1. TRANSPARENCIA DE DISTRIBUCIÓN PARA LA INSTRUCCIÓN SELECT

1. TRANSPARENCIA DE DISTRIBUCIÓN PARA LA INSTRUCCIÓN SELECT.....1
1.1. Objetivo.....2
1.2. Implementación de transparencia.....2
1.2.1. Transparencia de distribución.....2
1.2.1.1. Transparencia de fragmentación.....2
Ejemplo.....2
1.2.1.2. Transparencia de localización.....2
Ejemplo.....2
1.2.1.3. Transparencia de mapeo local.....3
Ejemplo.....3
1.3. Implementación de Transparencia de localización.....3
Ejemplo.....4
1.3.1. Consultas empleando transparencia de localización.....5
Ejemplo.....6
1.4. Implementación de Transparencia de fragmentación.....8
1.5. Manejo de blobs.....9
1.5.1. Observaciones de la solución anterior.....15
Ejemplo 2.....17
1.5.2. Ejecución del ejemplo.....20
1.6. Implementación de transparencia de fragmentación con datos blob.....21
1.7. Consultas empleando transparencia de fragmentación.....23
1.7.1. Conteo de registros.....23
Ejemplo.....24
1.7.2. Consultas de transparencia de fragmentación para datos blob.....24
Ejemplo.....24
1.8. Validación de resultados.....25
1.9. Contenido del reporte.....26

Notas:

- Para todas las actividades de este ejercicio práctico, considerar el esquema de fragmentación implementado en ejercicios anteriores.
- El ejercicio práctico se realiza de forma individual

1.1. Objetivo

Comprender e implementar transparencia de localización y de fragmentación aplicados a una base de datos distribuida para realizar operaciones `select`.

1.2. Implementación de transparencia

En el ejercicio práctico anterior se implementó el primer nivel de transparencia de distribución: Mapeos Locales. En esta ocasión se implementarán los 2 siguientes niveles: transparencia de fragmentación y transparencia de localización únicamente para la instrucción `select`.

1.2.1. Transparencia de distribución

A nivel general, este tipo de transparencia permite realizar operaciones en una base de datos distribuida exactamente de la misma forma que una base de datos centralizada. Existen 3 niveles de transparencia de distribución que pueden ser implementados: mapeos locales, transparencia de localización y transparencia de fragmentación.

1.2.1.1. Transparencia de fragmentación

Representa el nivel mayor. El usuario final no se percata que una tabla está fragmentada, por lo que en la consulta no se requiere especificar ni el nombre del fragmento ni su ubicación.

Ejemplo

```
select * from pais;
```

En esta consulta, la BD obtendrá todos los países registrados sin importar que los registros se encuentren en diferentes sitios y en fragmentos específicos. Notar que la tabla `PAIS` pertenece al esquema global de la BDD.

1.2.1.2. Transparencia de localización

En este nivel el usuario necesita especificar los nombres de los fragmentos más sin embargo no necesita especificar donde se encuentran los fragmentos.

Ejemplo

```
select * from pais_s1
union all
select * from pais_s2
```

En esta consulta los datos de la tabla `pais` se encuentran en los sitios s1 y s2. En ambos sitios se ha definido un sinónimo que oculta el nombre de la tabla (fragmento) y la ubicación de la misma (a través del uso de una liga).

1.2.1.3. Transparencia de mapeo local

En este caso el usuario necesita especificar tanto el nombre del fragmento como sus ubicaciones

Ejemplo

```
select * from f_jrcbd_pais_1;
union all
select * from f_jrcbd_pais_2@jrcbdd_s2;
```

En la consulta se especifica ambas cosas, el nombre del fragmento `pais` y su ubicación representada por el nombre de la liga: `jrcbdd_s2`.

1.3. Implementación de Transparencia de localización.

Recordando los componentes de una BDD empleados para implementar transparencia, destacan los siguientes:

- *Esquema global de la BDD*. Representa la vista de un usuario final que oculta su distribución. El esquema global estará presente en todos los sitios de la BDD. Ejemplo: En este esquema existirá un objeto llamado `pais` (no es una tabla) que representa a la entidad `pais` del esquema global. La definición de este objeto permitirá realizar las operaciones necesarias para acceder a los datos de sus fragmentos asociados.
- *Esquema local*. Ubicado en cada sitio. Cada objeto representa por lo general a cada fragmento. Ejemplo: En el sitio 1 existirá la tabla `pais_s1` y en el sitio 2 existirá la tabla `pais_s2`. Ambas tablas representan a los fragmentos de la entidad global `pais`.
- Para implementar este tipo de transparencia se hace uso de ***sinónimos***. La finalidad principal es ocultar el uso de ligas con lo que se libera al usuario conocer detalles en cuanto a la ubicación del fragmento. En este nivel solo se requiere conocer el nombre del fragmento.

Para cada PDB, generar un sinónimo por cada fragmento empleando como nombre la convención: `<nombre_tabla_global>_<n>` donde `n` representa el número de fragmento.

Ejemplo

Para los fragmentos `f_jrcbd_pais_1` y `f_jrcbd_pais_2` que se encuentran en los sitios S1 y S2 respectivamente, se deberán crear 2 sinónimos en cada sitio. Es decir, un total de 4 sinónimos:

- Para la PDB `jrcbdd_s1`:
  - Sinónimo con nombre `pais_1` que apunta al fragmento local `f_jrcbd_pais_1`.
  - Sinónimo con nombre `pais_2` que apunta al fragmento ubicado en el otro sitio `f_jrcbd_pais_2@jrcbdd_s2`.
- Para la PDB `jrcbdd_s2`:
  - Sinónimo con nombre `pais_1` que apunta al fragmento remoto `f_jrcbd_pais_1@jrcbdd_s1.`
  - Sinónimo con nombre `pais_2` que apunta al fragmento local `f_jrcbd_pais_2`.

