Implementando Transaction Guard con ODP.NET 12c

Por Francisco Riccio

Introducción

Nuestras aplicaciones transaccionales constantemente envían transacciones a la base de datos, pero que sucedería si al momento de confirmar las operaciones que componen nuestra transacción se presenta un problema en la comunicación de red ó una caída en la instancia de base de datos que se encuentra conectada la aplicación.

Basado en el escenario presentado, podemos tener las siguientes situaciones:

- La base de datos confirmó el cambio pero la aplicación no pudo obtener el mensaje de confirmación.
- El incidente ocurrió antes que se pudiera realizar la confirmación de la transacción.

En ambos puntos, la aplicación tiene la duda si debe volver a relanzar la transacción y si lo hace, se corre el riesgo de duplicar información.

Para evitar estos escenarios y el programa pueda tener la información necesaria si debe o no relanzar su transacción nuevamente, se debe implementar Transaction Guard; el cual es un nuevo feature de Oracle Database 12c y disponible únicamente en edición Enterprise Edition.

Oracle Database 12c introduce un nuevo concepto llamado Logical Transaction Identifier (LTXID), el cual es un código único generado al inicio de una transacción para una conexión de base de datos específica. Al generarse un LTXID en el servidor de base de datos, este es devuelto a la aplicación cliente y en caso de un incidente con la instancia de base de dato donde la aplicación se encuentra conectada, el código de la aplicación podrá consultar el LTXID después de haber hecho el failover de conexión y saber el estado de su última transacción. Con dicha información, la aplicación tomará la decisión de replicar de nuevo la última transacción que estuvo pendiente.

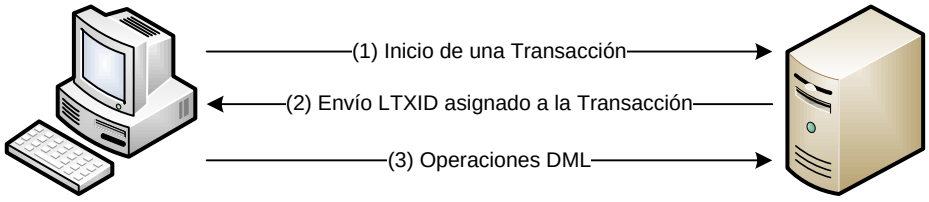


Figura 1

La figura 1 muestra como un LTXID es generado de manera única por conexión de base de datos por cada transacción que se inicia y automáticamente es enviado a la aplicación cliente para su conocimiento.

Los LTXID generados por la base de datos son almacenados en la tabla SYS. LTXID_TRANS y serán retenidos durante 24 horas (siendo 30 días la máxima retención) posterior al COMMIT. Esta tabla se encuentra en el tablespace SYSAUX y se encuentra particionada. La cantidad de particiones que se crean sobre esta tabla es igual a la cantidad de instancias que forman la base de datos del Oracle RAC. Cuando añadimos una nueva instancia a nuestra base de datos Oracle RAC se crea una nueva partición sobre la tabla.

En la implementación que más adelante se detallará, se desarrolló sobre una base de datos Oracle RAC 12c que se conforma de 2 instancias de base de datos. Basado en esta configuración, podremos ver que la tabla TXID_TRANS tiene 2 particiones:

```
SQL> select PARTITION_NAME from dba_tab_partitions where TABLE_NAME='LTXID_TRANS';

PARTITION_NAME
--------------------------------------------------------------------------------
LTXID_TRANS_1
LTXID_TRANS_2
```

Podemos mover las particiones a otros tablespace si lo vemos conveniente. Este procedimiento es completamente válido con el siguiente script.

```
alter TABLE LTXID_TRANS move partition LTXID_TRANS_# tablespace <nombre_tablespace>;
```

Transaction Guard soporta los siguientes escenarios:

- Transacciones locales.
- Operaciones DML & DCL.
- Transacciones distribuidas.
- Transacciones remotas.
- Transacciones con paralelismo.
- AUTO-COMMIT configurado en el lado de la aplicación cliente.
- Disponible para los drivers 12c: JDBC Thin, OCI y ODP.NET (Unmanaged driver, más información: http://docs.oracle.com/cd/E16655_01/win.121/e17732/intro004.htm#ODPNT8147).

Escenarios excluidos:

- Autónomas transacciones (PRAGMA AUTONOMOUS_TRANSACTION).
- Transacciones mediante el estándar XA.

- Replicación a GoldenGate y Standby Database Lógicos.
- Active Dataguard con dblinks de escritura/lectura para transacciones de reenvío.

Transaction Guard requiere como requisito que nuestra aplicación sea capaz de recibir eventos Fast Application Notification (FAN).

Nota 1: Existe un feature de Oracle Database 12c llamado Application Continuity el cual está bien relacionado con Transaction Guard. Application Continuity permite a una aplicación automáticamente disparar nuevamente la transacción que fue conocida por la aplicación como fallida gracias a Transaction Guard.

