1. TRANSPARENCIA DE DISTRIBUCIÓN PARA INSTRUCCIONES DML

1.1. Estrategia general para implementar transparencia para operaciones DML

En ejercicios prácticos anteriores se realizó la carga inicial de datos a partir del uso de un script que se encarga de asignar registros a su fragmento correspondiente. La desventaja de esta solución es que las sentencias `insert` y en general las operaciones DML básicas carecen de niveles de transparencia. El usuario debería poder invocar la inserción de un nuevo registro de forma idéntica a una base de datos centralizada. Para poder implementar transparencia de distribución se puede emplear la siguiente estrategia:
Interceptar la instrucción DML empleando un ***trigger***.

El trigger deberá contener el código necesario para analizar los datos de entrada y aplicar la operación en los diferentes sitios con base a los valores proporcionados y al esquema de fragmentación para realizar una correcta distribución de los datos. Debido a que se requiere ejecutar más de una instrucción DML, es necesario realizar el manejo adecuado de las transacciones (transparencia de transacciones).

1.1.1. Creación del trigger

El uso de un trigger permite ocultar la lógica requerida para distribuir los datos y enviarlos al sitio correspondiente con base al esquema de fragmentación. Sin embargo, existe un inconveniente. Suponer que el usuario final ejecuta la siguiente sentencia:

```
insert into suscriptor (...) values (...);
```

Suponer que los valores del nuevo registro deben enviarse al sitio remoto en lugar de insertarse localmente. El código del trigger verifica esta condición con base a las reglas de fragmentación e invoca una inserción al sitio remoto. La inserción remota se realiza con éxito, y la ejecución del trigger continua. Al terminar, el registro se va a insertar también en el sitio local lo cual es incorrecto.
El comportamiento anterior se debe a que el trigger solo realiza una acción previa o posterior a la inserción local, pero no impide que el registro se inserte en este caso en la tabla local (a menos que se provoque un error). Para solucionar el comportamiento anterior, la mayoría de los manejadores ofrecen el concepto de ***Instead of DML Trigger***.

Este tipo de trigger se asocia por lo general a una vista que no tiene capacidades o permisos de realizar operaciones DML sobre las tablas con las cuales se asocia. Cabe mencionar que es posible realizar operaciones DML sobre una vista. En esta situación, al ejecutar una sentencia DML sobre una vista que no tiene capacidades de ejecutar operaciones DML, la base de datos dispara la ejecución del trigger sin afectar a la vista ni las tablas asociadas, que es justamente el comportamiento que se desea para implementar transparencia para operaciones DML en una BDD:

Se requiere interceptar la operación DML, pero se deberá omitir la ejecución de la operación local. El trigger será el encargado de realizar esta operación. La sintaxis de un Trigger tradicional es:

```
create or replace trigger <trigger_name>
  {before | after} {delete | insert | update} of <column_name1, .., > on
  <table_name>
 [for each row]
 [ declare ]
  [<nombre_variable> <tipo_de_dato>[:= <valor_inicial >]]
begin
  <instrucciones pl - sql >
end;
```

Para el caso de un Instead of trigger, la sintaxis cambia:

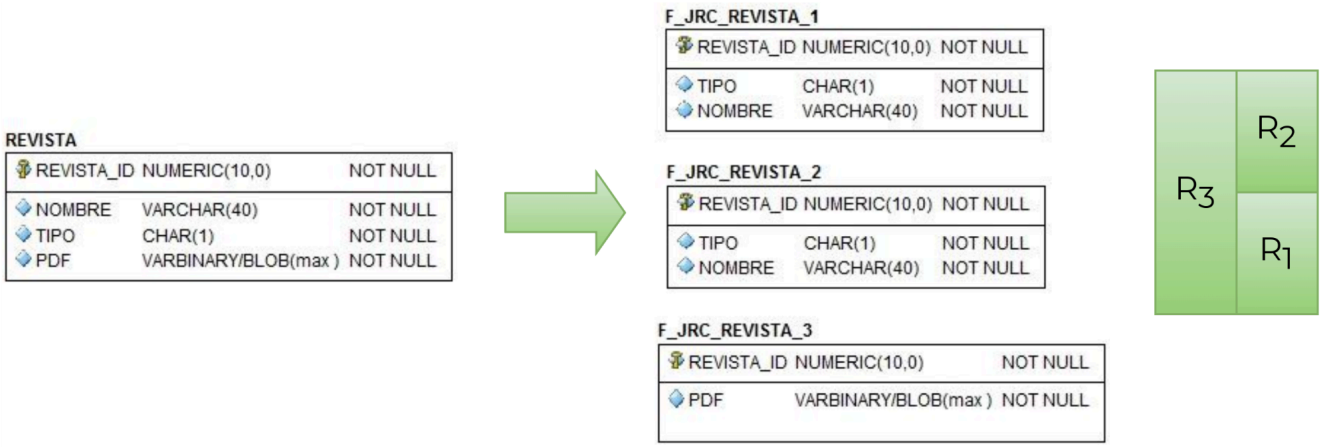
```
create or replace trigger <trigger_name>
  instead of {delete | insert | update} [or {delete | insert | update}]...
  on <table_name>
  [ declare ]
    [<nombre_variable> <tipo_de_dato>[:= <valor_inicial >]]
  begin
    < instrucciones pl - sql >
  end;
```