- Observar que se ha eliminado el prefijo `f_<iniciales>`
- Notar que se requiere hacer uso de las ligas creadas anteriormente: S1 →S2 y S2 → S1 para realizar la comunicación bidireccional.
- Tener cuidado al hacer uso de las ligas, ya que esta se debe usar para apuntar al fragmento remoto.

Los sinónimos deberán cumplir con los siguientes requisitos:

- Deberán ser sinónimos privados creados en cada PDB por el usuario empleado en ejercicios prácticos anteriores (no se debe emplear al usuario `SYS` para crear sinónimos).
- Dicho usuario deberá tener privilegios para crear sinónimos privados. El usuario `SYS` deberá otorgar estos privilegios.

Generar scripts SQL `s-01-<iniciales>-privilegios-usuarios.sql` El script deberá realizar las siguientes acciones:

- Conectarse a cada PDB como usuario `SYS`
- Ejecutar las instrucciones necesarias para otorgar los privilegios para crear sinónimos al usuario empleado en ejercicios prácticos anteriores. No olvidar agregar el encabezado a cada script.
- Para el desarrollo de este ejercicio práctico se requieren privilegios adicionales. Agregar al script los siguientes privilegios:
  - Privilegios para crear vistas
  - Privilegios para crear tipos de datos personalizados (`create type`)
  - Privilegios para crear procedimientos
- Ejecutar el script..

Para mayores detalles en cuanto a la creación de sinónimos públicos y privados: `tema7-parte1.pdf` en la carpeta BD correspondiente al tema 7.

Generar un script SQL `s-02-<iniciales>-sinonimos.sql` donde `<iniciales>` corresponden a las iniciales del esrudiante. El script deberá realizar la conexión a cada PDB para crear los sinónimos

```
--@Autor:           Jorge Rodriguez
--@Fecha creación:  dd/mm/yyyy
--@Descripción:     Creación de sinónimos

Prompt conectandose a jrcbdd_s1
connect <usuario>_bdd/<usuario>_bdd@jrcbdd_s1

Prompt creando sinónimos en jrcbdd_s1

create or replace synonym pais_1 for f_jrc_pais_1;
create or replace synonym pais_2 for f_jrc_pais_2@jrcbdd_s2;

-- otros sinonimos

Prompt conectandose a jrcbdd_s2
connect <usuario>_bdd/<usuario>_bdd@jrcbdd_s2

Prompt creando sinónimos en jrcbdd_s2

create or replace synonym pais_1 for f_jrc_pais_1@jrcbdd_s1;
create or replace synonym pais_2 for f_jrc_pais_2;

-- otros sinonimos

Prompt Listo!
exit
```

1.3.1. Consultas empleando transparencia de localización

- Generar un script llamado `s-03-<iniciales>-consultas-localizacion.sql` El script contendrá una sentencia `select` que obtenga el total de los registros de cada fragmento (tanto locales como remotos) empleando los sinónimos creados anteriormente.
- El script es similar al realizado en el ejercicio práctico anterior haciendo uso de la expresión de reconstrucción, solo que ahora se emplearán los sinónimos en lugar del uso de las ligas.

Notar que el uso de sinónimos permite ejecutar el script en cualquiera de las 2 PDBs.

Para evitar duplicar el código de este script, crear uno nuevo llamado `s-03-<iniciales>-consultas-localizacion-main.sql` El script deberá conectarse a cada una de las PDBs y ejecutar el script `s-03-<iniciales>-consultas-localizacion.sql`

Ejemplo

```
--@Autor:           Jorge Rodriguez
--@Fecha creación:  dd/mm/yyyy
--@Descripción:     Script encargado de realizar consultas con sinónimos
--                  en ambas PDBs

prompt conectando a jrcbdd_s1
connect <usuario>/<password>@jrcbdd_s1
prompt Realizando conteo de registros
set serveroutput on
start s-03-jrc-consultas-localizacion.sql

prompt conectando a jrcbdd_s2
connect <usuario>/<password>@jrcbdd_s2
prompt Realizando conteo de registros
set serveroutput on
start s-03-jrc-consultas-localizacion.sql
```

- Ejecutar el script anterior.
- El código del script `s-03-<iniciales>-consultas-localizacion.sql` será similar al siguiente ejemplo:

```
--@Autor:           Jorge Rodriguez
--@Fecha creación:  dd/mm/yyyy
--@Descripción:     Script encargado de realizar consultas con sinónimos.
--                  El script se puede ejecutar en cualquier PDB.
declare
 v_num_paises number;
  --agregar una variable para cada tabla
begin

 dbms_output.put_line('Realizando consulta empleando sinonimos');
 --consultando paises
 select count(*) into v_num_paises
 from (
  select pais_id
  from  pais_1
  union all
  select pais_id
  from  pais_2
 ) q1;

 --Completar  para las demás tablas del modelo

  dbms_output.put_line('Resultado del conteo de registros');
  dbms_output.put_line('================================');
  dbms_output.put_line('Paises:         '||v_num_paises);
  --completar para las demás tablas del modelo
end;
/
```

Ejecutar el script. Su salida deberá similar a la siguiente:

```
dle> start s-03-jrc-consultas-main.sql
conectando a jrcbdd_s1
Connected.
Realizando conteo de registros
Realizando consulta empleando sinónimos
Resultado del conteo de registros
================================
Países:  2
-- otras tablas
...
...

PL/SQL procedure successfully completed.
```

```
conectando a jrcbdd_s2
Connected.
Realizando conteo de registros
Realizando consulta empleando sinonimos
Resultado del conteo de registros
================================
Paises:  2
-- otras tablas
...
...

PL/SQL procedure successfully completed.
```

1.4. Implementación de Transparencia de fragmentación

Para implementar este nivel de transparencia se hará uso de una vista que permita a un usuario final mostrar la lista de todos los registros existentes sin tener que generar una consulta por cada sitio de forma separada.

