1.1. TRANSPARENCIA DE DISTRIBUCIÓN PARA LA INSTRUCCIÓN SELECT.

El siguiente paso a desarrollar es la implementación de transparencia. Para efectos del proyecto, se realizará únicamente transparencia para las instrucciones `select`, `insert` y `delete`.

1.1.1. Creación de sinónimos para fragmentos.

- Crear un archivo por sitio llamado `s-04-netmax-`**`<pdb>`**`-sinonimos.sql`. Ejemplo: `s-04-netmax-`**`jrc-s1`**`-sinonimos.sql`
- Incluir los sinónimos requeridos por PDB los cuales serán empleados para implementar transparencia de localización.
- El script deberá conectarse a la PDB y crear los sinónimos correspondientes.
- Observar en el siguiente ejemplo la convención a emplear: `<nombre_global>_f<n>` Donde N es el número de fragmento. A partir de este punto se oculta la información de ubicación de los fragmentos. Por ejemplo, se oculta `jrc_s1`, `jrc_s2`, etc.
- Observar que también se crean sinónimos para fragmentos locales. Por ejemplo: para el nodo `jrcbd_s1` se crea el sinónimo `usuario_f1` a partir de `usuario_f1_jrc_s1`

1.1.2. Creación de sinónimos para tablas replicadas.

- Para el caso de las tablas que serán replicadas, también deberán contar con sus sinónimos: 3 remotos y uno local. Emplear la notación `<nombre_global>_r<n>`
- En este caso "N" no significa el número de fragmento, representa el número de réplica. Al contar con 4 PDBs, se tendrá un total de 4 réplicas. Por convención la primera réplica será considerada la réplica local. Por ejemplo, para la tabla `tipo_cuenta_r_jrc_s1` sus sinónimos serán `tipo_cuenta_r1`, `tipo_cuenta_r2`, `tipo_cuenta_r3` y `tipo_cuenta_r4`. La primera réplica apuntará a la tabla local `marca_r_jrc_s1` y los otros 3 apuntarán a las réplicas remotas.
- Las copias manuales de tablas no requieren sinónimos.

Ejemplo:

Sinónimos para `jrcbd_s1`.

```
--@Autor:           Jorge A. Rodríguez C
--@Fecha creación:  dd/mm/yyyy
--@Descripción:     Creacion de sinonimos para jrcbd_s1

--usuario
create or replace synonym usuario_f1 for usuario_f1_jrc_s1;
create or replace synonym usuario_f2 for usuario_f2_arc_s1@arcbd_s1.fi.unam;
create or replace synonym usuario_f3 for usuario_f3_jrc_s2@jrcbd_s2.fi.unam;
create or replace synonym usuario f4 for usuario f4 arc s2@arcbd s2.fi.unam;
create or replace synonym usuario_f5 for usuario_f5_arc_s1@arcbd_s1.fi.unam;

--completar
```

1.1.2.1. Validación de sinónimos

- Para verificar que los sinónimos y fragmentos han sido creados correctamente en sus correspondientes sitios, crear un solo script llamado `s-04-netmax-valida-sinonimos.sql` El script mostrará un conteo del número de registros existente principalmente para validar que el sinónimo esté creado correctamente.
- Cabe mencionar que el manejador no verifica si la tabla realmente existe durante la creación del sinónimo, por lo que este script será útil para detectar posibles errores.

Ejemplo:

```
--@Autor:           Jorge A. Rodríguez C
--@Fecha creación:  dd/mm/yyyy
--@Descripción:     Script de validacion de sinonimos
```

```
Prompt validando sinonimos para USUARIO
select
 (select count(*) from USUARIO_F1) as usuario_f1,
 (select count(*) from USUARIO_F2) as usuario_f2,
 (select count(*) from USUARIO_F3) as usuario_f3,
 (select count(*) from USUARIO F4) as usuario f4,
 (select count(*) from USUARIO_F5) as usuario_f5
from dual;

Prompt validando sinonimos para PLAYLIST

-- completar
```

