de los datos, asegurando la integridad sin comprometer las restricciones del modelo conceptual.
- Centralizar el control de errores, mediante el uso de `RAISE_APPLICATION_ERROR`, para emitir mensajes personalizados ante situaciones inválidas.

Los triggers son estos

```
DROP TRIGGER VistaCuenta_Insert;

DROP TRIGGER VistaCuenta_Update;

DROP TRIGGER VistaCuentaCliente_Insert;

DROP TRIGGER VistaOperacionBancaria_Insert;

DROP TRIGGER VistaOperacionBancaria_Update;

DROP TRIGGER VistaSucursal_Insert;

DROP TRIGGER VistaAhorro_Insert;

DROP TRIGGER VistaCorriente_Insert;
```

```
CREATE OR REPLACE TRIGGER VistaCuenta_Insert
INSTEAD OF INSERT ON VistaCuenta
FOR EACH ROW
DECLARE
 v_len NUMBER;
BEGIN
 v_len := LENGTH(:NEW.iban);
 IF v_len < 5 THEN
   RAISE_APPLICATION_ERROR(-20001, 'IBAN inválido: longitud mínima no
alcanzada');
 END IF;

 -- 1) Inserta en la tabla base "Cuenta"
 INSERT INTO Cuenta (
   iban,
   numero_cuenta,
   saldo,
   fecha_creacion
 ) VALUES (
   :NEW.iban,
   :NEW.numero_cuenta,
   :NEW.saldo,
   :NEW.fecha_creacion
 );
```

```
END;

/

CREATE OR REPLACE TRIGGER VistaCorriente_Insert

INSTEAD OF INSERT ON VistaCuentaCorriente

FOR EACH ROW

BEGIN

 -- Inserta en la tabla Corriente

 INSERT INTO Corriente (

   iban,

   codigo_sucursal

 ) VALUES (

   :NEW.iban,

   :NEW.codigo_sucursal

 );

 END;

/



CREATE OR REPLACE TRIGGER VistaAhorro_Insert

INSTEAD OF INSERT ON VistaCuentaAhorro

FOR EACH ROW

BEGIN

 -- Inserta en la tabla Ahorro

 INSERT INTO Ahorro (

   iban,

   interes
```

```
 ) VALUES (
   :NEW.iban,
   :NEW.interes
 );
END;
/



CREATE OR REPLACE TRIGGER VistaCuenta_Update
INSTEAD OF UPDATE ON VistaCuenta
FOR EACH ROW
DECLARE
 v_exists NUMBER;
 v_old_iban VARCHAR2(34);
BEGIN
 -- Verificar si la cuenta antigua existe
 v_old_iban := :OLD.iban;
 SELECT COUNT(*) INTO v_exists
   FROM Cuenta
  WHERE iban = v_old_iban;


 IF v_exists = 0 THEN
   RAISE_APPLICATION_ERROR(-20031, 'No se puede actualizar: la cuenta
no existe en la base local');
 ELSE
```

```
   -- Actualiza la tabla "Cuenta"
   UPDATE Cuenta
      SET saldo          = :NEW.saldo,
          numero_cuenta  = :NEW.numero_cuenta,
          fecha_creacion = :NEW.fecha_creacion
    WHERE iban = v_old_iban;
 END IF;
END;
/



CREATE OR REPLACE TRIGGER VistaCuentaCliente_Insert
INSTEAD OF INSERT ON VistaCuentaCliente
FOR EACH ROW
BEGIN
 INSERT INTO CuentaCliente (dni, iban)
 VALUES (:NEW.dni, :NEW.iban);
END;
/



CREATE OR REPLACE TRIGGER VistaCuentaCliente_Insert
INSTEAD OF INSERT ON VistaCuentaCliente
FOR EACH ROW
BEGIN
```

```
 INSERT INTO CuentaCliente (dni, iban)
 VALUES (:NEW.dni, :NEW.iban);
END;
/



CREATE OR REPLACE TRIGGER VistaOperacionBancaria_Insert
INSTEAD OF INSERT ON VistaOperacionBancaria
FOR EACH ROW
BEGIN
 INSERT INTO OperacionBancaria (
   codigo_numerico,
   iban,
   cantidad,
   fecha,
   hora
 ) VALUES (
   :NEW.codigo_numerico,
   :NEW.iban,
   :NEW.cantidad,
   :NEW.fecha,
   :NEW.hora
 );
END;
/
```

```
CREATE OR REPLACE TRIGGER VistaOperacionBancaria_Update
INSTEAD OF UPDATE ON VistaOperacionBancaria
FOR EACH ROW
DECLARE
 v_exists NUMBER;
BEGIN
 -- Comprueba si la operación existe en la tabla local
 SELECT COUNT(*) INTO v_exists
   FROM OperacionBancaria
  WHERE codigo_numerico = :OLD.codigo_numerico
    AND iban = :OLD.iban;

 IF v_exists = 0 THEN
   RAISE_APPLICATION_ERROR(-20033, 'No se puede actualizar: la
operación no existe en la base local');