Para cada tabla del esquema global se deberá crear una vista con el mismo nombre. Por ejemplo, para los fragmentos `pais_1` y `pais_2` se creará una vista llamada `pais` en cada uno de los sitios. Dicha vista pertenece al esquema global con el cual el usuario final podrá realizar operaciones como si se tratara de una sola tabla. La vista ocultará la consulta hacia los fragmentos empleando la expresión de reconstrucción en su definición. La consulta que define a cada vista deberá hacer uso de los sinónimos creados en la actividad anterior para hacer referencia a cada sitio.

Generar 2 scripts:

- `s-04-<iniciales>-vistas-main.sql` Debido a que la definición de las vistas es exactamente la misma para cada nodo de la BDD, en este script se deberá realizar una conexión a cada PDB y se deberá invocar al siguiente archivo para crear las vistas.
- `s-04-<iniciales>-def-vistas.sql` Este script contendrá la definición de cada vista. No emplear `select *`, es decir, se deberán escribir todos los campos. Esto es importante en especial para fragmentaciones verticales o híbridas. La razón es que las columnas comunes en cada fragmento, por ejemplo, la PK aparecerá duplicada en la definición de la vista.

Ejecutar el script `s-04-<iniciales>-vistas.sql`

Ejemplo:
Script `s-04-<iniciales>-vistas-main.sql`

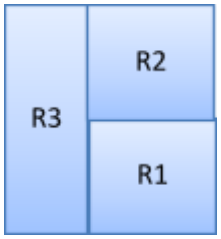
```
--@Autor:           Jorge Rodriguez
--@Fecha creación:  dd/mm/yyyy
--@Descripción:    Script de ejecución para crear de vistas
--                 en ambas PDBs

prompt conectándose a jrcbdd_s1
connect <usuario_bdd>/<usuario_bdd>@jrcbdd_s1

Prompt creando vistas en jrcbdd_s1
@s-04-jrc-def-vistas.sql

Prompt conectándose a jrcbdd_s2
connect <usuario_bdd>/<usuario_bdd>@jrcbdd_s2

Prompt creando vistas en jrcbdd_s2
@s-04-jrc-def-vistas.sql

Prompt Listo!
exit
```

Nota Importante: No incluir en estos scripts entidades que tengan columnas con tipo de dato `blob/clob`. El tratamiento para los fragmentos de estas entidades es diferente, se revisará en la siguiente sección.

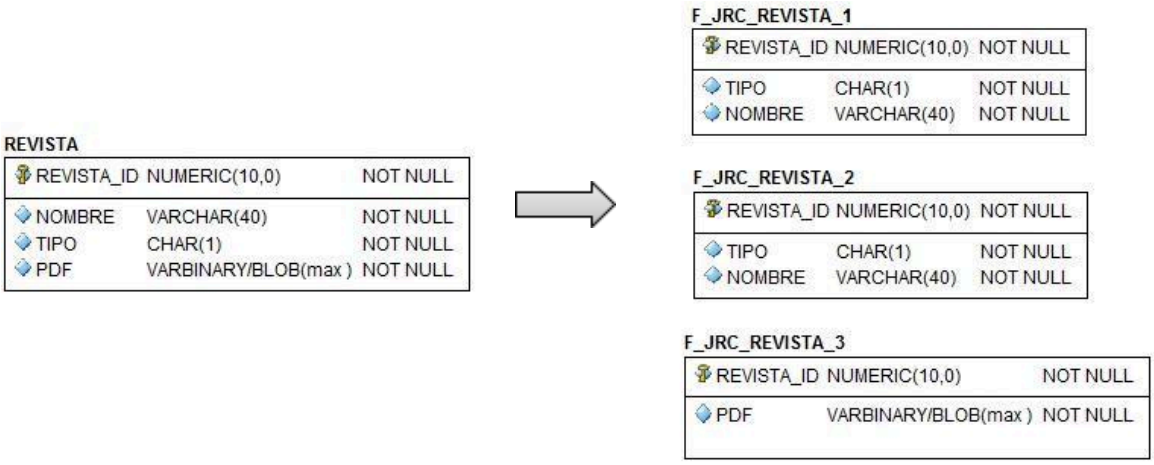
1.5. Manejo de blobs

Para implementar transparencia de distribución de datos tipo `blob` y `clob` entre sitios remotos existen ciertas restricciones. Una de ellas es la empleada para hacer selecciones. Para ilustrar el manejo de este tipo de datos, suponer la siguiente entidad `revista` la cual se ha fragmentado de forma híbrida:

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



El diseño de fragmentos quedará de la siguiente manera:



- R2 y R3 están en el sitio 2, y R1 en el sitio 1
- La definición de la vista para implementar transparencia de distribución será:

```
create or replace view revista as
  select q1.revista_id,q1.tipo,q1.nombre
  from (
    select revista_id,tipo,nombre from revista_1
    unión all
    select revista_id,tipo,nombre from revista_2
  ) q1
  join (select revista_id, pdf from revista_3)  q2
  on q1.revista_id = q2.revista_id;
```

Al intentar crear la vista en el sitio 2 no se obtendrá error, pero al intentar crearla en el sitio 1 se obtendrá:

```
ORA-22992: cannot use LOB locators selected from remote tables
```

En el sitio 2 no se produce error debido a que el fragmento que contiene el campo `blob` se encuentra de forma local.