- Observar que este tipo de triggers no aceptan las cláusulas `before` y `after` debido a que no se realiza acción alguna sobre la tabla o vista a la cual está asociado.
- Un instead Of trigger siempre se ejecuta para cada registro por lo que la cláusula `for each row` no es necesaria.
- El trigger puede leer los valores de las variables `:new` y `:old` pero no puede actualizarlas.
- Se puede crear un solo trigger para interceptar operaciones `insert`, `update` y `delete`.

1.2. Implementación de transparencia con instead of triggers

En esta sección se ilustra el proceso de implementación de transparencia de distribución para el ejemplo de la tabla `revista` revisado en el ejercicio práctico anterior:

$$R_1 = \pi_{revista-id,tipo,nombre}(\sigma_{tipo='A'}(R))$$
$$R_2 = \pi_{revista-id,tipo,nombre}(\sigma_{tipo='B'}(R))$$
$$R_3 = \pi_{revista-id,pdf}(R)$$



- R2 y R3 están en el sitio 2, y R1 en el sitio 1
- Por cada una de las vistas creadas en cada uno de los sitios se deberá crear un Trigger Instead Of.
- En este caso se crearán 2 triggers, cada uno asociado a la vista `revista` creada en el ejercicio práctico anterior.

1.2.1. Estructura general del trigger

```
create or replace trigger t_dml_revista
instead of insert or update or delete on revista
  declare
    --programar
  begin
    case
      when inserting then
    --programar
      when updating then
    --programar
      when deleting then
    --programar
    end case;
  end;
/
```

- El trigger del código anterior intercepta operaciones `insert`, `update` y `delete`.
- Observar el uso de una sentencia de control `case`. Dependiendo la operación DML será el bloque de código a ejecutar.
- El código para implementar transparencia de inserción se muestra en las siguientes secciones.

1.2.2. Transparencia para operaciones **insert** - sitio 1.

```
create or replace trigger t_dml_revista
  instead of insert or update or delete on revista
declare
begin
   case
     when inserting then
      if :new.tipo = 'A' then
        insert into revista_1(revista_id,tipo,nombre)
        values(:new.revista_id,:new.tipo,:new.nombre);
      elsif :new.tipo = 'B' then
        insert into revista_2(revista_id,tipo,nombre)
        values(:new.revista_id,:new.tipo,:new.nombre);
      else
        raise_application_error(20001,
            'Valor incorrecto para el campo tipo : '
          || :new.tipo
          || ' Solo se permiten los valores A , B ');
      end if;
      --inserta el binario, uso de una tabla temporal
      insert into t_insert_revista_3(revista_id,pdf)
      values(:new.revista_id,:new.pdf);
      --inserta en el sitio remoto a través de la tabla temporal
      insert into revista_3
        select * from t_insert_revista_3
        where revista_id = :new.revista_id;
      delete from t_insert_revista_3
      where revista_id = :new.revista_id;

    when updating then
       -- continua..
```

- Observar que para este caso no se requirió declaración de variables.
- Observar la lógica `if - else` en la que se decide el sitio en el que se deberá insertar el nuevo registro.
- Se emplean las variables `:new.<campo>` para hacer referencia a los valores del nuevo registro.
- Observar el código que lanza una excepción en caso que el criterio de fragmentación sea incorrecto.
- Observar la estrategia para insertar el dato `blob`. Se hace uso de una tabla temporal `t_insert_revista_3`. La idea es similar a la realizada en el ejercicio práctico anterior. La tabla es necesaria debido a que se requiere un acceso remoto sitio 1 → Sitio 2 donde se insertarán todos los datos `blob`. Primero se inserta en la temporal, después

se hace la inserción en el sitio 2 (copia) y finalmente se elimina el registro de la temporal (recordar que esta técnica se requiere por las limitantes del manejador para datos LOB en bases de datos remotas).