 ELSE
   -- Actualiza la operación principal
   UPDATE OperacionBancaria
      SET
          fecha       = :NEW.fecha,
          hora        = :NEW.hora,
          cantidad    = :NEW.cantidad
    WHERE codigo_numerico = :OLD.codigo_numerico
```

```
      AND iban = :OLD.iban;

 END IF;

END;

/



CREATE OR REPLACE TRIGGER VistaSucursal_Insert

INSTEAD OF INSERT ON VistaSucursal

FOR EACH ROW

BEGIN

 IF :NEW.codoficina IS NULL THEN

   RAISE_APPLICATION_ERROR(-20010, 'Código de oficina no puede ser
nulo');

 ELSIF :NEW.dir IS NULL THEN

   RAISE_APPLICATION_ERROR(-20011, 'Dirección no puede ser nula');

 ELSIF :NEW.tfno IS NULL THEN

   RAISE_APPLICATION_ERROR(-20012, 'Teléfono no puede ser nulo');

 END IF;



 IF LENGTH(:NEW.tfno) != 9 THEN

   RAISE_APPLICATION_ERROR(-20013, 'El teléfono debe tener 9 dígitos');

 END IF;



 -- Inserta en la tabla local Sucursal

 INSERT INTO Sucursal (
```

```
   codigo_sucursal,
   direccion,
   telefono
 ) VALUES (
   :NEW.codoficina,
   :NEW.dir,
   :NEW.tfno
 );
END;
/



CREATE OR REPLACE TRIGGER VistaSucursal_Update
INSTEAD OF UPDATE ON VistaSucursal
FOR EACH ROW
DECLARE
 v_exists NUMBER;
BEGIN
 SELECT COUNT(*) INTO v_exists
   FROM Sucursal
  WHERE codigo_sucursal = :OLD.codoficina;


 IF v_exists = 0 THEN
   RAISE_APPLICATION_ERROR(-20034, 'No se puede actualizar: la oficina
no existe en la base local');
```

```
 ELSE
   UPDATE Sucursal
      SET direccion = :NEW.dir,
          telefono  = :NEW.tfno
    WHERE codigo_sucursal = :OLD.codoficina;
 END IF;
END;
/



-- ======================================================
-- Ejemplos de inserciones y actualizaciones para probar los triggers
-- ======================================================


-- Insert en VistaCuenta
INSERT INTO VistaCuenta (
 iban,
 numero_cuenta,
 saldo,
 fecha_creacion
) VALUES (
 'ES1234567890123456789012345678901234567890',  -- IBAN válido (más de 5
caracteres)
 '00123456789',
 1000,
```

```
 TO_DATE('2025-04-01', 'YYYY-MM-DD')
);


-- Insert en VistaCuentaCliente (dni numérico)
INSERT INTO VistaCuentaCliente (
 dni,
 iban
) VALUES (
 12345678,
 'ES1234567890123456789012345678901'
);


-- Insert en VistaOperacionBancaria
INSERT INTO VistaOperacionBancaria (
 codigo_numerico,
 iban,
 cantidad,
 fecha,
 hora
) VALUES (
 1001,
 'ES1234567890123456789012345678901',
 250,
 TO_DATE('2025-04-05', 'YYYY-MM-DD'),
 TO_TIMESTAMP('2025-04-05 10:30:00', 'YYYY-MM-DD HH24:MI:SS')
```

```
);


-- Insert en VistaSucursal (codoficina numérico)
INSERT INTO VistaSucursal (
 codoficina,
 dir,
 tfno
) VALUES (
 100,           -- código numérico
 'Calle Falsa 123',
 '123456789'
);


-- 3. Insertar en VistaCorriente para activar el trigger que inserta en Corriente
-- y actualiza el tipo de cuenta a "CORRIENTE" en la tabla Cuenta
INSERT INTO VistaCuentaCorriente (iban, codigo_sucursal)
VALUES ('ES0000000000000000000000000001', 101);


-- 4. Insertar en VistaAhorro para activar el trigger que inserta en Ahorro
-- y actualiza el tipo de cuenta a "AHORRO" en la tabla Cuenta
INSERT INTO VistaCuentaAhorro (iban, interes)
VALUES ('ES0000000000000000000000000002', 2.50);
```

Al final se realizan una serie de inserts para probar todo el sistema que comprueban que los datos se inserten correctamente. En este sistema una vez actualizados los datos en el esquema local habría que volver a generar las vistas para que tengan los datos nuevos.

