On the Fifth Day of 18c/19c, Viscosity Gave to me...

SQL Quarantine

December 14, 2020

A highly anticipated new feature of Oracle Database 19c is SQL Quarantine. Before we begin, it is important that customers are aware that this feature is only available on Oracle Exadata and Autonomous Databases on Oracle Cloud or Cloud at Customer. As the name would imply, this feature adds a new tool for monitoring and managing runaway queries. In the past you may have implemented a cron job to monitor long running queries or utilized Enterprise Manager to alert on these events. Now, we can simply set up the initial configuration and enable Oracle to handle the issue for us.

Complete Preinstallation Tasks

In Oracle Database 18c, Oracle introduced a new feature to provide DBAs the ability to cancel a query (I.e. kill the offending query) and not the session. Leveraging the ALTER SYSTEM command, we can pass the arguments to cancel a query, allowing the session to continue to process. Many times, for adhoc users, we do not want to kill the offending user issuing the SELECT statement, but just cancel the query. The syntax to cancel a SQL statement is:

```
ALTER SYSTEM CANCEL SQL 'SID, SERIAL[, @INST_ID][, SQL_ID]';
```

A common use case for leveraging this feature would be to cancel a SQL in a session on the current instance or on the remote RAC instance. The syntax would look like this:

```
ALTER SYSTEM CANCEL SQL '921, 99210';
```

To cancel the SQL in session on the remote RAC instance (INST_ID = 2)

```
ALTER SYSTEM CANCEL SQL '921, 99210, @2';
```

Going back to our topic on SQL Quarantine, when Resource Manager (DBRM) detects SQL is exceeding resources or run time, the SQL execution plan is quarantined, and further attempts to run this plan will cause it to be terminated immediately prior to execution. If any of the Resource Manager thresholds is equal to or less than a quarantine threshold specified in a SQL statement's quarantine configuration, then the SQL statement is not allowed to run. Previous versions allowed sql to run again and again until it hit a resource limit, then it was terminated and allowed to run again.

Technology Overview

Database: 18c / 19c

Company: Oracle Corporation

Topic: SQL Quarantine

Viscosity can help with any of your Database Upgrade needs

Viscosity has performed numerous zero-downtime database migrations and upgrades over the years and has a proven track record with business critical and mission critical databases.

Viscosity's Database Migration & Upgrade Services can plan, upgrade, validate and migrate all database content - quickly and effectively with our automated approach and proven methodology.

Learn more about how you can maintain and maximize your investments at viscosityna.com or email us at hello@viscosityna.com.

www.viscosityna.com

Steps to Enable SQL Quarantine

In order to activate SQL Quarantine, you must first do some configuration. At a very high level, this is what you will need to setup and test SQL Quarantine:

- Create a default resource plan using DBRM that limits the SQL execution time to a specific number of seconds. SQL Statements that exceed this limit, run longer than X seconds, will be terminated.
- Run a SQL statement that exceeds this limit, X seconds.
- Quarantine the SQL statement using DBMS_SQLQ package. You can check the DBA_SQL_QUARANTINE view to see quarantined SQL statements.
- Run the SQL statement again. If the statement runs with the same execution plan, then the statement is terminated right away with: ORA-56955 quarantined plan used.

As we mentioned above, the SQL Quarantine feature is only available on the Exadata platform. To take advantage of this feature for testing purposes on non-engineered systems, we can leverage an underscore parameter called _exadata_feature_on:

```
sqlplus / as sysdba <<EOF

alter system set "_exadata_feature_on"=true scope=spfile;

shutdown immediate;

startup;

exit;

EOF
```

Enable SQL Quarantine

To enable SQL Quarantine, we leverage the DBMS_SQLQ package:

```
BEGIN
DBMS_SQLQ.ALTER_QUARANTINE(
QUARANTINE_NAME => '&SQL_QUARANTINE_ID,
PARAMETER_NAME  => 'ENABLED',
PARAMETER_VALUE => 'YES');
END;
/
```

Create a Quarantine for an execution plan and sql id.

```
DECLARE
```

```
    quarantine_config VARCHAR2(30);
BEGIN
    quarantine_config := DBMS_SQLQ.CREATE_QUARANTINE_BY_SQL_ID(
                          SQL_ID => '8vu7s907prbgr',
                          PLAN_HASH_VALUE => '3488063716');
END;
/
```

Note: If a PLAN_HASH_VALUE is null then quarantine applies to all executions of the sql_id.

