Cuarentena a Planes de Ejecución Perjudiciales - Oracle Database 19c

Por Francisco Riccio

Introducción

Oracle Database 19c nos provee una nueva funcionalidad que permite bajo ciertos parámetros establecidos aislar un plan de ejecución de una sentencia SQL ya identificada como perjudicial para evitar su ejecución. Esta funcionalidad se puede integrar con Resource Manager para evitar correr planes de ejecución donde ya se identificaron que son perjudiciales en el pasado.

La implementación de la nueva funcionalidad puede ser realizada de manera explícita a través del paquete DBMS_SQLQ o automáticamente a través de Resource Manager cuando una sentencia SQL es terminada por algún criterio establecido, ambos métodos definen si una sentencia SQL o un plan de ejecución deben estar en cuarentena por defecto por 53 semanas.

Implementación

A. Método Explícito a través del paquete DBMS_SQLQ

Las pruebas realizadas serán hechas sobre una tabla llamada TEST la cual cuenta con 20 millones de filas.

Se ejecutará una consulta SQL para posteriormente definirla a cuarentena, esto a través de 3 opciones:

Sentencia SQL:

```
SQL> select * from table(dbms_xplan.display_cursor());

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------
SQL_ID  dwmg00cvtnhz2, child number 0
-------------------------------------
select count(*) from test t1, test t2

Plan hash value: 4067767418

-------------------------------------------------------------------------------
| Id  | Operation             | Name | Rows  | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |      |       |    76G(100)|          |
|   1 |  SORT AGGREGATE       |      |     1 |            |          |

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------
|   2 |   MERGE JOIN CARTESIAN|      |   105T|    76G   (1)|834:46:55 |
|   3 |    TABLE ACCESS FULL  | TEST |    10M|  7482    (1)| 00:00:01 |
|   4 |    BUFFER SORT        |      |    10M|    76G   (1)|834:46:54 |
|   5 |     TABLE ACCESS FULL | TEST |    10M|  7480    (1)| 00:00:01 |
-------------------------------------------------------------------------------

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)


21 rows selected.
```

Sentencia SQL: select count(*) from test t1, test t2;

SQL ID: dwmg00cvtnhz2

PLAN HASH: 4067767418

<u>Opción 1</u>: Configurando en Cuarentena una sentencia SQL y todos sus planes de ejecución a través de su SQL ID

<u>Función</u>: *DBMS_SQL.CREATE_QUARANTINE_BY_SQL_ID*

```
SQL> set serveroutput on
declare
 v_nombre VARCHAR2(30);
begin
 v_nombre:=DBMS_SQLQ.CREATE_QUARANTINE_BY_SQL_ID(SQL_ID =>'dwmg00cvtnhz2');
 dbms_output.put_line(v_nombre);
end;
/
SQL_QUARANTINE_cyvnzyrck69bm

PL/SQL procedure successfully completed.
```

Si ejecutamos nuevamente la consulta obtendremos el siguiente mensaje:

```
SQL> select count(*) from test t1, test t2;
select count(*) from test t1, test t2
                      *
ERROR at line 1:
ORA-56955: quarantined plan used
```

<u>Opción 2</u>: Configurando en Cuarentena una sentencia SQL y todos sus planes de ejecución a través de la codificación literal de la sentencia

<u>Función</u>: *DBMS_SQL.CREATE_QUARANTINE_BY_SQL_TEXT*

```
SQL> declare
 v_nombre VARCHAR2(30);
begin
 v_nombre:=DBMS_SQLQ.CREATE_QUARANTINE_BY_SQL_TEXT(
   SQL_TEXT => to_clob('select * from test'));
end;
/

PL/SQL procedure successfully completed.
```

<u>Opción 3</u>: Configurando en Cuarentena un plan de ejecución específico para una sentencia SQL

<u>Función</u>: *DBMS_SQL.CREATE_QUARANTINE_BY_SQL_ID o DBMS_SQL.CREATE_QUARANTINE_BY_SQL_TEXT*

```
SQL> set serveroutput on
declare
 v_nombre clob;
begin
 v_nombre:=DBMS_SQLQ.CREATE_QUARANTINE_BY_SQL_ID
   (SQL_ID=>'dwmg00cvtnhz2',PLAN_HASH_VALUE=>'4067767418');
 dbms_output.put_line(v_nombre);
end;
/
SQL_QUARANTINE_cyvnzyrck69bmf275347a

PL/SQL procedure successfully completed.
```

A diferencia de las opciones previas, esta última opción permitirá la ejecución de la sentencia SQL siempre y cuando el plan de ejecución escogido por el optimizador no esté en cuarentena.