Actualizaciones de datos sobre el esquema global: inserciones y modificaciones mediante triggers INSTEAD OF en Postgres

Se ha evaluado la posibilidad de insertar, modificar y eliminar registros directamente desde el **esquema global (esquema_global)** en PostgreSQL, a través de las vistas construidas en el apartado anterior. Para ello, se han definido **triggers que aplican la operación INSTEAD OF** que dirige las operaciones a las tablas reales ubicadas en la base de datos **db2** directamente

Gracias a esta posibilidad, es posible trabajar sobre las vistas globales como si fuesen tablas normales, haciendo mucho más sencilla la escritura en cada una de las bases de datos locales a través del esquema global. Las operaciones de escritura (INSERT, UPDATE, DELETE) son interceptadas por los triggers y ejecutadas remotamente mediante **dblink_exec**.

Dichos trigger creados son los siguientes:

```
-- ============================

-- FUNCIONES PARA INSERTAR, MODIFICAR Y ELIMINAR EN DB2

-- ============================



-- 1. GlobalCompany

CREATE OR REPLACE FUNCTION esquema_global.insert_global_company()

RETURNS TRIGGER AS $$

BEGIN

   PERFORM dblink_exec('db2_connection',

       FORMAT('INSERT INTO esquema_videojuegos_en.Company (Name,
Director, CreatedAt, Country, License, Type)

               VALUES (%L, %L, %L, %L, %L, %L)',

               NEW.CompanyName, NEW.Director, NEW.CreationDate,
NEW.Country, NEW.License, NEW.Type));
```

```
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION esquema_global.update_global_company()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.Source = 'db2' THEN
        PERFORM dblink_exec('db2_connection',
            FORMAT('UPDATE esquema_videojuegos_en.Company
                    SET Director = %L,
                        CreatedAt = %L,
                        Country = %L,
                        License = %L,
                        Type = %L
                    WHERE Name = %L',
                    NEW.Director, NEW.CreationDate, NEW.Country,
NEW.License, NEW.Type, NEW.CompanyName));
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION esquema_global.delete_global_company()
RETURNS TRIGGER AS $$
```

```
BEGIN
   IF OLD.Source = 'db2' THEN
       PERFORM dblink_exec('db2_connection',
           FORMAT('DELETE FROM esquema_videojuegos_en.Company WHERE
Name = %L',
                  OLD.CompanyName));
   END IF;
   RETURN OLD;
END;
$$ LANGUAGE plpgsql;


-- 2. GlobalVideogame
CREATE OR REPLACE FUNCTION esquema_global.insert_global_videogame()
RETURNS TRIGGER AS $$
BEGIN
   PERFORM dblink_exec('db2_connection',
       FORMAT('INSERT INTO esquema_videojuegos_en.Videogame (Name,
Developer, Console, Price, ReleaseDate, Rating, Genres)
               VALUES (%L, %L, %L, %L, %L, %L, %L)',
               NEW.Name, NEW.Developer, NEW.Console, NEW.Price,
NEW.ReleaseDate, NEW.Rating, NEW.Genres));
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION esquema_global.update_global_videogame()
```

```
RETURNS TRIGGER AS $$

BEGIN

   IF NEW.Source = 'db2' THEN

       PERFORM dblink_exec('db2_connection',

           FORMAT('UPDATE esquema_videojuegos_en.Videogame

                   SET Developer = %L,

                       Console = %L,

                       Price = %L,

                       ReleaseDate = %L,

                       Rating = %L,

                       Genres = %L

                   WHERE Name = %L',

                   NEW.Developer, NEW.Console, NEW.Price,
NEW.ReleaseDate, NEW.Rating, NEW.Genres, OLD.Name));

   END IF;

   RETURN NEW;

END;

$$ LANGUAGE plpgsql;



CREATE OR REPLACE FUNCTION esquema_global.delete_global_videogame()