- La definición de la tabla temporal es similar a la del ejercicio práctico anterior:

```
--tabla temporal para manejar blobs - transparencia para insert
create global temporary table t_insert_revista_3(
  revista_id number(10,0) constraint t_jrc_insert_revista_3_pk  primary key,
  pdf blob not null
) on commit preserve rows;
```

1.2.3. Transparencia para operaciones insert - sitio 2

```
create or replace trigger t_dml_revista
instead of insert or update or delete on revista
declare
begin
  case
    when inserting then
      if :new.tipo = 'A' then
        insert into revista_1(revista_id,tipo,nombre)
        values(:new.revista_id,:new.tipo,:new.nombre);
      elsif :new.tipo = 'B' then
        insert into revista_2(revista_id,tipo,nombre)
        values(:new.revista_id,:new.tipo,:new.nombre);
      else
        raise_application_error(-20001,
            'Valor incorrecto para el campo tipo : '
          || :new.tipo
          || ' Solo se permiten los valores A , B ');
      end if;
      --inserta el binario de forma directa  al ser local
      insert into revista_3(revista_id,pdf)
      values(:new.revista_id,:new.pdf);

      -- continua..
```

- Para el caso del dato `blob`, observar que en este caso no se requiere el uso de una tabla temporal ya que no se hace acceso remoto, el fragmento `revista_3` se encuentra en el sitio 2.

1.2.4. Transparencia para operaciones delete - sitio 1 y Sitio 2

```
when deleting then
  if :old.tipo ='A' then
    delete from revista_1 where revista_id = :old.revista_id;
  elsif :old.tipo='B' then
    delete from revista_2 where revista_id  = :old.revista_id;
  else
    raise_application_error(-20001,
      'Valor incorrecto para el campo tipo : '
      || :old.tipo
      || ' Solo se permiten los valores A , B ');
  end if;
  --elimina el binario
  delete from revista_3 where revista_id = :old.revista_id;
```

- Observar que, en este caso, el código es el mismo para ambos nodos, ya que no existe diferencia en cuanto a la lógica requerida.
- Notar que se hace uso de la variable `:old` No es posible emplear `:new` ya que en una operación `delete` esta variable tiene valor nulo (revisar el documento `BD/apuntes/tema10.pdf` para mayores detalles).

1.2.5. Transparencia de operaciones update – sitio 1

Representa la estrategia más complicada de las anteriores:

- Existe un caso particular que debe ser validado: ¿Qué sucede si el valor de la columna(s) que se emplearon como criterio para fragmentar cambia de valor provocando que el registro deba ser trasladado a otro sitio? En este ejemplo, si el campo tipo cambia de A →B o B →A, el trigger deberá validar esta condición y aplicar las operaciones necesarias para que el registro se actualice y se ubique en el fragmento que le corresponda.
- Para el caso del dato `blob`, observar que nuevamente se requiere el uso de una tabla temporal `t_update_revista_3`. El Manejador no soporta la actualización remota de datos `blob`. Para ello se emplea la tabla.

```
--tabla temporal para manejar blobs - transparencia para update
create global temporary table t_update_revista_3(
  revista_id number(10,0) constraint t_jrc_update_revista_3_pk  primary key,
  pdf blob not null
) on commit preserve rows;
```

- Observar que su definición es similar a la tabla temporal empleada para realizar transparencia de operaciones `select` e `insert`.

La estrategia para realizar la actualización del dato blob es:

- Se limpia la tabla temporal
- Se accede al nuevo dato empleando `:new.pdf`
- El dato es almacenado en la tabla temporal
- Se lanza la actualización remota a través de la tabla temporal
- Se elimina el registro de la tabla temporal una vez que se ha realizado la actualización.

```
when updating then
  --el registro se queda en el sitio A
  if :new.tipo ='A' and :old.tipo ='A' then
    update revista_1 set revista_id = :new.revista_id,
     tipo=:new.tipo,nombre=:new.nombre
    where revista_id = :old.revista_id;

  --el registro cambia de sitio S2->S1
  elsif :new.tipo = 'A' and :old.tipo ='B' then
    delete from revista_2 where revista_id = :old.revista_id;
    insert into revista_1(revista_id,tipo,nombre)
    values(:new.revista_id,:new.tipo,:new.nombre);
  --el registro se queda en el sitio B
  elsif :new.tipo ='B' and :old.tipo = 'B' then
    update revista_2 set revista_id = :new.revista_id,
      tipo=:new.tipo,nombre=:new.nombre
    where revista_id = :old.revista_id;
  --el registro cambia de sitio S1->S2
  elsif :new.tipo='B' and :old.tipo = 'A' then
    delete from revista_1 where revista_id = :old.revista_id;
    insert into revista_2(revista_id,tipo,nombre)
    values(:new.revista_id,:new.tipo,:new.nombre);
  --valores inválidos
  else
    raise_application_error(-20001,
    'Valor incorrecto para el campo tipo : '
    || :new.tipo
    || ' Solo se permiten los valores A , B ');
  end if;
  --actualiza el binario empleando tabla temporal para update
  delete from t_update_revista_3 where revista_id = :new.revista_id;
  insert into t_update_revista_3 (revista_id,pdf)
  values(:new.revista_id,:new.pdf);

--actualiza el binario en el sitio remoto empleando la tabla temporal
  update revista_3
```

```
  set revista_id = :new.revista_id, pdf = (
    select pdf
    from t_update_revista_3
    where revista_id = :new.revista_id
  ) where revista_id = :old.revista_id;

  delete from t_update_revista_3 where revista_id = :new.revista_id;
```

- Se emplea la variable `:old.<campo>` para hacer referencia al dato viejo, al dato que va a ser sobrescrito.

Observaciones a la solución anterior:

- Existe un problema de desempeño: ¿Qué pasa si en la sentencia `update` no se incluye el dato `blob`?, o simplemente no se requiere actualizarlo.
- Como se puede observar, en todos los casos se realiza un acceso remoto para actualizar el dato `blob` independientemente si cambió o no. No es posible detectar dentro del trigger si dicho dato fue incluido en la sentencia `update`. La única opción es comparar los valores de `:new.pdf` y `:old.pdf` y decidir si se lanza la actualización o no. Esta validación es costosa y no reduciría el problema.
- ¿Solución a lo anterior? Sacrificar transparencia de actualización a través del uso de un procedimiento almacenado. La idea es la siguiente:
    - Omitir la actualización del dato blob en el trigger para evitar el problema de desempeño.
    - Desarrollar un procedimiento almacenado que tendrá que ser invocado por el usuario cuando se requiera aplicar una actualización del dato `blob`.
    - El procedimiento deberá implementar transparencia de inserción para evitar que el usuario indique el sitio donde se hará la actualización..

El código se muestra a continuación:

```
create or replace procedure actualiza_pdf(v_revista_id in number, v_pdf in blob
) is
  begin
    --en este caso todos los blobs se van al sitio 2.
    delete from t_update_revista_3 where revista_id = v_revista_id;
    insert into t_update_revista_3 (revista_id,pdf)
    values(v_revista_id,v_pdf);
    update revista_3
    set  pdf = ( select pdf
      from t_update_revista_3
      where revista_id = v_revista_id
    ) where revista_id = v_revista_id;
```

```
  end;
/
```

Para este ejemplo, no se requiere validar el sitio en el que se encuentra ya que todos los pdfs se guardan en el sitio 2.