Resource Manager

A través de Resource Manager se puede limitar los recursos que podrán ser consumidos por una sentencia SQL e inclusive cancelarla.

A continuación, se muestra una configuración de Resource Manager la cual servirá para integrarla con esta nueva funcionalidad:

```
--Creando grupos consumidores
execute dbms_resource_manager.create_pending_area();
execute DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP('GRUPO_ORCL','Grupo de Prueba');
execute dbms_resource_manager.submit_pending_area;
```

```
--Asignado mapeos de usuarios al grupo consumidor creado
execute dbms_resource_manager.create_pending_area();
execute dbms_resource_manager_privs.grant_switch_consumer_group('FRICCIO','GRUPO_ORCL',false);
execute dbms_resource_manager.set_consumer_group_mapping(
 attribute=>DBMS_RESOURCE_MANAGER.oracle_user,value=>'FRICCIO',consumer_group=>'GRUPO_ORCL');
execute dbms_resource_manager.submit_pending_area;
```

```
--Creando planes
execute dbms_resource_manager.create_pending_area();
execute dbms_resource_manager.create_plan(plan=>'PLAN_ORCL',comment=>'Plan de Ejemplo');
--Creando directivas
execute dbms_resource_manager.create_plan_directive(plan=>'PLAN_ORCL',group_or_subplan=>'GRUPO_ORCL',
        switch_group=>'CANCEL_SQL',switch_time=>300,mgmt_p1=>50);
execute dbms_resource_manager.create_plan_directive(plan=>'PLAN_ORCL',group_or_subplan=>'OTHER_GROUPS',mgmt_p1=>50);
execute dbms_resource_manager.validate_pending_area();
execute dbms_resource_manager.submit_pending_area;
execute dbms_resource_manager.clear_pending_area;
```

```
--Activando el plan.
alter system set resource_manager_plan='PLAN_ORCL';
```

El plan configurado de Resource Manager permitirá que el usuario FRICCIO no pueda consumir más de 50% de procesamiento de los cores asignados a la base de datos considerando un 100% de consumo. Además estará limitado a ejecutar consultas no mayores a 300 segundos (5 minutos), en caso contrario su consulta será anulada.

Validamos el plan definido en Resource Manager con la nueva funcionalidad de Cuarentena:

```
SQL> select count(*) from test t1, test t2;
select count(*) from test t1, test t2
                      *
ERROR at line 1:
ORA-00040: active time limit exceeded - call aborted


SQL> select sql_text from dba_sql_quarantine;

SQL_TEXT
--------------------------------------------------
select count(*) from test t1, test t2

SQL> select count(*) from test t1, test t2;
select count(*) from test t1, test t2
                      *
ERROR at line 1:
ORA-56955: quarantined plan used
```

Como se puede apreciar, la sentencia SQL fue cancelada por Resource Manager al consumir más de 5 minutos y por ende es registrado para estar en cuarentena. Al volver a ejecutar la sentencia y usar el mismo plan de ejecución está es evitada.

C. Operaciones sobe las Cuarentenas

C1. Establecer umbrales sobre cuarentenas creadas.

Es posible crear umbrales basado en los siguientes recursos:

- CPU Time (segundos)
- Elapsed Time (segundos)
- I/O MB
- # Physical Reads
- # Logical Reads

En el siguiente ejemplo se modificará la cuarentena previamente creada para definir un umbral que defina la ejecución de la consulta SQL a 30 segundos.

La implementación se realizará a través del procedimiento: *DBMS_SQLQ.ALTER_QUARENTINE*

```
SQL> begin
 DBMS_SQLQ.ALTER_QUARANTINE(QUARANTINE_NAME=>'SQL_QUARANTINE_cyvnzyrck69bmf275347a',
 PARAMETER_NAME=>'ELAPSED_TIME',PARAMETER_VALUE => '30');
end;
/

PL/SQL procedure successfully completed.
```

Las constantes son:

CPU_TIME	ELAPSED_TIME	IO_MEGABYTES	IO_REQUESTS
IO_LOGICAL	ENABLED	AUTOPURGE	

Las constantes:

- ENABLED = Habilita o deshabilita la configuración de la cuarentena.
- AUTOPURGE = Habilita o deshabilita la eliminación de la cuarentena. Por defecto está activada y su periodo de eliminación de una cuarentena es de 53 semanas.