Nota 2: La versión Oracle Client 12.1 no tiene implementado aún Application Continuity para ODP.NET posiblemente lo tendremos en la versión Oracle Client 12.2.

Nota 3: La habilitación de Transaction Guard en promedio incrementa el uso del CPU en menos de 1%, siendo imperceptible.

Implementación - Transaction Guard

Nuestra aplicación de ejemplo permitirá registrar productos. Si en caso ocurriera alguna una incidencia con la instancia de base de datos donde se encuentra conectada nuestra aplicación, esté será capaz de reenviar nuevamente la operación fallida sin intervención del usuario final.

Nuestra base de datos está en una versión Oracle RAC 12.1 sobre una plataforma Oracle Linux 5.10 x64 bits.

I) Base de Datos

a) Nuestra base de datos debe presentar un servicio con las opciones COMMIT_OUTCOME y RETENTION.

```
[oracle@srv1-121 ~]$ srvctl modify service -d PRD -s PROD -commit_outcome TRUE -retention 86400
[oracle@srv1-121 ~]$ srvctl config service -d PRD -s PROD
Service name: PROD
Service is enabled
Server pool: PRD_PROD
Cardinality: 2
Disconnect: false
Service role: PRIMARY
Management policy: AUTOMATIC
DTP transaction: false
AQ HA notifications: true
Global: false
Commit Outcome: true
Failover type:
Failover method:
TAF failover retries:
TAF failover delay:
Connection Load Balancing Goal: SHORT
Runtime Load Balancing Goal: SERVICE_TIME
TAF policy specification: NONE
Edition:
Pluggable database name:
Maximum lag time: ANY
SQL Translation Profile:
Retention: 86400 seconds
Replay Initiation Time: 300 seconds
Session State Consistency:
Preferred instances: PRD1,PRD2
Available instances:
```

COMMIT_OUTCOME: Determina si los LTXID serán registrados.

RETENTION: Determina el tiempo que un LTXID será almacenado en la base datos después que su transacción asociada haya sido confirmada.

Nota 1: No podemos utilizar el servicio por default que está configurado al parámetro DB_NAME o DB_UNIQUE_NAME.

Nota 2: Es importante que el atributo AQ HA Notifications lo tengamos con el valor de true. Este parámetro vino como configuración en la recepción de eventos FAN, el cual es requerido por Transaction Guard. En caso no estuviera en true, debemos ejecutar el siguiente comando:

```
[oracle@srv1-121 ~]$ srvctl modify service -d PRD -s PROD -q true
```

Posteriormente lo siguiente:

```
execute DBMS_AQADM.GRANT_QUEUE_PRIVILEGE('DEQUEUE','SYS.SYS$SERVICE_METRICS',

'<NOMBRE_USUARIO_BD> ');
```

b) Garantizamos permisos al usuario de base de datos sobre el paquete DBMS_APP_CONT.

El paquete DBMS_APP_CONT ofrece las interfaces para determinar si una transacción pudo ser confirmada en la base de datos.

```
[oracle@srv1-121 ~]$ sqlplus / as sysdba

SQL*Plus: Release 12.1.0.1.0 Production on Mon Feb 10 22:59:48 2014

Copyright (c) 1982, 2013, Oracle.  All rights reserved.


Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.1.0 - 64bit Production
With the Partitioning, Real Application Clusters, Automatic Storage Management, OLAP,
Advanced Analytics and Real Application Testing options

SQL> grant execute on DBMS_APP_CONT to friccio;

Grant succeeded.
```

II) Aplicación

a) Creamos la tabla que almacenará los productos registrados por la aplicación.

```
SQL> connect friccio/oracle
Connected.
SQL> create table PRODUCTO (id number, descripcion varchar(30), pu number, stockActual number);

Table created.

SQL> create sequence SEQ_PRODUCTO;

Sequence created.
```

b) Código .NET

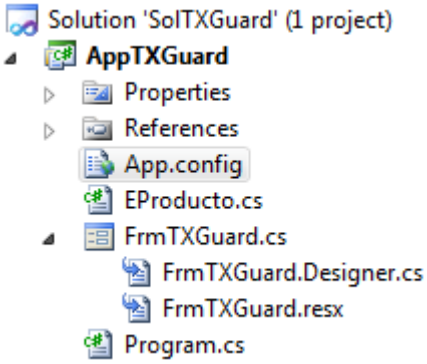
b.1) Cadena de Conexión.

```
<?xml version="1.0" encoding="utf-8" ?>

<configuration>

 <appSettings>

  <add key="CONEXION" value="Data Source=(DESCRIPTION = (ADDRESS_LIST = (ADDRESS = (PROTOCOL = TCP)(HOST = srvscan-121.riccio.com)(PORT = 1521)))(FAILOVER=YES)(LOAD_BALANCE=YES)(CONNECT_DATA=(SERVICE_NAME = PROD)(SERVER = DEDICATED)));User Id=friccio;Password=oracle;Pooling=true;Min Pool Size=1;Max Pool Size=1;HA Events=true;Validate Connection=true"/>

 </appSettings>

</configuration>
```