1.2.6. Transparencia para operaciones update – sitio 2

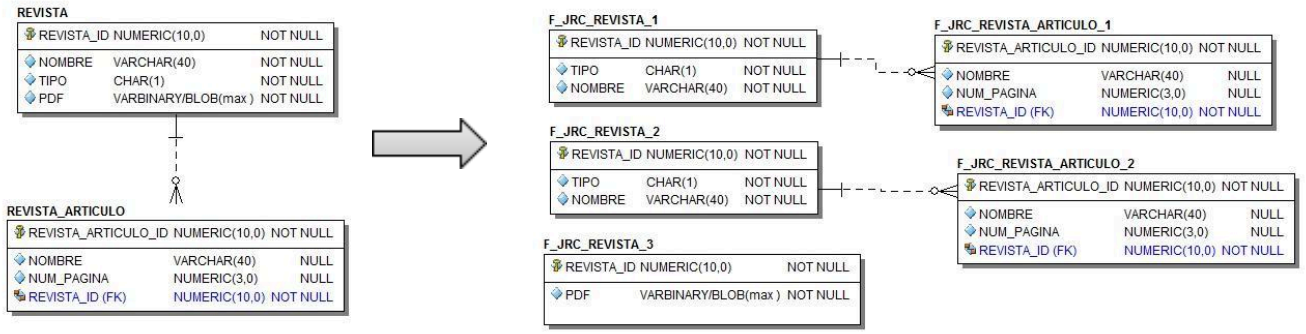
```
when updating then

  --el registro se queda en el sitio B
  if :new.tipo ='B' and :old.tipo = 'B' then
    update revista_2 set revista_id = :new.revista_id,
      tipo=:new.tipo,nombre=:new.nombre
    where revista_id = :old.revista_id;

  --el registro cambia de sitio S1->S2
  elsif :new.tipo='B' and :old.tipo = 'A' then
    delete from revista_1 where revista_id = :old.revista_id;
    insert into revista_2(revista_id,tipo,nombre)
    values(:new.revista_id,:new.tipo,:new.nombre);

  --el registro se queda en el sitio A
    elsif :new.tipo ='A' and :old.tipo ='A' then
    update revista_1 set revista_id = :new.revista_id,
      tipo=:new.tipo,nombre=:new.nombre
    where revista_id = :old.revista_id;

  --el registro cambia de sitio S2->S1
  elsif :new.tipo = 'A' and :old.tipo ='B' then
    delete from revista_2 where revista_id = :old.revista_id;
    insert into revista_1(revista_id,tipo,nombre)
    values(:new.revista_id,:new.tipo,:new.nombre);

  --valores invalidos
  else
    raise_application_error(-20001,
    'Valor incorrecto para el campo tipo : '
    || :new.tipo
    || ' Solo se permiten los valores A , B ');
  end if;
```

```
--actualiza el binario de forma local
update revista_3 set revista_id = :new.revista_id,
  pdf= :new.pdf where revista_id = :old.revista_id;
```

- La diferencia con el código del sitio 1 es que primero se valida con el sitio local y después con el remoto.
- La actualización del dato `blob` se hace de forma normal sin hacer uso de tabla temporal ya que el fragmento que contiene a los datos binarios se encuentra en el mismo sitio.

1.3. Transparencia de operaciones DML para un esquema de fragmentación horizontal derivado

Cuando existe un esquema de fragmentación horizontal derivada, la estrategia para implementar el trigger cambia ligeramente. Para ilustrar este concepto, considerar la siguiente variante al esquema de fragmentación anterior.



Se ha agregado la entidad `revista_articulo` con un esquema de fragmentación horizontal derivada:

$$RA_1 = RA \ltimes_{revista-id} R_1$$

$$RA_2 = RA \ltimes_{revista-id} R_2$$

En este caso la implementación del trigger para la vista `revista_articulo` para una operación de insert será:

```
create or replace trigger t_dml_revista_articulo
instead of insert or update or delete on revista_articulo
declare
   v_count number;
begin
 case
```

```
  when inserting then

   --verifica si hay correspondencia local para evitar acceso remoto
   select count(*) into v_count
   from revista_1
   where revista_id =:new.revista_id;
   --insercion local
   if v_count > 0 then
    insert into revista_articulo_1(revista_articulo_id,
     nombre,num_pagina,revista_id)
    values(:new.revista_articulo_id,:new.nombre,
     :new.num_pagina,:new.revista_id);
   --insercion remota
   else
    select count(*) into v_count
    from revista_2
    where revista_id =:new.revista_id;
  if v_count > 0 then
    insert into revista_articulo_2(revista_articulo_id,
      nombre,num_pagina,revista_id)
    values(:new.revista_articulo_id,:new.nombre,
      :new.num_pagina,:new.revista_id);
  else
    raise_application_error(-20001,
       'Error de integridad para el campo revista_id : '
          ||  :new.revista_id
          || ' No se encontró el registro padre en fragmentos');
   end if;
 end if;


 when updating then
   dbms_output.put_line('implementar');
  when deleting then
    dbms_output.put_line('implementar');
 end case;
end;
/
```

- Observar las líneas destacadas. En este tipo de fragmentación, el registro a insertar debe ubicarse en el mismo fragmento que el registro padre debido al esquema de fragmentación horizontal derivada.
- Es incorrecto aplicar el mismo criterio que en la tabla padre ya que en este caso, el campo tipo no se encuentra en la tabla `revista_articulo`.