RETURNS TRIGGER AS $$

BEGIN

   IF OLD.Source = 'db2' THEN

       PERFORM dblink_exec('db2_connection',

           FORMAT('DELETE FROM esquema_videojuegos_en.Videogame WHERE
Name = %L',
```

```
                    OLD.Name));

    END IF;

    RETURN OLD;

END;

$$ LANGUAGE plpgsql;



-- 3. GlobalPlatform

CREATE OR REPLACE FUNCTION esquema_global.insert_global_platform()

RETURNS TRIGGER AS $$

BEGIN

    PERFORM dblink_exec('db2_connection',

        FORMAT('INSERT INTO esquema_videojuegos_en.Platform (Name,
ReleaseDate, SalesVolume, Manufacturer, Generation)

                VALUES (%L, %L, %L, %L, %L)',

                NEW.Name, NEW.ReleaseDate, NEW.SalesVolume,
NEW.Manufacturer, NEW.Generation));

    RETURN NEW;

END;

$$ LANGUAGE plpgsql;



CREATE OR REPLACE FUNCTION esquema_global.update_global_platform()

RETURNS TRIGGER AS $$

BEGIN

    IF NEW.Source = 'db2' THEN

        PERFORM dblink_exec('db2_connection',

            FORMAT('UPDATE esquema_videojuegos_en.Platform
```

```
                    SET ReleaseDate = %L,
                        SalesVolume = %L,
                        Manufacturer = %L,
                        Generation = %L
                    WHERE Name = %L',
                    NEW.ReleaseDate, NEW.SalesVolume, NEW.Manufacturer,
NEW.Generation, OLD.Name));
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION esquema_global.delete_global_platform()
RETURNS TRIGGER AS $$
BEGIN
    IF OLD.Source = 'db2' THEN
        PERFORM dblink_exec('db2_connection',
            FORMAT('DELETE FROM esquema_videojuegos_en.Platform WHERE
Name = %L',
                    OLD.Name));
    END IF;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;


-- 4. GlobalUser
```

```
CREATE OR REPLACE FUNCTION esquema_global.insert_global_user()

RETURNS TRIGGER AS $$

BEGIN

   PERFORM dblink_exec('db2_connection',

       FORMAT('INSERT INTO esquema_videojuegos_en.Account (NickName,
RegisteredAt, Name, Email, Country, BalanceAmount, MembershipType,
ExpiresAt, AutoRenew, AdsPerSession)

               VALUES (%L, %L, %L, %L, %L, %L, %L, %L, %L, %L)',

               NEW.NickName, NEW.RegisteredAt, NEW.Name, NEW.Email,
NEW.Country, NEW.BalanceAmount, NEW.MembershipType, NEW.ExpiresAt,
NEW.AutoRenew, NEW.AdsPerSession));

   RETURN NEW;

END;

$$ LANGUAGE plpgsql;



CREATE OR REPLACE FUNCTION esquema_global.update_global_user()

RETURNS TRIGGER AS $$

BEGIN

   IF NEW.Source = 'db2' THEN

       PERFORM dblink_exec('db2_connection',

           FORMAT('UPDATE esquema_videojuegos_en.Account

                   SET RegisteredAt = %L,

                       Name = %L,

                       Email = %L,

                       Country = %L,

                       BalanceAmount = %L,

                       MembershipType = %L,
```

```
                        ExpiresAt = %L,
                        AutoRenew = %L,
                        AdsPerSession = %L
                    WHERE NickName = %L',
                    NEW.RegisteredAt, NEW.Name, NEW.Email, NEW.Country,
NEW.BalanceAmount,
                    NEW.MembershipType, NEW.ExpiresAt, NEW.AutoRenew,
NEW.AdsPerSession,
                    OLD.NickName));
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION esquema_global.delete_global_user()
RETURNS TRIGGER AS $$
BEGIN
    IF OLD.Source = 'db2' THEN
        PERFORM dblink_exec('db2_connection',
            FORMAT('DELETE FROM esquema_videojuegos_en.Account WHERE
NickName = %L',
                    OLD.NickName));
    END IF;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;
```

```
-- 5. GlobalOwnership

CREATE OR REPLACE FUNCTION esquema_global.insert_global_ownership()