1.1.2.2. Ejecución de script para sinónimos.

- Crear un solo script llamado `s-04-netmax-main-sinonimos.sql` El script se deberá conectar a cada PDB y ejecutar los scripts creados anteriormente. Notar que el script de validación se ejecuta en cada una de las PDBs para confirmar su correcta creación.

```
--@Autor:          Jorge A. Rodríguez C
--@Fecha creación:
--@Descripción:     Creación de  sinónimos - main
clear screen
whenever sqlerror exit rollback;

prompt ====================================
prompt Creando sinonimos para jrcbd_s1
prompt ====================================
connect netmax_bdd/netmax_bdd@jrcbd_s1
@s-04-netmax-jrc-s1-sinonimos.sql
@s-04-netmax-valida-sinonimos.sql

prompt ====================================
prompt creando sinonimos para jrcbd s2
prompt ====================================
connect netmax_bdd/netmax_bdd@jrcbd_s2
@s-04-netmax-jrc-s2-sinonimos.sql
@s-04-netmax-valida-sinonimos.sql

prompt ====================================
prompt creando sinonimos para arcbd_s1
prompt ====================================
connect netmax_bdd/netmax_bdd@arcbd_s1
@s-04-netmax-arc-s1-sinonimos.sql
@s-04-netmax-valida-sinonimos.sql

prompt ====================================
prompt creando sinonimos para arcbd_s2
prompt ====================================
connect netmax_bdd/netmax_bdd@arcbd_s2
@s-04-netmax-arc-s2-sinonimos.sql
@s-04-netmax-valida-sinonimos.sql

prompt Listo!
```

1.1.3. Creación de vistas.

1.1.3.1. Vistas para tablas globales sin columnas BLOB/CLOB

- Crear un solo script llamado `s-05-netmax-vistas.sql`. El script contendrá la definición de las vistas que se van a crear en cada sitio.
- **No incluir** en este script las vistas que correspondan a tablas con datos BLOB/CLOB. Esto **sin importar** el sitio en el que se encuentre. Las vistas para estas tablas se crearán en la siguiente sección.

Ejemplo:

```
--@Autor:          Jorge A. Rodríguez C
--@Fecha creación:
--@Descripción:     Creación de vistas comunes a todos los nodos
--                  Se excluyen las vistas que contienen columnas BLOB
```

```
--PLAYLIST
Prompt creando vista PLAYLIST
create or replace view PLAYLIST as
 select playlist_id,calificacion,indice,num_reproducciones,programa_id,
  usuario id
 from playlist_f1
 union all
 select playlist_id,calificacion,indice,num_reproducciones,programa_id,
  usuario_id
 from playlist_f2
 union all
 select playlist id,calificacion,indice,num reproducciones,programa id,
  usuario_id
  from playlist_f3
 union all
 select playlist id,calificacion,indice,num reproducciones,programa id,
  usuario_id from playlist_f4;

--completar
```

- Observar que se listan los nombres de cada columna. Evitar el uso de "*" ya que el orden en el que se definen los atributos en cada fragmento puede cambiar y generar inconsistencias y/o errores. Hacer uso de `union all`.
- Observar que este código se puede ejecutar en las 4 PDBs ya que el código es exactamente el mismo para todos los sitios.

1.1.3.2. Vistas para tablas replicadas.

- Para implementar el mecanismo de replicación es necesario crear una vista para cada una de las tablas que serán replicadas.
- Para una tabla que cuenta con su propio esquema de fragmentación, su vista define una consulta que permite recuperar todos los datos (expresión de reconstrucción). Sin embargo, para una tabla replicada que no cuenta con esquema de fragmentación, la vista no incluye la expresión de reconstrucción.
- Debido a que en todos los sitios existirá exactamente la misma copia de datos (replica), la vista únicamente deberá mostrar los datos de alguna de estas réplicas. La técnica más eficiente es apuntar a la copia local que anteriormente se estableció con el sinónimo `<nombre_tabla>_r1`. Es decir, la réplica r1 apunta a la réplica local.
- Por lo anterior, en cada PDB se deberá crear una vista que apunte a su réplica local.