- La técnica en este caso es buscar al registro padre empleando la el valor de la PK.
- Observar el uso de una sentencia `select` empleando la función count para validar si el registro padre se encuentra en el fragmento `revista_1` empleando el valor de la PK como criterio de búsqueda. El resultado de la consulta se asigna a la variable `v_count`.
- Esta técnica es similar a realizar una operación de semijoin: Se verifica si el nuevo registro participará en una operación `join` con respecto a la tabla `revista_1`.
- Si la validación encuentra correspondencia (la función `count` obtendrá el valor 1), significa que el registro es derivado de `revista_1` por lo tanto, debe ser insertado en `revista_articulo_1` .
- Observar el orden: primero se valida con respecto a `revista_1` y después con respecto a `revista_2`. La razón de seguir este orden es por desempeño: primero se trata de validar localmente, y sólo en caso de no haber correspondencia local, se valida con el fragmento remoto `revista_2`.
- Finalmente, en caso de no encontrar el registro padre en ninguno de los sitios, observar que se lanza un error equivalente a un error de integridad indicando que el registro padre no fue encontrado. Esto permite verificar la integridad de los datos entre fragmentos.

La misma lógica se realiza para el sitio 2, solo que en orden contrario: primero se valida en `revista_2` y después en `revista_1`:

```
when inserting then

  --verifica si hay correspondencia local para evitar acceso remoto
  select count(*) into v_count
  from revista_2
  where revista_id =:new.revista_id;
  --insercion local
  if v_count > 0 then
    insert into revista_articulo_2(revista_articulo_id,
      nombre,num_pagina,revista_id)
    values(:new.revista_articulo_id,:new.nombre,
      :new.num_pagina,:new.revista_id);
  --insercion remota
  else
    select count(*) into v_count
    from revista_1
    where revista_id =:new.revista_id;
    if v_count > 0 then
      insert into revista_articulo_1(revista_articulo_id,
        nombre,num_pagina,revista_id)
      values(:new.revista_articulo_id,:new.nombre,
        :new.num_pagina,:new.revista_id);
```

```
    else
      raise_application_error(-20001,
        'Error de integridad para el campo revista_id :'
           ||  :new.revista_id
           || ' No se encontró el registro padre en fragmentos');
    end if;
  end if;
```

Para operaciones `update` y `delete` se puede emplear la misma técnica: uso de una sentencia select y la función count para validar correspondencia entre fragmento padre y fragmento hijo.

1.4. Verificar transparencia de operaciones DML

Para verificar los resultados del desarrollo anterior, se realizará un nuevo script PL/ SQL que realizará operaciones insert, update, delete lanzadas desde ambos sitios hacia la entidad `revista`. El código se muestra a continuación.

```
set serveroutput on

Prompt ejecutando prueba en sitio 1
connect ejemplo_revistas/ejemplo_revistas@jrcbd_s1
@s-06-jrc-sentencias-prueba.sql

Prompt ejecutando prueba en sitio 2
connect ejemplo_revistas/ejemplo_revistas@jrcbd_s2
@s-06-jrc-sentencias-prueba.sql
```