Nota: Si se crea o modifica una política de Resource Manager y se define un tiempo menor o igual al definido en el umbral, la sentencia SQL no se ejecutará, en caso contrario, se ejecutará con la finalidad de tener una probabilidad de finalizar la consulta a un tiempo menor al establecido en el plan de Resource Manager.

C2. Exportar cuarentenas.

La exportación se ejecuta a través del procedimiento:
DBMS_SQLQ.CREATE_STGTAB_QUARENTINE

```
SQL> execute DBMS_SQLQ.CREATE_STGTAB_QUARANTINE(STAGING_TABLE_NAME=>'TABLA_CUARENTENA');

PL/SQL procedure successfully completed.
```

La herramienta creará una tabla llamada TABLA_CUARENTENA y lo almacenará en el tablespace del Usuario quien ejecutó el procedimiento además de colocarlo como owner.

Añadida la tabla en la nueva base de datos se ingresa las cuarentenas a través de la función *DBMS_SQLQ.PACK_STGTAB_QUARANTINE.*

```
SQL> DECLARE
       configuracion NUMBER;
     BEGIN
       configuracion:=DBMS_SQLQ.PACK_STGTAB_QUARANTINE(
                        STAGING_TABLE_NAME=>'TABLA_CUARENTENA',
                        NAME=>'SQL_QUA%');
     END;
     /

PL/SQL procedure successfully completed.
```

Poblada la tabla se exporta e importa hacia la nueva base de datos y se inicia el proceso de carga de cuarentenas a través de la función *DBMS_SQLQ.UNPACK_STGTAB_QUARANTINE.*

```
SQL> DECLARE
  configuracion NUMBER;
BEGIN
  configuracion:=DBMS_SQLQ.UNPACK_STGTAB_QUARANTINE(
                   STAGING_TABLE_NAME=>'TABLA_CUARENTENA');
END;
/

PL/SQL procedure successfully completed.
```

C3. Listar las cuarentenas creadas.

Las cuarentenas creadas pueden ser consultas a través de la vista: *DBA_SQL_QUARANTINE*

```
SQL> desc DBA_SQL_QUARANTINE;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------
 SIGNATURE                                 NOT NULL NUMBER
 NAME                                      NOT NULL VARCHAR2(128)
 SQL_TEXT                                  NOT NULL CLOB
 PLAN_HASH_VALUE                                    NUMBER
 CPU_TIME                                           VARCHAR2(4000)
 IO_MEGABYTES                                       VARCHAR2(4000)
 IO_REQUESTS                                        VARCHAR2(4000)
 ELAPSED_TIME                                       VARCHAR2(4000)
 IO_LOGICAL                                         VARCHAR2(4000)
 CREATOR                                            VARCHAR2(128)
 ORIGIN                                             VARCHAR2(16)
 DESCRIPTION                                        VARCHAR2(500)
 CREATED                                   NOT NULL TIMESTAMP(6)
 LAST_EXECUTED                                      TIMESTAMP(6)
 ENABLED                                            VARCHAR2(3)
 AUTOPURGE                                          VARCHAR2(3)
```

Nota 1: En la vista *V$SQL* se cuentan con los campos SQL_QUARANTINE y AVOIDED_EXECUTIONS que podrían dar información adicional sobre el número de veces que la consulta fue prevenida de ejecutar.

Nota 2: Es posible revisar la configuración de los umbrales a través de la función *DBMS_SQLQ.GET_PARAM_VALUE_QUARANTINE*

C4. Eliminación de una cuarentena

La eliminación se realiza a través del procedimiento: *DBMS_SQLQ.DROP_QUARANTINE*.

```
SQL> execute DBMS_SQLQ.DROP_QUARANTINE('SQL_QUARANTINE_cyvnzyrck69bm66a077fd');

PL/SQL procedure successfully completed.
```

Conclusión

Esta nueva funcionalidad provista en Oracle Database 19c permite complementar el beneficio de Resource Manager para controlar los recursos del servidor extendiéndolo a prevenir planes de ejecución y sentencias SQL ya identificadas como peligrosas bajo experiencias previas y así mantener una estabilidad adecuada en la base de datos.

Publicado por:

Francisco Riccio, actualmente se desempeña como Arquitecto de Soluciones en Oracle Perú y es instructor de cursos oficiales de certificación Oracle. Es un Oracle Certified Professional en productos de Oracle Application, Base de Datos, Cloud & Virtualización.