Con base a lo anterior, agregar la definición de las vistas para cada tabla replicada en el archivo `s-05-netmax-vistas.sql`.

Ejemplo:

```
create or replace view tipo_serie as
 select tipo_serie_id,clave,descripcion
 from tipo_serie_r1;
```

1.1.3.3. Creación de tablas temporales para acceso a datos BLOB/CLOB

- En prácticas anteriores se revisaron 2 técnicas para implementar transparencia de distribución para acceder a un dato BLOB/CLOB de forma remota. En este proyecto se empleará únicamente la estrategia 2 en la que se obtiene un dato binario a través de su Id. Para ello, se deberán crear los siguientes objetos adicionales:
  - Tabla temporal para almacenar el dato BLOB/CLOB para transparencia de operaciones `select`. Se empleará la notación **ts**_<nombre_global>_<num_fragmento>. Por ejemplo, `ts_archivo_programa_1` es una tabla temporal que se empleará para almacenar el dato BLOB que se obtiene del fragmento 1
  - Tabla temporal para almacenar el dato BLOB/CLOB para transparencia de operaciones `insert`. Se empleará la notación **ti**_<nombre_global>_<num_fragmento>.
  - Funciones encargadas de obtener un dato BLOB/CLOB de un sitio remoto. Se empleará la sintaxis `get_remote_<nombre_columna_blob>_f<numero_fragmento>_by_id`.
    Por ejemplo, la función `get_remote_trailer_f1_by_id` será una función que obtiene el contenido del dato BLOB asociado a la columna `trailer` del fragmento 1 (`documental_f1`).
- Notar que no se requiere crear otros objetos que fueron empleados en prácticas anteriores (objetos `type, table`) ya que se estará empleando la estrategia 2.
- Observar que estas tablas temporales y funciones son accedidas por prácticamente todos los nodos. Algunos nodos pudieran no requerirlas ya que el dato BLOB/CLOB se encontrará localmente. Lo ideal sería crear estas funciones únicamente en los sitios donde se requieren, pero para evitar duplicidad de código y aumento de la complejidad de los scripts, se creará uno solo y será ejecutado en todos los sitios.

Para implementar estos objetos, realizar las siguientes acciones:

- Crear un solo script llamado `s-05-netmax-tablas-temporales.sql` . El script deberá contener todas las tablas temporales que requieran manejo de datos CLOB/BLOB tanto para operaciones `insert` como para operaciones `select`. El código es exactamente el mismo, solo varían por el nombre, pero se usarán para propósitos diferentes.

- Observar en el siguiente ejemplo la creación de un bloque PL/SQL anónimo que se encarga de eliminar las tablas temporales en caso de existir. Esto con la finalidad de poder ejecutar el script N veces sin la necesidad de eliminar las tablas previamente creada de forma manual.