- Lob Locator es una técnica empleada para acceder al valor o contenido de un dato de tipo `clob` y `blob`. Puede ser visto como un "puntero" a los datos binarios que se emplean para almacenar datos `clob`, `blob`, `Nclob` y `BFILE`.
- El uso de Lob Locators está limitado en tablas remotas. Los valores de estos "punteros" no pueden ser tratados de forma correcta en sitios remotos.
- Sin embargo, el uso de funciones y scripts PL/SQL son soportados para manejar LOBs en sitios remotos entre otras técnicas. Ver este enlace para mayor información.
- Para resolver este problema, se emplea la siguiente técnica. Todos los objetos que se crearán a continuación deberán crearse en el sitio donde se requiere acceder de forma remota al fragmento con el campo `blob`. En este ejemplo, todos los objetos se crearán en el sitio 1.

1. Crear un tipo de dato personalizado que contenga en este caso las columnas del fragmento que contiene la columna `blob`, `clob` o `BFILE`. Antes se deberá otorgar permisos al usuario correspondiente para crear objetos `TYPE` (este privilegio ya fue otorgado en el primer script de este ejercicio práctico).

```
connect sys/system@jrcbdd_s1 as sysdba
grant create type to jorge;

connect jorge/jorge@jrcbdd_s1
create type pdf_type as object (
  revista_id number(10,0),
  pdf blob
);
/
show errors;
```

Como se puede observar, en este ejemplo, el fragmento remoto se forma de solo 2 columnas: `revista_id` y `pdf`. En general, el objeto `type` debe contener todas las columnas del fragmento.

2. Crear un nuevo objeto tipo `TABLE` asociado al objeto `pdf_type` creado anteriormente. Este objeto será empleado como valor de retorno de una función creada más adelante. En Oracle, si una función regresa un objeto tipo `table`, este podrá ser empleado como tal; es decir, se emplea como si se tratara de una tabla real permitiendo hacer operaciones como `join`, etc.

```
create type pdf_table as table of pdf_type;
/
show errors;
```

- Observar que estas 2 últimas instrucciones requieren del carácter `/` al final. En Oracle, `/` se emplea para distinguir el final de un script, o para identificar el final de un bloque PL/SQL. Recordando, un bloque PL/SQL es aquel que contiene al menos las instrucciones `BEGIN` - `END;` y en su interior contiene múltiples instrucciones, cada una terminando en `;`. Si en un solo script SQL existen varios bloques PL/SQL, `/` se puede emplear al final de cada bloque o también, se puede escribir `/` una sola vez hasta el final del archivo.
- Para el caso de la instrucción `create type` , se requiere `/` ya que dicha instrucción puede contener bloques PL/SQL. De aquí surge la necesidad de incluir `/` al final.

3. Crear una tabla temporal. Los datos de la tabla remota se leerán y se almacenarán en una tabla temporal, de preferencia con el prefijo `t_` y el nombre del fragmento remoto:

```
create global temporary table t_jrc_revista_3(
  revista_id number(10,0) constraint t_jrc_revista_3_pk  primary key,
  pdf blob not null
) on commit preserve rows;
```

4. Crear una función. Su objetivo es leer los datos de la tabla remota, almacenarlos en una tabla temporal para posteriormente iterar sobre la lista obtenida y regresar objetos tipo `pdf_type` para que puedan ser empleados como si se tratara de una tabla local. Revisar cuidadosamente el siguiente ejemplo, otorgar permisos al usuario correspondiente para que pueda crear procedimientos y/o funciones.

```
connect sys/system@jrcbdd_s1 as sysdba
grant create procedure to jorge;

connect jorge/jorge@jrcbdd_s1
create or replace function get_remote_pdf return pdf_table pipelined is
pragma autonomous_transaction;
  v_temp_pdf blob;
  begin
    --Inicia txn autónoma 1.
    --asegura que no haya registros
    delete from t_jrc_revista_3;
    --Inserta los datos obtenidos del fragmento remoto a la tabla temporal.
    insert into t_jrc_revista_3 select revista_id,pdf from revista_3;
    --termina txn autónoma 1 antes de iniciar con la construcción
    --del objeto pdf_table
    commit;

    --obtiene los registros de la tabla temporal y los regresa como
    --objetos tipo pdf_type
    for cur in (select revista_id,pdf from t_jrc_revista_3)
    loop
      pipe row(pdf_type(cur.revista_id,cur.pdf));
    end loop;
    --Inicia txn autónoma 2 para limpiar la tabla
    --elimina los registros de la tabla temporal una vez que han sido obtenidos.
    delete from t_jrc_revista_3;
    --termina txn autónoma 2
    commit;
```

```
    return;
  exception
    when others then
    --termina txn autónoma en caso de ocurrir un error
    rollback;
   --relanza el error para que sea propagado a quien invoque a esta función
  end;
  /
show errors;
```

- Del código anterior, considerar la siguiente instrucción:

```
insert into t_jrc_revista_3 select revista_id,pdf from revista_3
```