After creating a quarantine configuration for an execution plan of a SQL statement, you can specify quarantine thresholds for it using the DBMS_SQLQ.ALTER_QUARANTINE procedure.

Quarantine threshold options:

- CPU time
- Elapsed time
- I/O in megabytes
- Number of physical I/O requests
- Number of logical I/O requests

Set threshold example:

```
BEGIN
    DBMS_SQLQ.ALTER_QUARANTINE(
       QUARANTINE_NAME => 'SQL_QUARANTINE_1m02zerop8o2ytm45',
       PARAMETER_NAME  => 'CPU_TIME',
       PARAMETER_VALUE => '5');

    DBMS_SQLQ.ALTER_QUARANTINE(
       QUARANTINE_NAME => 'SQL_QUARANTINE_1m02zerop8o2ytm45',
       PARAMETER_NAME  => 'ELAPSED_TIME',
       PARAMETER_VALUE => '10');
END;
/
```

Validate Quarantine Configuration

```
DECLARE
   quarantine_config_setting_value VARCHAR2(30);
BEGIN
   quarantine_config_setting_value :=
      DBMS_SQLQ.GET_PARAM_VALUE_QUARANTINE(
          QUARANTINE_NAME => 'SQL_QUARANTINE_1m02zerop8o2ytm45',
          PARAMETER_NAME  => 'CPU_TIME');
END;
/
```

Note: Also query the view SQl> select * from DBA_SQL_QUARANTINE;

Drop Quarantine Configuration

Any unused quarantine configurations are automatically purged or deleted after 53 weeks, but you can manually drop the SQL quarantine configuration for a SQL statement.

```
BEGIN
   DBMS_SQLQ.DROP_QUARANTINE('SQL_QUARANTINE_1m02zerop8o2ytm45');
END;
/
```

- A quarantine configuration is specific to an execution plan for a SQL statement. If two different SQL statements use the same execution plan, they do not share the same quarantine configuration.
- If there is no quarantine configuration created for an execution plan of a SQL statement, or if no quarantine thresholds are specified in its quarantine configuration, the execution plan for a SQL statement still gets automatically quarantined; if the Resource Manager terminates it for exceeding any of the Resource Manager thresholds.
- You can transfer quarantine configurations from one database to another database using the DBMS_SQLQ package subprograms – CREATE_STGTAB_QUARANTINE, PACK_STGTAB_QUARANTINE, and UNPACK_STGTAB_QUARANTINE.

To view which plans are quarantined:

```
select sql_text, plan_hash_value, avoided_executions, sql_quarantine
from v$sql
where sql_quarantine is not null;
select sql_text, name, plan_hash_value, last_executed, enabled
from dba_sql_quarantine;
```

Resource Manager

When a SQL statement is terminated by the Resource Manager, as it exceeds a threshold, the execution plan for the SQL statement is now added to the quarantine list, so that it is not allowed to run again.

DBRM Setup – High Level

1. Create resource manager pending area
2. Create consumer group 'Runaway Group'
3. Map schema sessions to consumer group
4. Create resource plan limit on resource
5. Create resource plan directive to assign limit to consumer group with switch options
6. Validate and submit pending area

That's all you need to get started with SQL Quarantine.

Summary

In Oracle Database 18c, Oracle provides the capability to kill an offending SQL and leave the connected session with the database intact. In Oracle Database 19c, Oracle provides us the capability to quarantine the query that it can never come back to cause havoc.

One of the sayings we have at Viscosity is our customer's, "have four aces in their pocket". Over the next 7 days, the talented staff at Viscosity along with our Oracle ACEs will address more Oracle Database 18c and 19c new features. Continue to join us next year, as we continue our Oracle Database 19c hands-on-lab workshops.

Happy Holidays!