Ejemplo:

```
--@Autor:          Jorge A. Rodriguez C
--@Fecha creación:  dd/mm/yyyy
--@Descripción:    Definición de tablas temporales para manejo de datos BLOB

Prompt eliminando tablas en caso de existir
declare
  cursor cur_tablas is
    select table_name
    from user tables
    where table_name in ('TS_DOCUMENTAL_1','TS_DOCUMENTAL_2','TS_DOCUMENTAL_3',
     'TI DOCUMENTAL 1','TI DOCUMENTAL 2','TI DOCUMENTAL 3',
     'TS_ARCHIVO_PROGRAMA_1','TS_ARCHIVO_PROGRAMA_2',
     'TI_ARCHIVO_PROGRAMA_1','TI_ARCHIVO_PROGRAMA_2');
begin
  for r in cur tablas loop
    execute immediate 'drop table ' ||r.table_name;
  end loop;
end;
/

Prompt tablas temporales para transparencia - Select

-- Tablas temporales para DOCUMENTAL
create global temporary table ts_documental_1 (
 programa_id number(10,0) constraint ts_documental_1_pk  primary key,
 trailer blob not null
) on commit preserve rows;

create global temporary table ts_documental_2 (
 programa_id number(10,0) constraint ts_documental_2_pk  primary key,
 trailer blob not null
) on commit preserve rows;

create global temporary table ts_documental_3 (
 programa_id number(10,0) constraint ts_documental_3_pk  primary key,
 trailer blob not null
) on commit preserve rows;

--tablas temporales para ARCHIVO_PROGRAMA

create global temporary table ts_archivo_programa_1 (
 num_archivo number(5,0),
 programa_id number(10,0),
 archivo blob not null,
 constraint ts archivo programa 1 pk primary key(num archivo,programa id)
) on commit preserve rows;

create global temporary table ts_archivo_programa_2 (
 num archivo number(5,0),
 programa_id number(10,0),
 archivo blob not null,
 constraint ts_archivo_programa_2_pk primary key(num_archivo,programa_id)
) on commit preserve rows;

Prompt Prompt tablas temporales para transparencia - Insert

-- Tablas temporales para DOCUMENTAL
create global temporary table ti_documental_1 (
 programa_id number(10,0) constraint ti_documental_1_pk  primary key,
 tematica varchar2(100) not null,
 duracion numeric(5,2) not null,
 pais_id numeric(2,0) not null
 trailer blob not null
) on commit preserve rows;

--completar
```

- Observar que, para la tabla `documental`, se crean 6 tablas temporales: Las primeras 3 se crean para implementar transparencia con la instrucción `select` (una por fragmento), y las otras 3, para implementar transparencia con la instrucción `insert`.
- Observar que las tablas temporales empleadas para implementar transparencia para operaciones `select` solo requieren incluir la PK y la columna CLOB/BLOB mientras que las tablas temporales requeridas para implementar transparencia de operaciones `insert` incluyen todas las columnas.

1.1.3.1. Creación de funciones para acceso a datos CLOB/BLOB

- Crear un solo script llamado `s-05-netmax-funciones-blob.sql` El script contendrá la definición de las funciones requeridas para realizar acceso remoto a los datos CLOB/BLOB. Se requiere generar una función que extraiga el dato binario de cada fragmento.

Ejemplo:

```
--@Autor:          Jorge A. Rodríguez C
--@Fecha creación: dd/mm/yyyy
--@Descripción:    Definición de funciones para acceso a BLOBs


Prompt funciones  para acceso de blobs - DOCUMENTAL

--Funcion que obtiene BLOB del fragmento 1
create or replace function get_remote_trailer_f1_by_id(v_id in number)
 return blob is
 pragma autonomous transaction;
 v_temp_trailer blob;
begin
 --asegura que no haya registros
 delete from ts_documental_1;
 --inserta los datos obtenidos del fragmento remoto a la tabla temporal.
 insert into ts_documental_1 select programa_id,trailer
  from documental_f1 where programa_id = v_id;
 --obtiene el registro de la tabla temporal y lo regresa como blob
 select trailer into v_temp_trailer from ts_documental_1 where programa_id = v_id;
 --elimina los registros de la tabla temporal una vez que han sido obtenidos.
 delete from ts_documental_1;
 commit;
  return v temp trailer;
 exception
  when others then
   rollback;
   raise;
end;
/
show errors
--completar para los demás fragmentos y tablas.
```

- Observar que la función obtiene el dato binario del fragmento 1, y se emplea la tabla temporal `ts_documental_1`. Esta función será empleada por todos aquellos sitios que requieran acceder de forma remota al fragmento 1
- Se deberán crear otras 2 funciones para los fragmentos 2 y 3. Es decir, las funciones `get_remote_trailer_f2_by_id` y `get_remote_trailer_f3_by_id`
- Aplicar la misma técnica para las demás tablas globales que contentan datos binarios.

1.1.3.2. Creación de vistas con datos CLOB/BLOB

- Crear un script por cada PDB llamado `s-05-netmax-<pdb>-vistas-blob.sql` Cada script contendrá la definición de las vistas que contienen columnas con datos CLOB/BLOB. El script deberá hacer uso de las funciones creadas en la sección anterior únicamente cuando sea necesario hacer un acceso remoto para obtener un dato CLOB/BLOB.