RETURNS TRIGGER AS $$

BEGIN

   PERFORM dblink_exec('db2_connection',

       FORMAT('INSERT INTO esquema_videojuegos_en.GameOwnership
(NickName, Videogame, PurchasedAt, TotalPlaytime)

               VALUES (%L, %L, %L, %L)',

               NEW.NickName, NEW.Videogame, NEW.PurchasedAt,
NEW.TotalPlaytime));

   RETURN NEW;

END;

$$ LANGUAGE plpgsql;



CREATE OR REPLACE FUNCTION esquema_global.update_global_ownership()

RETURNS TRIGGER AS $$

BEGIN

   IF NEW.Source = 'db2' THEN

       PERFORM dblink_exec('db2_connection',

           FORMAT('UPDATE esquema_videojuegos_en.GameOwnership

                   SET PurchasedAt = %L,

                       TotalPlaytime = %L

                   WHERE NickName = %L AND Videogame = %L',

                   NEW.PurchasedAt, NEW.TotalPlaytime, OLD.NickName,
OLD.Videogame));
```

```
    END IF;

    RETURN NEW;

END;

$$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION esquema_global.delete_global_ownership()

RETURNS TRIGGER AS $$

BEGIN

    IF OLD.Source = 'db2' THEN

        PERFORM dblink_exec('db2_connection',

            FORMAT('DELETE FROM esquema_videojuegos_en.GameOwnership

                    WHERE NickName = %L AND Videogame = %L',

                    OLD.NickName, OLD.Videogame));

    END IF;

    RETURN OLD;

END;

$$ LANGUAGE plpgsql;


-- ==========================

-- TRIGGERS SOBRE LAS VISTAS

-- ==========================


-- Trigger para GlobalCompany

CREATE TRIGGER trg_insert_global_company

INSTEAD OF INSERT ON esquema_global.GlobalCompany
```

```
FOR EACH ROW

EXECUTE FUNCTION esquema_global.insert_global_company();

CREATE TRIGGER trg_update_global_company

INSTEAD OF UPDATE ON esquema_global.GlobalCompany

FOR EACH ROW EXECUTE FUNCTION esquema_global.update_global_company();

CREATE TRIGGER trg_delete_global_company

INSTEAD OF DELETE ON esquema_global.GlobalCompany

FOR EACH ROW EXECUTE FUNCTION esquema_global.delete_global_company();



-- Trigger para GlobalVideogame

CREATE TRIGGER trg_insert_global_videogame

INSTEAD OF INSERT ON esquema_global.GlobalVideogame

FOR EACH ROW

EXECUTE FUNCTION esquema_global.insert_global_videogame();

CREATE TRIGGER trg_update_global_videogame

INSTEAD OF UPDATE ON esquema_global.GlobalVideogame

FOR EACH ROW EXECUTE FUNCTION esquema_global.update_global_videogame();

CREATE TRIGGER trg_delete_global_videogame

INSTEAD OF DELETE ON esquema_global.GlobalVideogame

FOR EACH ROW EXECUTE FUNCTION esquema_global.delete_global_videogame();



-- Trigger para GlobalPlatform

CREATE TRIGGER trg_insert_global_platform

INSTEAD OF INSERT ON esquema_global.GlobalPlatform

FOR EACH ROW
```

```
EXECUTE FUNCTION esquema_global.insert_global_platform();

CREATE TRIGGER trg_update_global_platform

INSTEAD OF UPDATE ON esquema_global.GlobalPlatform

FOR EACH ROW EXECUTE FUNCTION esquema_global.update_global_platform();

CREATE TRIGGER trg_delete_global_platform

INSTEAD OF DELETE ON esquema_global.GlobalPlatform

FOR EACH ROW EXECUTE FUNCTION esquema_global.delete_global_platform();


-- Trigger para GlobalUser

CREATE TRIGGER trg_insert_global_user

INSTEAD OF INSERT ON esquema_global.GlobalUser

FOR EACH ROW

EXECUTE FUNCTION esquema_global.insert_global_user();

CREATE TRIGGER trg_update_global_user

INSTEAD OF UPDATE ON esquema_global.GlobalUser

FOR EACH ROW EXECUTE FUNCTION esquema_global.update_global_user();

CREATE TRIGGER trg_delete_global_user

INSTEAD OF DELETE ON esquema_global.GlobalUser

FOR EACH ROW EXECUTE FUNCTION esquema_global.delete_global_user();


-- Trigger para GlobalOwnership

CREATE TRIGGER trg_insert_global_ownership

INSTEAD OF INSERT ON esquema_global.GlobalOwnership