Observar que el script invoca a otro llamado `s-06-jrc-sentencias-prueba.sql` en ambos sitios. La solución deberá funcionar en cualquiera de los 2 sitios por lo que se ejecuta el mismo script en el sitio 1 y en el sitio 2.
El contenido del archivo `s-06-jrc-sentencias-prueba.sql` se muestra a continuación.

```
Prompt Copiando Pdfs de muestra a /tmp
!cp pdf/sample*.pdf  /tmp

declare
 v_count number;
begin
  dbms_output.put_line('Probando transparencia de eliminación');
  delete from revista;
  dbms_output.put_line('validando que existen 0 registros en los fragmentos');
```

```
 select count(*) into v_count
 from (
  select 1 from revista_1
  union
  select 1 from revista_2
  union
  select 2 from revista_3
 ) q1;

 if v_count <> 0 then
  raise_application_error(-20001, 'Se encontraron registros para la tabla
revista');
 else
  dbms_output.put_line('Ok, 0 registros encontrados');
 end if;

 --Probando insercion
 dbms_output.put_line('insertando  revista 101  tipo A');
 insert into revista(revista_id,tipo,nombre,pdf) values(
  101,'A','Revista 101',load_blob_from_file('TMP_DIR','sample1.pdf'));

 dbms_output.put_line('insertando  revista 102  tipo B');
 insert into revista(revista_id,tipo,nombre,pdf) values(
  102,'B','Revista 102',load_blob_from_file('TMP_DIR','sample2.pdf'));

 dbms_output.put_line('insertando  revista 103  tipo A');
 insert into revista(revista_id,tipo,nombre,pdf) values(
  103,'A','Revista 103',load_blob_from_file('TMP_DIR','sample3.pdf'));

 dbms_output.put_line('Insertando  revista 104  tipo B');
 insert into revista(revista_id,tipo,nombre,pdf) values(
  104,'B','Revista 100',load_blob_from_file('TMP_DIR','sample4.pdf'));

 dbms_output.put_line('insertando  revista 105  tipo A');
 insert into revista(revista_id,tipo,nombre,pdf) values(
  105,'A','Revista 100',load_blob_from_file('TMP_DIR','sample5.pdf'));

 select count(*) into v_count from revista;

 if v_count <> 5 then
  raise_application_error(20001, 'Insercion invalida, se esperaban 5
registros');
 else
```

```
  dbms_output.put_line('OK, '|| v_count|| ' registros encontrados');
 end if;
 --confirmar cambios antes de modificar
 commit;

 dbms_output.put_line('Probando update');
 update revista set nombre='revista 100-1' where revista_id = 101;
 update revista set nombre='revista 100-2' where revista_id = 102;
 update revista set nombre='revista 100-3' where revista_id = 103;
 update revista set nombre='revista 100-4' where revista_id = 104;
 update revista set nombre='revista 100-5' where revista_id = 105;

 dbms_output.put_line(
  'Probando update con datos binarios, se asignan al revés');

 dbms_output.put_line('Cambiando pdf 1');
 update revista set pdf = load_blob_from_file('TMP_DIR','sample5.pdf')
 where revista_id = 1;

dbms_output.put_line('Cambiando pdf 2');
 update revista set pdf = load_blob_from_file('TMP_DIR','sample4.pdf')
 where revista_id = 2;

dbms_output.put_line('Cambiando pdf 3');
 update revista set pdf = load_blob_from_file('TMP_DIR','sample3.pdf')
 where revista_id = 3;

dbms_output.put_line('Cambiando pdf 4');
 update revista set pdf = load_blob_from_file('TMP_DIR','sample2.pdf')
 where revista_id = 4;

dbms_output.put_line('Cambiando pdf 5');
 update revista set pdf = load_blob_from_file('TMP_DIR','sample1.pdf')
 where revista_id = 5;
 dbms_output.put_line(' confirmando cambios');
 commit;
exception
  when others then
  dbms_output.put_line('error al ejecutar la prueba, se hará rollback');
  rollback;
  raise;
end;
/
select revista_id,tipo,nombre,dbms_lob.getlength(pdf) pdf_length from revista;
```

- Observar el comando `!cp` . El signo "!" permite ejecutar comandos del sistema operativo desde SQL *Plus. En este caso se realiza la copia de 5 archivos PDF de muestra para ser insertados.
- La primera prueba que se realiza es la transparencia de eliminación. Observar que la sentencia se ve como si la BD no estuviera distribuida.
- Para validar el resultado a continuación se realiza un conteo de registros en cada fragmento. Si se encuentran registros se lanza un error.
- El siguiente paso es verificar la transparencia de inserción. Nuevamente, observar que las sentencias `insert` son totalmente transparentes.
- Observar que se invoca a la función `load_blob_from_file` .
- Esta función es similar a los procedimientos almacenados que realizan la carga y escritura de archivos binarios. En esta ocasión se emplea una función en lugar de un procedimiento almacenado. Su objetivo es leer un dato `blob` del directorio configurado en prácticas anteriores y regresarlo como un dato tipo `blob`. El código de esta versión se muestra a continuación.

```
create or replace function load_blob_from_file(
  v_directory_name      in varchar2,
  v_src_file_name       in varchar2 ) return blob is  --se regresa un dato BLOB

  --variables
  v_src_blob bfile:=bfilename(v_directory_name,v_src_file_name);
  v_dest_blob blob := empty_blob();
  v_src_offset number := 1;
  v_dest_offset number :=1;
  v_src_blob_size number;

  begin
    --abre el archivo
    if dbms_lob.fileexists(v_src_blob)=1
      and not dbms_lob.isopen(v_src_blob)=1 then
      v_src_blob_size := dbms_lob.getlength(v_src_blob);
      dbms_lob.open(v_src_blob,dbms_lob.LOB_READONLY);
    else
      raise_application_error(-20001, v_src_file_name
            ||' El archivo no existe o está abierto');
    end if;

    --crea un objeto lob temporal
```

```
    dbms_lob.createtemporary(
        lob_loc => v_dest_blob,
          cache   => true,
          dur     => dbms_lob.call
    );
    --lee el archivo y escribe en la variable v_dest_blob
    dbms_lob.loadblobfromfile(
        dest_lob    => v_dest_blob,
        src_bfile   => v_src_blob,
        amount      => dbms_lob.getlength(v_src_blob),
        dest_offset => v_dest_offset,
        src_offset  => v_src_offset
    );
    --cerrando blob
    dbms_lob.close(v_src_blob);
    if v_src_blob_size = dbms_lob.getlength(v_dest_blob) then
      dbms_output.put_line('done '|| v_src_blob_size || ' bytes read.' );
    else
      raise_application_error(-20104,'Invalid blob size. Expected: '
            ||v_src_blob_size||', actual: '|| dbms_lob.getlength(v_dest_blob));
    end if;
    return v_dest_blob;
  end;
/
```