Ejemplo:

```
--@Autor:          Jorge A. Rodríguez C
--@Fecha creación:
--@Descripción:     Definición de vistas con columnas BLOB para jrcbd_s1
--
```

```
--DOCUMENTAL
Prompt creando vista DOCUMENTAL
create or replace view DOCUMENTAL as
 select programa_id,tematica,duracion,trailer,pais_id
 from documental_f1
 union all
 select programa_id,tematica,duracion,
  get_remote_trailer_f2_by_id(programa_id),pais_id
 from documental_f2
 union all
 select programa_id,tematica,duracion,
  get_remote_trailer_f3_by_id(programa_id),pais_id
 from documental_f3;


--ARCHIVO PROGRAMA
Prompt creando vista ARCHIVO_PROGRAMA
create or replace view ARCHIVO_PROGRAMA as
 select num_archivo,programa_id,
  get_remote_archivo_f1_by_id(num_archivo,programa_id) as archivo,
  tamanio from archivo programa f1
 union all
 select num_archivo,programa_id,
  get_remote_archivo_f2_by_id(num_archivo,programa_id),
  tamanio from archivo_programa_f2;
```

- Observar que en el caso de la vista `documental`, se requiere un acceso local al fragmento 1 y un acceso remoto al fragmento 2 para obtener el dato binario, por lo tanto se hace uso de la función `get_remote_trailer_f2_by_id`.

1.1.3.3. Ejecución de scripts de creación de vistas

Generar un archivo llamado `s-05-netmax-main-vistas.sql`. El script deberá conectarse a cada PDB y ejecutar los scripts anteriores.

Ejemplo:

```
--@Autor:          Jorge A. Rodríguez C
--@Fecha creación:
--@Descripción:     Creación de  vistas para todos los sitios

clear screen
whenever sqlerror exit rollback;

prompt ========================================
prompt Creando vistas para jrcbd_s1
prompt ========================================

connect netmax_bdd/netmax_bdd@jrcbd_s1
@s-05-netmax-vistas.sql
@s-05-netmax-tablas-temporales.sql
@s-05-netmax-funciones-blob.sql
@s-05-netmax-jrc-s1-vistas-blob.sql

prompt ========================================
prompt Creando vistas para jrcbd_s2
prompt ========================================

prompt Creando vistas para jrcbd_s2
connect netmax_bdd/netmax_bdd@jrcbd_s2
@s-05-netmax-vistas.sql
@s-05-netmax-tablas-temporales.sql
@s-05-netmax-funciones-blob.sql
@s-05-netmax-jrc-s2-vistas-blob.sql

prompt ========================================
prompt Creando vistas para arcbd_s1
prompt ========================================

prompt Creando vistas para arcbd s1
connect netmax_bdd/netmax_bdd@arcbd_s1
@s-05-netmax-vistas.sql
@s-05-netmax-tablas-temporales.sql
@s-05-netmax-funciones-blob.sql
@s-05-netmax-arc-s1-vistas-blob.sql
```

```
prompt ========================================
prompt Creando vistas para arcbd s2
prompt ========================================



prompt Creando vistas para arcbd_s2
connect netmax_bdd/netmax_bdd@arcbd_s2
@s-05-netmax-vistas.sql
@s-05-netmax-tablas-temporales.sql
@s-05-netmax-funciones-blob.sql
@s-05-netmax-arc-s2-vistas-blob.sql

prompt Listo!
```

- Observar que se ejecutan los mismos scripts en todos los sitios excepto el último (marcado en negritas). Este último script contiene la definición de las vistas que contienen datos CLOB/BLOB y su definición es diferente para cada PDB.