FOR EACH ROW

EXECUTE FUNCTION esquema_global.insert_global_ownership();
```

```
CREATE TRIGGER trg_update_global_ownership

INSTEAD OF UPDATE ON esquema_global.GlobalOwnership

FOR EACH ROW EXECUTE FUNCTION esquema_global.update_global_ownership();

CREATE TRIGGER trg_delete_global_ownership

INSTEAD OF DELETE ON esquema_global.GlobalOwnership

FOR EACH ROW EXECUTE FUNCTION esquema_global.delete_global_ownership();



-- ========================

-- Pruebas de inserciones, modificaciones y eliminaciones



--GlobalCompany

-- INSERT

INSERT INTO esquema_global.GlobalCompany (CompanyName, Director,
CreationDate, Country, License, Type)

VALUES ('PhantomSoft', 'Diana Cruz', '2023-08-12', 'España', 'Copyright
2023 PhantomSoft', 'Desarrolladora');



-- UPDATE

UPDATE esquema_global.GlobalCompany

SET Country = 'México', Director = 'Diana C. Rojas'

WHERE CompanyName = 'PhantomSoft';



-- DELETE

DELETE FROM esquema_global.GlobalCompany

WHERE CompanyName = 'PhantomSoft';
```

```sql
--GlobalVideogame

-- INSERT

INSERT INTO esquema_global.GlobalVideogame (Name, Developer, Console,
Price, ReleaseDate, Rating, Genres)

VALUES ('Galaxy Strike', 'Ubisoft', 'PlayStation 4', 39.99,
'2024-01-10', 8, 'Arcade, Sci-Fi');



-- UPDATE

UPDATE esquema_global.GlobalVideogame

SET Price = 29.99, Rating = 9

WHERE Name = 'Galaxy Strike';



-- DELETE

DELETE FROM esquema_global.GlobalVideogame

WHERE Name = 'Galaxy Strike';







--GlobalPlatform

-- INSERT

INSERT INTO esquema_global.GlobalPlatform (Name, ReleaseDate,
SalesVolume, Manufacturer, Generation)

VALUES ('NeoBox', '2023-11-01', 1000000, 'Sony', 'Novena');



-- UPDATE
```

```sql
UPDATE esquema_global.GlobalPlatform

SET SalesVolume = 1500000, Generation = 'Décima'

WHERE Name = 'NeoBox';



-- DELETE

DELETE FROM esquema_global.GlobalPlatform

WHERE Name = 'NeoBox';



--GlobalUser

-- INSERT

INSERT INTO esquema_global.GlobalUser (

   NickName, RegisteredAt, Name, Email, Country, BalanceAmount,
MembershipType, ExpiresAt, AutoRenew, AdsPerSession)

   VALUES ('elena_gamer', '2024-07-10', 'Elena Torres',
'elena@example.com', 'Spain', 100.00,'premium', '2025-07-10', true, 0);



-- UPDATE

UPDATE esquema_global.GlobalUser

SET BalanceAmount = 150.00

WHERE NickName = 'elena_gamer';



-- DELETE

DELETE FROM esquema_global.GlobalUser

WHERE NickName = 'elena_gamer';



--GlobalOwnership
```

```sql
-- INSERT

INSERT INTO esquema_global.GlobalOwnership (NickName, Videogame,
PurchasedAt, TotalPlaytime)

VALUES ('user3', 'FIFA 21', '2024-09-16', 5);



-- UPDATE

UPDATE esquema_global.GlobalOwnership

SET TotalPlaytime = 12

WHERE NickName = 'user3' AND Videogame = 'FIFA 21';



-- DELETE

DELETE FROM esquema_global.GlobalOwnership

WHERE NickName = 'user3' AND Videogame = 'FIFA 21';
```

El **único inconveniente** es que estas operaciones **sólo se aplican sobre los datos provenientes de db2**, ya que los triggers han sido diseñados para ignorar registros cuya fuente sea **db1**. Por tanto, cualquier dato que provenga de **db1** es **solo de lectura** y no puede modificarse desde el esquema global, esto ha sido una decisión por complejidad, si se quisiera modificar y eliminar tablas de la db1 habría que hacer el proceso inverso de convertir los campos de las vistas a los campos de la base de la db1 que son bastante distintos, por ello solo se ha probado en db2 que era más sencillo. En la práctica real además habría que disponer de alguna información extra para poder insertar correctamente un nuevo dato en una base de datos u otra ya que al final esto es un esquema global pero el dato se almacenará en uno u otro esquema local.