- La sentencia permite extraer el dato `blob` del fragmento remoto. Esta técnica si es soportada en Oracle, se emplea una tabla temporal.
- Por definición las operaciones DML no se permiten en una función. Una de las razones es que una función no debe alterar el estado de los datos. Una función solo debe realizar ciertos cálculos con base a los parámetros recibidos y obtener un resultado.
- Por otro lado, ejecutar operaciones DML implica el uso de transacciones. Esta condición puede afectar la ejecución global de una transacción la cual se inició fuera de la función. Por lo tanto, una función no debe alterar la ejecución y control de transacciones existentes o en curso.
- La razón por la cual se emplea una función como mecanismo de obtención de valores `blob` es debido a que en la definición de la vista se requiere de un mecanismo que regrese como resultado la lista de valores `blob` para poder integrarlo al resultado de la sentencia `select`. Otra opción sería emplear un procedimiento almacenado con un parámetro de salida lo cual no es opción viable para implementar transparencia, ya que el procedimiento requiere de variables; cuestión que no se puede realizar con una sentencia `select` simple.
- La instrucción `pragma autonomous_transaction` cambia la forma en la que se ejecuta un subprograma; en este caso, el código que integra a la función. Cuando el subprograma (bloque de código) es marcado con esta instrucción, este puede realizar operaciones DML, puede crear y cerrar transacciones sin modificar el estado de la transacción externa o principal que existe al invocar a la función. Por lo tanto, la instrucción `pragma autonomous_transaction` es obligatoria cuando existan operaciones DML dentro de la función.
- Para cumplir con la regla en la que una función no debe modificar el estado de los datos, observar que en la función se obtienen los valores `blob` del fragmento remoto, se insertan en una tabla temporal, se recuperan los valores `blob` de la tabla temporal y se envían como resultado. Antes de concluir, los datos se eliminan de la

tabla temporal para dejar a la BD en el mismo estado al inicio de la ejecución de la función.

- Notar el uso de la instrucción `pipe` para incluir el registro `pdf_type` dentro del objeto `pdf_table`. Este último objeto se emplea como si se tratara de una tabla local y poder aplicar una operación `join` con los demás fragmentos.
- Observar que en el código se crean 2 transacciones autónomas. La primera realiza la limpieza inicial e inserción de los datos. Justo después de este paso, la transacción debe concluir para poder iniciar con la construcción del objeto `pdf_table` (tabla de datos). Una vez que la construcción de esta tabla de datos ha concluido, se procede a limpiar la tabla temporal para que el estado de la BD quede intacto. Para poder ejecutar la instrucción `delete`, se crea la segunda transacción anónima y termina justo después de la instrucción `delete`.
- Finalmente, observar el bloque de excepción. Si llegara a ocurrir algún error durante la ejecución de la función, existiría una transacción autónoma iniciada pero no concluida. Esto representaría un error ya que con base a lo explicado anteriormente, la función debe ser responsable de iniciar y terminar las transacciones autónomas generadas al interior de la función con la finalidad de no afectar otras transacciones en curso. En este caso, al detectar un error, se procede a realizar un `rollback` para regresar a un estado consistente y se relanza el error para que sea manejado por el programa o código que invoque a la función.

5. Definición de vistas

Para el caso del sitio 2, la vista puede omitir el uso de la función anterior ya que el fragmento que contiene los datos `blob` se encuentra local:

```
connect jorge/jorge@jrcbdd_s2
create or replace view revista as
  select q1.revista_id,q1.tipo,q1.nombre,q2.pdf
  from (
    select revista_id,tipo,nombre from revista_1
    unión all
    select revista_id,tipo,nombre from revista_2
  ) q1
  join revista_3  q2
  on q1.revista_id = q2.revista_id;
```

Sin embargo, para el sitio 1, la definición de la vista cambia. Observar que se involucra a la función `get_remote_pdf`:

```
connect jorge/jorge@jrcbdd_s1
create or replace view revista as
  select q1.revista_id,q1.tipo,q1.nombre, q2.pdf as pdf
  from (
    select revista_id,tipo,nombre from revista_1
    unión all
    select revista_id,tipo,nombre from revista_2
  ) q1
  join (
    select * from table (get_remote_pdf)
  ) q2 on q1.revista_id = q2.revista_id;
```

1.5.1. Observaciones de la solución anterior

La técnica anterior tiene 2 observaciones muy importantes:

- Ventaja: Ofrece transparencia de distribución
- Desventaja: Desempeño. Como se puede apreciar en la definición de la función `get_remote_pdf`, la siguiente instrucción

```
insert into t_jrc_revista_3 select revista_id,pdf from revista_3;
```

provoca que se lean ***todos*** los registros del fragmento remoto mismo que contienen los valores `blob`. Esto representa un problema mayor de desempeño. Imaginar la siguiente consulta: `select * from revista where revista_id = 1;` A pesar de tener esta condición, la función `get_remote_pdf` obtendrá los N registros del fragmento remoto y por lo tanto se transmitirá el 100% de registros con los valores `blob` al sitio 1. ¡Si la tabla tiene 1,000,000 de datos `blob`, se transmitirá la misma cantidad, y en el resultado solo se mostrará 1 registro!

En este caso, el optimizador no puede detectar y cambiar la consulta que se hace al interior de una función.

¿Cómo solucionar lo anterior?

A continuación, se muestra una técnica que mejora el desempeño al obtener únicamente los valores blob que se necesitan en lugar de obtener el 100 % de los registros.

- Ventaja: Ofrece transparencia de distribución, disminuye el tráfico de red y cantidad de valores `blob` a transmitir, sólo los que se requieren.
- Desventaja: Si la cantidad de valores `blob` a obtener es grande, se requiere realizar una gran cantidad de peticiones remotas (pequeñas). Bajo esta situación, la solución anterior puede ser mejor.