1.2. TRANSPARENCIA DE DISTRIBUCIÓN PARA INSTRUCCIONES DML.

- El alcance del proyecto se limita a transparencia para instrucciones `insert` y `delete`
- Para implementar transparencia para instrucciones `insert`, se deberán crear `instead of triggers` para cada tabla fragmentada.
- Solo se deberá implementar los eventos de inserción y eliminación. El trigger deberá enviar un error cuando se intente realizar una operación `update`, indicando que estas aún no están implementadas.

Similar a la técnica vista en prácticas anteriores, se hará uso de triggers tipo `instead of` para implementar este nivel de trasparencia.

1.2.1. Creación de triggers para tablas con esquema de fragmentación.

- En algunos casos, el código del trigger es distinto para cada sitio, por ejemplo, en fragmentaciones horizontales derivadas, o en tablas con datos CLOB/BLOB. En otros casos el código es el mismo. Con la finalidad de evitar duplicidad de código, realizar lo siguiente:
  - Si el código es diferente para cada sitio, crear un archivo llamado `s-06-netmax-trigger-<pdb>-<tabla>.sql` para cada trigger
  - Si el código es el mismo, crear un archivo llamado `s-06-netmax-trigger-<tabla>.sql`

- Los triggers deberán lanzar una excepción cuando ocurra cada una de las siguientes situaciones:

Código de error	Descripción del error.
-20010	El registro que se intenta insertar o eliminar no cumple con el esquema de fragmentación horizontal primaria. Por ejemplo, suponer que los posibles valores para decidir el fragmento donde se aplicará la operación DML son A y B. SI el valor del campo es diferente a los 2 valores permitidos, el trigger deberá lanzar un error empleando este código.
-20020	El registro que se intenta insertar o eliminar no cumple con el esquema de fragmentación horizontal derivada. Este caso aplica cuando el trigger intenta ubicar al registro en el fragmento padre. Por ejemplo. Suponer que se desea insertar un registro de la tabla global `sucursal` que tiene una llave foránea `país_id = 1`. EL trigger intentará ubicar el fragmento donde se encuentra el registro padre cuya llave primaria tenga el valor `país_id = 1`. Si el registro no es localizado en algún nodo, el trigger deberá lanzar un error empleado este código.
-20030	Se intentó realizar una operación `update` en una tabla con esquema de fragmentación. Para propósitos del proyecto, esta operación no estará implementada. El trigger deberá lanzar un error indicando que operaciones `update` aún no se han implementado.

1.2.2. Creación de triggers para tablas replicadas.

- Cada una de las vistas que representan a las tablas replicadas también deberán contar con su trigger de tipo `instead of`.
- A diferencia de las tablas con esquema de fragmentación, el trigger de cada tabla replicada si deberá implementar las 3 operaciones DML : `insert`, `update` y `delete`.
- En este caso el código del trigger no requiere lanzar ninguna de las 3 excepciones mencionadas anteriormente. La única excepción que puede generarse es la siguiente:

Código de error	Descripción del error.
-20040	Esta excepción se deberá lanzar cuando el código del trigger detecte que el registro no fue creado, actualizado o eliminado por alguna razón. Recordar que la BD puede regresar que se han afectado 0 registros, lo cual sería incorrecto ya que el trigger se dispara cuando se realiza una operación DML. Para los 3 tipos de operaciones, siempre se debe afectar un registro por réplica. En total 4 registros.

	La forma más simple para validar este requerimiento es empleando la instrucción `sql%rowcount` que indica el número de registros afectados de la última operación DML ejecutada. Ver el código de ejemplo más adelante.

1.2.2.1. Replicación síncrona de tablas.

La técnica que se empleará para replicar las operaciones DML se le conoce como "replicación Síncrona". Esta estrategia tiene las siguientes ventajas y desventajas:

Ventajas:

- Existe integridad y consistencia de datos en todo instante sin importar el número de réplicas.
- Al lanzar una operación DML sobre una tabla replicada, el trigger deberá ejecutar dicha instrucción en las 4 réplicas incluyendo la réplica local. Esto permite garantizar que las 4 réplicas conserven el mismo estado consistente en todo momento.
- En caso de ocurrir un error durante la aplicación de las operaciones DML entre las réplicas, se producirá una excepción la cual podrá ser manejada y en su caso generar un `rollback`. Esto garantizará regresar a un estado consistente de los datos.
- Fácil de implementar a través de un trigger.