Es importante que tengamos definido las propiedades: Pooling y HA Events que permitirán habilitar la recepción de eventos FAN.

b.2) Interfaz.

b.3) Proyecto



b.4) Código

Clase EProducto

```
namespace AppTXGuard
{
    class EProducto
    {

        public int id { get; set; }
        public String descripcion { get; set; }
        public double pu { get; set; }
        public double stockActual { get; set; }

    }

}
```

Click en el Botón Registrar:

```
private void registrarProducto(ref EProducto pProducto, OracleConnection pcn){
    String SQL = "insert into PRODUCTO values (SEQ_PRODUCTO.NEXTVAL,:DESCRIPCION,:PU,:STOCKACTUAL)";
    SQL = " returning id into :PID";
    OracleCommand cmd = pcn.CreateCommand();
    cmd.CommandText = SQL;
    cmd.Parameters.Add("DESCRIPCION", OracleDbType.Varchar2).Value = pProducto.descripcion;
    cmd.Parameters.Add("PU", OracleDbType.Double).Value = pProducto.pu;
    cmd.Parameters.Add("STOCKACTUAL", OracleDbType.Double).Value = pProducto.stockActual;
    cmd.Parameters.Add("PID", OracleDbType.Int32, ParameterDirection.ReturnValue);
    cmd.ExecuteNonQuery();
    pProducto.id = Convert.ToInt32(cmd.Parameters[3].Value.ToString());
    cmd.Dispose();
}
```

```
private void btnregistrar_Click(object sender, EventArgs e)
{
    EProducto objProducto = new EProducto();
    objProducto.descripcion = txtdescripcion.Text;
    objProducto.pu = Convert.ToDouble(txtpu.Text);
    objProducto.stockActual = Convert.ToDouble(txtstockactual.Text);
    string conexion = System.Configuration.ConfigurationManager.AppSettings["CONEXION"].ToString();
    OracleConnection cn = null;
    cn = new OracleConnection(conexion);
    cn.Open();
    if (cn.State == ConnectionState.Open)
    {
        byte[] ltxid;
        OracleLogicalTransactionStatus estado = null;
        OracleTransaction tx = cn.BeginTransaction();
        registrarProducto(ref objProducto,cn);
        try
        {
            tx.Commit();
        }
        catch
        {
            ltxid = cn.LogicalTransactionId;
            if (ltxid != null)
            {
                cn = new OracleConnection(conexion);
                cn.Open();
                estado = cn.GetLogicalTransactionStatus(ltxid);
            }
            if ((estado != null) && (estado.Committed == false))
            {
                tx = cn.BeginTransaction();
                registrarProducto(ref objProducto, cn);
                tx.Commit();
            }
        }
        finally
        {
            cn.Close();
            cn.Dispose();
        }
        MessageBox.Show("Producto registrado, con codigo = " + objProducto.id.ToString(),
            "Registro satisfactorio",MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
}
```

<OracleConnection>.LogicalTransactionId, devuelve el LTXID que se compone por un arreglo de enteros.

cn.GetLogicalTransactionStatus(<OracleLogicalTransactionStatus >), devuelve el estado de la última transacción enviada a la base de datos.

La clase OracleLogicalTransactionStatus tiene 2 propiedades que son de interés para nuestra aplicación con la finalidad de definir una acción concreta:

- Propiedad Committed, devolverá true o false si la instancia de base de datos pudo confirmar la transacción.
- Propiedad UserCallCompleted, indicará si la aplicación no pudo conseguir un resultado esperado. Resultados como: bind variables de retorno ó número de filas modificadas, etc.

En nuestro caso hemos tomado la decisión de relanzar la transacción en caso la transacción no se pudo confirmar en la base de datos.

Conclusiones

Durante versiones previas a Oracle Database 12c, teníamos a disposición para nuestras aplicaciones mecanismos de failover de conexión de base de datos e inclusive continuidad de nuestras operaciones SELECT en un escenario de faliover ocasionado por algún incidente que hubiera ocurrido.

Ahora con Transaction Guard podemos dar continuidad a nuestras operaciones DML & DDL & DCL, permitiendo un continuidad completa a la aplicación sin el temor de duplicar información, evitando largas líneas de código personalizado de los desarrolladores para obtener el mismo objetivo, donde dichos códigos generaban un re-trabajo a la base de datos perjudicando su desempeño.

Publicado por Ing. Francisco Riccio. Es un IT Architect en IBM Perú e instructor de cursos oficiales de certificación Oracle. Está reconocido por Oracle como un Oracle ACE y certificado en productos de Oracle Application & Base de Datos.

e-mail: francisco@friccio.com

web: www.friccio.com