1. La función ahora recibe el identificador o valor de la PK para recuperar un solo valor. Si el fragmento tuviera una PK compuesta, la función deberá recibir como parámetros todos los atributos que forman a la PK ya que se requiere identificar de manera única al registro que contiene el dato blob a recuperar.

```
connect jorge/jorge@jrcbdd_s1
create or replace function get_remote_pdf_by_id(v_revista_id in number ) return
blob is
pragma autonomous_transaction;
  v_temp_pdf blob;
begin
  --inicia txn autónoma 1
  --asegura que no haya registros
  delete from t_jrc_revista_3;
  --inserta un solo registro obtenido del fragmento remoto a la tabla temporal.
  insert into t_jrc_revista_3 select revista_id,pdf
     from revista_3 where revista_id = v_revista_id;
  --obtiene el registro de la tabla temporal y lo regresa como blob
  select pdf into v_temp_pdf
  from t_jrc_revista_3 where revista_id = v_revista_id;
  --elimina los registros de la tabla temporal una vez que han sido obtenidos.
  delete from t_jrc_revista_3;
  --termina txn autónoma 1.
  commit;
  return v_temp_pdf;
exception
  when others then
    rollback;
    raise;
end;
/
show errors;
```

Observar que en esta técnica solo se crea una transacción autónoma ya que no se genera una lista de objetos tipo `pdf_table`. De igual forma, se agrega un bloque de excepción para terminar la transacción en caso de que se produzca un error.

2. La definición de la vista ahora cambia a:

```
create or replace view revista as
  select q1.revista_id,q1.tipo,q1.nombre,
   get_remote_pdf_by_id(revista_id) as pdf
  from (
    select revista_id,tipo,nombre from revista_1
    union
    select revista_id,tipo,nombre from revista_2
  ) q1;
```

Observar que en este caso la función solo extrae el dato `blob` del fragmento remoto que se emplea para completar la definición de la tabla `revista`.

Ejemplo 2

Existe otro caso que pudiera presentarse en un esquema de fragmentación:

$$R_1 = \sigma_{tipo='A'}(R)$$
$$R_2 = \sigma_{tipo='A'}(R)$$

REVISTA

REVISTA_ID	NUMERIC(10,0)	NOT NULL
TIPO	CHAR(1)	NOT NULL
NOMBRE	VARCHAR(40)	NOT NULL
PDF	VARBINARY/BLOB(max)	NOT NULL

F_JRC_REVISTA_1

REVISTA_ID	NUMERIC(10,0)	NOT NULL
TIPO	CHAR(1)	NOT NULL
NOMBRE	VARCHAR(40)	NOT NULL
PDF	VARBINARY/BLOB(max)	NULL

F_JRC_REVISTA_2

REVISTA_ID	NUMERIC(10,0)	NOT NULL
TIPO	CHAR(1)	NOT NULL
NOMBRE	VARCHAR(40)	NOT NULL
PDF	VARBINARY/BLOB(max)	NULL

Observar que en este caso se tiene una fragmentación horizontal. En ambos sitios se encuentra una columna `blob`. La estrategia para manejar este escenario es similar a la anterior con algunas diferencias.

Notar que ahora los objetos `type` contienen todos los atributos del fragmento ya que es una fragmentación horizontal:

```
create type revista_type as object (
 revista_id number(10,0),
  tipo  char(1),
  nombre varchar2(40),
  pdf blob
);
/
show errors;
```

La tabla temporal también contiene todos los atributos del fragmento.

```
create global temporary table t_revista_2(
  revista_id number(10,0) constraint t_revista_2_pk  primary key,
  tipo  char(1) not null,
  nombre varchar2(40) not null,
  pdf blob not null
) on commit preserve rows;
```

La función para la estrategia 1 también contiene todos los atributos del fragmento.

```
-- función que implementa la estrategia 1
create or replace function get_remote_pdf return revista_table pipelined is
 pragma autonomous_transaction;
 v_temp_pdf blob;
 begin
   --inicia txn autónoma 1
   --asegura que no haya registros
   delete from t_revista_2;
   --inserta los datos obtenidos del fragmento remoto a la tabla temporal.
   insert into t_revista_2 select revista_id,tipo,nombre,pdf from revista_2;
   --termina txn autónoma 1 antes de generar la lista de datos
   commit;
   --obtiene los registros de la tabla temporal y los regresa
   --como objetos tipo revista_type
   for cur in (select revista_id,tipo,nombre,pdf from t_revista_2) loop
     pipe row(revista_type(cur.revista_id,cur.tipo,cur.nombre,cur.pdf));
   end loop;
   --inicia txn autónoma 2 para limpiar la tabla
   --elimina los registros de la tabla temporal una vez que han sido obtenidos.
   delete from t_revista_2;
   --termina txn autónoma 2
   commit;
   return;
```

```
 exception
   when others then
     rollback;
     raise;
 end;
 /
 show errors;
```

Lo mismo ocurre para la estrategia 2:

```
--función que implementa la estrategia 2
create or replace function get_remote_pdf_by_id(v_revista_id in number ) return
blob is
 pragma autonomous_transaction;
 v_temp_pdf blob;
 begin
   --asegura que no haya registros
   delete from t_revista_2;
   --inserta los datos obtenidos del fragmento remoto a la tabla temporal.
   insert into t_revista_2
     select revista_id,tipo,nombre,pdf
     from revista_2 where revista_id = v_revista_id;
   --obtiene el registro de la tabla temporal y lo regresa como blob
   select pdf into v_temp_pdf
   from t_revista_2 where revista_id = v_revista_id;
   --elimina los registros de la tabla temporal una vez que han sido obtenidos.
   delete from t_revista_2;
   commit;
   return v_temp_pdf;
 exception
   when others then
     rollback;
     raise;
 end;
 /
 show errors;
```

Observar que la definición de las vistas se agrega el operador `union all`

```
--creando la vista en s1 con la estrategia 1
Prompt creando vistas en S1 para acceso remoto de blobs
```

```
create or replace view revista_e1 as
  select revista_id,tipo,nombre,pdf
  from revista_1
  union all
  select revista_id,tipo,nombre,pdf from table (get_remote_pdf);

--creando la vista en s1 con la estrategia 2
create or replace view revista_e2 as
    select revista_id,tipo,nombre,pdf
    from revista_1
    union all
    select revista_id,tipo,nombre,get_remote_pdf_by_id(revista_id)
    from revista_2;
```

Es importante el uso de `union all`. Si se omite 'all' , el manejador intentará comparar los valores de todas las columnas para descartar duplicados. La consulta contiene columnas tipo `blob` las cuales no son válidas con operadores del álgebra relacional como son `intersect`, `union`, etc., ya que no está soportada la comparación binaria. El uso de `union all` no representa problema alguno ya que el esquema de fragmentación no permite duplicados.