Desventajas:

- Cada operación DML requiere a su vez la aplicación de 4 operaciones. En el escenario donde existe una gran cantidad de operaciones DML sobre la tabla, el desempeño puede verse afectado.
- Si uno de los nodos presenta una falla o no está disponible, la operación DML no podrá ejecutarse, lo que afectaría directamente a la disponibilidad de la base de datos.

El esquema anterior es adecuado en situaciones donde existen pocas operaciones DML (baja frecuencia), y en situaciones donde se requiere integridad y consistencia en las N réplicas todo el tiempo. Estos requerimientos son los que se desean en este proyecto, por lo que se ha decidido emplearla. Existen otras técnicas, por ejemplo, sincronización asíncrona en la que es posible considerar exitosa una operación DML sin importar que en algunas de las réplicas no se ha realizado la operación. Las operaciones pendientes se programan para ser ejecutada tiempo después. Oracle Golden Gate es un producto que permite realizar este tipo de esquemas de replicación.

De forma similar a las tablas con replicación, crear scripts `s-06-netmax-<tabla>-trigger.sql` para implementar los trigger instead of. Observar que para este tipo de tablas el código es exactamente el mismo en todos los nodos.

Ejemplo:

```
--@Autor:          Jorge A. Rodríguez C
--@Fecha creación:
--@Descripción:    trigger  pais

create or replace trigger t_dml_pais
  instead of insert or update or delete on pais
declare
  v_count number;
begin
  case
  when inserting then
    v count := 0;
    --replica local
    insert into pais r1(pais id,clave,nombre,continente)
    values (:new.pais_id,:new.clave,:new.nombre,:new.continente);
    v_count := v_count + sql%rowcount;
    --replica 2
    insert into pais r2(pais id,clave,nombre,continente)
    values (:new.pais id,:new.clave,:new.nombre,:new.continente);
    v count := v count + sql%rowcount;
    --replica 3
    insert into pais_r3(pais_id,clave,nombre,continente)
    values (:new.pais_id,:new.clave,:new.nombre,:new.continente);
    v count := v count + sql%rowcount;
    --replica 4
    insert into pais r4(pais id,clave,nombre,continente)
    values (:new.pais_id,:new.clave,:new.nombre,:new.continente);
    v_count := v_count + sql%rowcount;

    if v count <> 4 then
      raise_application_error(-20040,
        'Número incorrecto de registros insertados en tabla replicada: '
        ||v_count);
    end if;

  when deleting then
```

```
    v_count := 0;
    --replica local
    delete from pais_r1 where pais_id = :old.pais_id;
    v_count := v_count + sql%rowcount;
    --replica 2
    delete from pais_r2 where pais_id = :old.pais_id;
    v_count := v_count + sql%rowcount;
    --replica 3
    delete from pais_r3 where pais_id = :old.pais_id;
    v_count := v_count + sql%rowcount;
    --replica 4
    delete from pais r4 where pais id = :old.pais id;
    v_count := v_count + sql%rowcount;

    if v_count <> 4 then
      raise_application_error(-20040,
        'Número incorrecto de registros eliminados en tabla replicada: '
        ||v count);
    end if;

  when updating then
    --replica local
    v_count := 0;
    update pais r1 set clave = :new.clave,nombre =:new.nombre,
      continente =:new.continente
    where pais id = :new.pais id;
    v_count := v_count + sql%rowcount;
    --replica 2
    update pais_r2 set clave = :new.clave,nombre =:new.nombre,
      continente =:new.continente
    where pais id = :new.pais id;
    v_count := v_count + sql%rowcount;
    --replica 3
    update pais_r3 set clave = :new.clave,nombre =:new.nombre,
      continente =:new.continente
    where pais_id = :new.pais_id;
    v count := v count + sql%rowcount;
    --replica 4
    update pais_r4 set clave = :new.clave,nombre =:new.nombre,
      continente =:new.continente
    where pais id = :new.pais id;
    v_count := v_count + sql%rowcount;
  end case;

  if v_count <> 4 then
    raise_application_error(-20040,
      'Número incorrecto de registros actualizados en tabla replicada: '
      ||v_count);
  end if;

end;
/
show errors
```

- Notar la lógica para validar la excepción con código -20040.
- Al terminar de crear los scripts anteriores, invocarlos en uno nuevo llamado `s-06-netmax-main-triggers.sql`. Tener cuidado con invocar cada script con base al sitio donde deben ejecutarse.