- La función recibe 2 parámetros: el nombre del directorio (objeto en la BD) y el nombre del archivo. La función realiza la lectura del archivo, construye un objeto tipo `blob` y regresa el puntero (Lob Locator) para ser empleado como valor en la sentencia `Insert`.
- Continuando con la explicación del archivo `s-06-jrc-sentencias-prueba.sql`, observar las sentencias update, primero se modifica un dato alfanumérico y después se actualiza el archivo PDF.
- Es importante mencionar el manejo transaccional. Debido a que se hacen múltiples operaciones sobre BD remotas, es indispensable asegurarse de encapsular y delimitar estas instrucciones dentro de una transacción.
- Observar que se realiza `commit` si todo va bien. Sin embargo, ¿Qué pasaría si se genera un error durante su ejecución?
- Si el error no es correctamente manejado, la transacción no podría llegar a un estado final ya sea `commit` o `rollback`.
- Para ilustrar este comportamiento, suponer que se intenta insertar un dato con una PK nula. Si el error no se manejara correctamente se obtendría el siguiente error:

```
ERROR at line 1:
ORA-02055: distributed update operation failed; rollback required
ORA-01400: cannot insert NULL into
("EJEMPLO_REVISTAS"."F_JRC_REVISTA_1"."REVISTA_ID")
ORA-02063: preceding line from JRCBD_S1
ORA-06512: at "EJEMPLO_REVISTAS.T_DML_REVISTA", line 14
ORA-04088: error during execution of trigger 'EJEMPLO_REVISTAS.T_DML_REVISTA'
ORA-06512: at line 94

select revista_id,tipo,nombre,dbms_lob.getlength(pdf)pdf_length from revista
                                                                        *
ERROR at line 1:
ORA-02067: transaction or savepoint rollback required
```

- Poner atención en las líneas resaltadas. Antes de realizar cualquier otra cosa en la BD se debe terminar esta transacción distribuida. El manejador protege el estado consistente de la BDD impidiendo cualquier otra operación hasta que se haga rollback debido al error generado.
- Para evitar este problema, el código que ejecuta operaciones DML sobre una BDD debe manejar posibles errores de forma correcta.
- El manejo del error en el script se realiza en el bloque `exception`:

```
exception
  when others then
    dbms_output.put_line('error al ejecutar la prueba, se hará rollback');
    rollback;
    raise;
```

- Si ocurre un error, la transacción se termina aplicando `rollback` y se re-lanza la excepción para que pueda ser reportada al usuario. No olvidar realizar este manejo básico para cualquier operación DML distribuida.
- Al realizar el manejo adecuado se obtiene lo siguiente:

```
ERROR at line 1:
ORA-01400: cannot insert NULL into
("EJEMPLO_REVISTAS"."F_JRC_REVISTA_1"."REVISTA_ID")
ORA-02063: preceding line from JRCBD_S1
ORA-06512: at "EJEMPLO_REVISTAS.T_DML_REVISTA", line 14
ORA-04088: error during execution of trigger 'EJEMPLO_REVISTAS.T_DML_REVISTA'
ORA-06512: at line 94
```

Observar que ahora ya no aparece el problema del manejo transaccional, la BD puede continuar realizando otras operaciones.

Hasta este punto concluye la explicación del proceso para implementar transparencia de distribución y transaccionalidad.

Continuar con el documento del ejercicio práctico para realizar los ejercicios correspondientes. Los scripts de este ejemplo se pueden obtener de la carpeta compartida BDD, pueden ser empleados como base para los ejercicios de la práctica.