1.5.2. Ejecución del ejemplo

Esta sección se puede omitir, pero se recomienda realizarla para tener una mejor comprensión de la sección anterior. En la carpeta compartida correspondiente a este ejercicio práctico,, se encuentran los siguientes archivos que contienen el código completo de estos 2 ejemplos, se recomienda revisarlos y ejecutarlos para comprobar su funcionamiento.

- `ejemplo-manejo-blobs-revistas-caso1.sql`
- `ejemplo-manejo-blobs-revistas-caso2.sql`
- `sample.pdf`

Para ejecutar estos scripts:

- Copiar los 3 archivos anteriores y el archivo `s-00-carga-blob-en-bd.sql` el cual fue creado en ejercicios prácticos anteriores en un mismo directorio.
- Abrir los 2 scripts (caso 1 y caso 2), cambiar la definición de las ligas para que correspondan con los nombres de servicio de la PDB donde serán ejecutados:

```
create database link jrcbdd_s2.fi.unam using 'JRCBDD_S2';
```

El único cambio que se debe realizar es modificar el texto que corresponde al nombre del servicio '`JRCBDD_S2`'. Notar que las ligas se llamarán `jrcbdd_s1` y `jrcbdd_s2`. Esto no representa problema alguno ya que lo importante es el nombre del servicio al que apuntan.

- Abrir una terminal y cambiarse a la carpeta donde se encuentran los archivos
- Ejecutar `sqlplus /nolog`

Posteriormente ejecutar :

```
start ejemplo-manejo-blobs-revistas-caso1.sql
start ejemplo-manejo-blobs-revistas-caso2.sql
```

1.6. Implementación de transparencia de fragmentación con datos blob

Empleando la estrategia mencionada en la sección anterior, crear scripts SQL `s-05-<SID>-soporte-blobs.sql,` un archivo por cada PDB que deberá conectarse a su PDB correspondiente y realizar las siguientes acciones:

1. Crear las vistas que corresponden a fragmentos con columnas `blob` que se encuentran en la misma PDB (locales). Estas vistas no requieren el uso de los objetos explicados en la sección anterior.
2. Crear los objetos `'type'` que serán empleados para las vistas que involucran columnas `blob` remotas.
3. Crear los objetos `'table'` que serán empleados para las vistas que involucran columnas `blob` remotas.
4. Crear tablas temporales requeridas para las vistas que involucran columnas `blob` remotas.
5. Crear las funciones que implementan la estrategia 1 para las vistas que involucran columnas `blob` remotas.
6. Crear las funciones que implementan la estrategia 2 para las vistas que involucran columnas `blob` remotas.
7. Empleando los objetos anteriores, definir las vistas que involucran blobs remotos empleando la estrategia 1, no olvidar usar el sufijo `_e1` , es decir: `<nombre_global>_e1`
8. Empleando los objetos anteriores, definir las vistas que involucran `blobs` remotos empleando la estrategia 2, no olvidar usar el sufijo `_e2` `<nombre_global>_e2`
9. Finalmente, crear un sinónimo que permita usar la vista con la estrategia 2 como solución por default. Es decir, el sinónimo tendrá como nombre el nombre global de la tabla, y apuntará a la vista que usa la estrategia 2:

```
create or replace synonym <nombre_global> for <nombre_global>_e2;
```

No olvidar crear un script para cada PDB, ya que cada PDB tiene requisitos diferentes.

```
Ejemplo:
--@Autor:           Jorge Rodriguez
--@Fecha creación:  dd/mm/yyyy
--@Descripción:     Definición de vistas para manejo de blobs en la PDB
jrcbdd_s2
```

```
Prompt connectando a jrcbdd_s2
connect <usuario>_bdd/<usuario>_bdd@jrcbdd_s2

prompt ---
Prompt Paso 1. creando vistas con columnas blob locales.
prompt ---

--- completar

prompt ---
Prompt Paso 2 creando objetos type para vistas  que involucran blobs remotos
prompt ---

--completar

prompt ---
Prompt Paso 3 creando objetos table para vistas  que involucran blobs remotos
prompt ---

--completar

prompt ---
Prompt Paso 4 creando tablas temporales para vistas  que involucran blobs
remotos
prompt ---

--completar

prompt ---
Prompt Paso 5 Creando funcion con estrategia 1 para vistas  que involucran blobs
remotos
prompt ---

--completar

prompt ---
Prompt Paso 6 Creando funcion con estrategia 2 para vistas  que involucran blobs
remotos
prompt ---

--completar

prompt ---
Prompt Paso 7 Crear las vistas con datos blob remotos empleando estrategia 1
```

```
prompt ---

--completar

prompt ---
Prompt Paso 8 Crear las vistas con datos blob remotos empleando estrategia 2
prompt ---

--completar

prompt ---
Prompt Paso 9 Crear un sinonimo con el nombre global del fragmento que apunte a
la estrategia 2.
prompt ---

--completar

Prompt Listo!
exit
```

C1. Incluir en el reporte únicamente el siguiente extracto de código tomado de alguna PDB.