Ejemplo:

```
--@Autor:          Jorge A. Rodríguez C
--@Fecha creación:
--@Descripción:     Script principal - creación de triggers
clear screen
whenever sqlerror exit rollback;

prompt ====================================
prompt Creando triggers para jrcbd s1
prompt ====================================
connect netmax bdd/netmax bdd@jrcbd s1
@s-06-netmax-trigger-usuario.sql
@s-06-netmax-trigger-programa.sql
@s-06-netmax-trigger-jrc-s1-serie.sql
```

```
@s-06-netmax-trigger-jrc-s1-pelicula.sql
@s-06-netmax-trigger-jrc-s1-documental.sql
@s-06-netmax-trigger-jrc-s1-archivo-programa.sql
@s-06-netmax-trigger-jrc-s1-playlist.sql
@s-06-netmax-trigger-tipo-cuenta.sql
@s-06-netmax-trigger-tipo-serie.sql
@s-06-netmax-trigger-pais.sql



prompt ====================================
prompt Creando triggers para jrcbd_s2
prompt ====================================
connect netmax_bdd/netmax_bdd@jrcbd_s2
@s-06-netmax-trigger-usuario.sql
@s-06-netmax-trigger-programa.sql
@s-06-netmax-trigger-jrc-s2-serie.sql
@s-06-netmax-trigger-jrc-s2-pelicula.sql
@s-06-netmax-trigger-jrc-s2-documental.sql
@s-06-netmax-trigger-jrc-s2-archivo-programa.sql
@s-06-netmax-trigger-jrc-s2-playlist.sql
@s-06-netmax-trigger-tipo-cuenta.sql
@s-06-netmax-trigger-tipo-serie.sql
@s-06-netmax-trigger-pais.sql


prompt ====================================
prompt Creando triggers para arcbd_s1
prompt ====================================
connect netmax_bdd/netmax_bdd@arcbd_s1
@s-06-netmax-trigger-usuario.sql
@s-06-netmax-trigger-programa.sql
@s-06-netmax-trigger-arc-s1-serie.sql
@s-06-netmax-trigger-arc-s1-pelicula.sql
@s-06-netmax-trigger-arc-s1-documental.sql
@s-06-netmax-trigger-arc-s1-archivo-programa.sql
@s-06-netmax-trigger-arc-s1-playlist.sql
@s-06-netmax-trigger-tipo-cuenta.sql
@s-06-netmax-trigger-tipo-serie.sql
@s-06-netmax-trigger-pais.sql


prompt ====================================
prompt Creando triggers para arcbd_s2
prompt ====================================
connect netmax bdd/netmax bdd@arcbd s2
@s-06-netmax-trigger-usuario.sql
@s-06-netmax-trigger-programa.sql
@s-06-netmax-trigger-arc-s2-serie.sql
@s-06-netmax-trigger-arc-s2-pelicula.sql
@s-06-netmax-trigger-arc-s2-documental.sql
@s-06-netmax-trigger-arc-s2-archivo-programa.sql
@s-06-netmax-trigger-arc-s2-playlist.sql
@s-06-netmax-trigger-tipo-cuenta.sql
@s-06-netmax-trigger-tipo-serie.sql
@s-06-netmax-trigger-pais.sql

prompt Listo!
```

- Observar que solo los triggers para usuario y programa son los únicos cuyo código se puede reutilizar debido a que su fragmentación es horizontal primaria.

Continuar con la última parte del Proyecto.