- Código de la definición de una vista que requiere manejo de datos blob con estrategia 1
- Código de la definición de una vista que requiere manejo de datos blob remotos con estrategia 2

1.7. Consultas empleando transparencia de fragmentación

1.7.1. Conteo de registros

Para verificar el correcto funcionamiento del nivel de transparencia de fragmentación, realizar un conteo de registros similar al realizado en el ejercicio práctico 5, solo que, en esta ocasión, en lugar de emplear fragmentos y sus expresiones de reconstrucción, se deberán emplear las vistas creadas en la sección anterior.

Observar que ya no es necesario realizar consultas empleando mapeos locales, ni consultas empleando sinónimos (transparencia de localización). Las consultas ahora son idénticas a las de una base de datos centralizada.

Realizar las siguientes acciones:

- Generar una sentencia SQL que muestre el conteo de todos los registros de cada relación global en cada PDB Generar un script llamado `s-06-<iniciales>-consultas-fragmentacion.sql`
- El script deberá conectarse a ambas PDBs y ejecutar la consulta.

Ejemplo
Suponer las siguientes tablas globales:

```
idle> @s-06-jrc-consultas-fragmentacion.sql
conectando a sitio s1
Connected.
Realizando conteo de registros
      PAIS     BANCO    SUCURSAL   EMPLEADO     CUENTA MOVIMIENTO
---------- ---------- ---------- ---------- ---------- ----------
         x          x          x          x          x          x

conectando a jrcbdd_s2
Connected.
Realizando conteo de registros

      PAIS     BANCO    SUCURSAL   EMPLEADO   CUENTA MOVIMIENTO
---------- ---------- ---------- ---------- ---------- ----------
         x          x          x          x          x          3

listo.
```

- Se deberá obtener un solo registro por cada PDB
- Observar que se emplean las vistas creadas anteriormente las cuales hacen uso de los nombres globales.
    - Tip: emplear subqueries en la cláusula `select` (Similar al ejercicio práctico 5).
- ***C2. Incluir en el reporte*** únicamente el código de una consulta con fragmentación híbrida.

1.7.2. Consultas de transparencia de fragmentación para datos blob

Para realizar la validación bastará con obtener la longitud del dato binario. Asegurarse que todas las columnas de tipo `blob` no estén vacías. Estos datos fueron inicializados en el ejercicio práctico anterior. Crear un script llamado `s-07-<iniciales>-consulta-blobs.sql`

El archivo contendrá instrucciones `select` para mostrar los valores de las PKs y las longitudes de los datos `blob`. El script se deberá conectar a cada PDB y ejecutar la misma consulta:

- Para la PDB que requiere accesos remotos se tendrán 3 consultas:
    - Consulta con la estrategia 1
    - Consulta con la estrategia 2
    - Consulta con el sinónimo extra creado anteriormente (sinónimo que apunta a la estrategia 2)
- Si la PDB no requiere acceso remoto (no fue necesario crear objetos type, table y tablas temporales) solo se tendría una consulta:
    - Consulta que hace uso de la vista (nombre global).

Ejemplo

Suponer que una entidad llamada `revista` tiene una columna `blob` por lo que se crearon las vistas `revista_e1` (estrategia 1), `revista_e2` (estrategia 2) y el sinónimo revista en el sitio s1. El sitio s2 no requiere de accesos remotos para obtener los objetos `blob`.

```
--@Autor:           Jorge Rodriguez
--@Fecha creación:  dd/mm/yyyy
--@Descripción:     Consultas para validar vistas con columnas BLOB

Prompt Conectandose a jrcbdd_s1
connect <usuario>_bdd/<usuario>_bdd@jrcbdd_s1

Prompt revista estrategia 1
select revista_id,dbms_lob.getlength(pdf) as longitud
from revista_e1;

Prompt revista estrategia 2
select revista_id,dbms_lob.getlength(pdf) as longitud
from revista_e2;

Prompt revista, uso de sinonimo
select revista_id,dbms_lob.getlength(pdf) as longitud
from revista;

-- completar con otras tablas que contengan datos BLOB.

Prompt Conectandose a jrcbdd_s2
connect <usuario>_bdd/<usuario>_bdd@jrcbdd_s2

-- En este sitio solo se usa la vista ya que el dato BLOB
-- se encuentra de forma local.
Prompt revista uso de la vista (acceso local al dato BLOB)
select revista_id,dbms_lob.getlength(pdf) as longitud
from revista;

Prompt Listo!
exit
```

1.8. Validación de resultados

En esta actividad se realizará la validación de las respuestas del ejercicio anterior. Para ello, realizar las siguientes acciones:

- En la carpeta compartida de este ejercicio práctico, obtener todos los scripts sql/plb.
- Editar el script `s-08-validador-main.sql` con los valores correspondientes

- En una nueva terminal cambiarse al directorio donde se encuentran los scripts, y ejecutar el script editado (no se requiere emplear al usuario Oracle).

```
sqlplus /nolog
start s-08-validador-main.sql
```

- En caso de existir errores, revisar y leer cuidadosamente los mensajes de error, corregir y reintentar. ***C3. Incluir en el reporte*** la captura de pantalla con la salida del validador.

1.9. Contenido de la entrega

- Elementos comunes. Para mayores detalles revisar el documento de instrucciones generales para realizar ejercicios prácticos
- C1. Código SQL que contiene la definición de una vista que requiere el manejo de datos `blob` con estrategia 1 y 2.
- C2. Código SQL para una consulta empleando las vistas con fragmentación híbrida
- C3. Salida de ejecución del script de validación.