Using the 12c Real-time SQL Monitoring Report History for Performance Analysis

Mathias Zarick
Düsseldorf, May 30th 2017

Introduction – Mathias Zarick

- Principal Consultant at Trivadis Delphi GmbH in Vienna
- Graduated from University of Rostock / Computer Science
- Trainer
  - Data Guard, Architecture and Internals for advanced DBAs, Maximum Availability Architecture Workshop, Grid Infrastructure
- E-Mail: Mathias.Zarick@trivadis.com
- Main focus:
  - Oracle database
  - Oracle high availability projects (Real Application Clusters, Data Guard, Maximum Availability Architecture, Replication with Streams and GoldenGate)
  - Backup/Recovery
  - Trivadis Toolbox Architect
  - Developer of TVD-Standby
  - Research projects in Trivadis Technology Center (TTC)

ORACLE®
Certified Master
Oracle Database 11*g*
Administrator

ORACLE®
Certified Master
Oracle Database 12*c*
Administrator

trivadis
makes IT easier.

Agenda

1. Introduction
2. New Features as of 12c – Report Structure Insights
3. TaRTeSMon
4. Conclusion

Introduction

Introduction: Real-Time SQL Monitoring

- Available as of 11g (11.1.0.6)
- Allows to monitor execution of long running SQL queries
- Enabled for PQ, executions with 5 sec. CPU or IO time, or – if MONITOR hint is used
- Provides useful runtime and real-time statistics of past or running statements
  - CPU time, IO time
  - Cardinality of intermediate results
  - Memory, temporary space consumption of each operator in an execution plan
  - Bind variables of the execution (as of 11.2)
- As of 12c you can also build custom *composite database operations* with dbms_sql_monitor.begin_operation and dbms_sql_monitor.end_operation

Views and License

- Views for exposure: V$SQL_MONITOR and V$SQL_PLAN_MONITOR
- Requires Diagnostics and Tuning Pack license
- Parameter “statistics_level” needs to be at least TYPICAL (Default)
- Parameter “control_management_pack_access” needs to be ‘DIAGNOSTIC+TUNING’

PL/SQL API Examples

- No variables → last monitored execution of own session is used

```
VARIABLE report CLOB
EXEC :report := dbms_sql_monitor.report_sql_monitor;
PRINT report
```

- Other way providing sql_id, type (TEXT, HTML, XML, ACTIVE) and report_level

```
SELECT dbms_sql_monitor.report_sql_monitor(
  sql_id       => 'f1178wkrba2y9',
  type         => 'TEXT',
  report_level => 'ALL') AS report
FROM dual
```

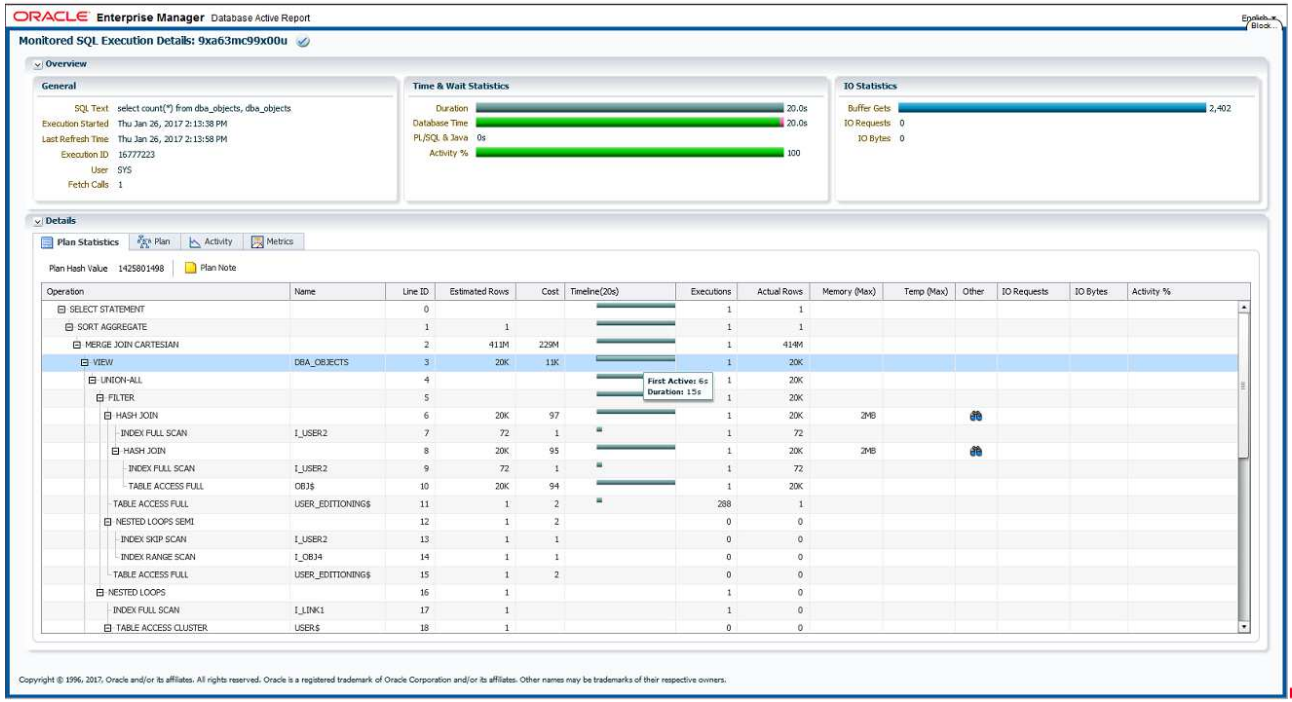
- As of 12c report_sql_monitor function is also available in package dbms_sql_monitor package, before we used the dbms_sqltune (which still works)

Active Report – generated with type → 'ACTIVE'

- This example generates an active report with SQL*Plus

```
SQL> SELECT ... ..

SQL> spool active_report_example.html
SQL> SELECT dbms_sql_monitor.report_sql_monitor(type=>'ACTIVE')
  2  FROM dual;
SQL> spool off
```

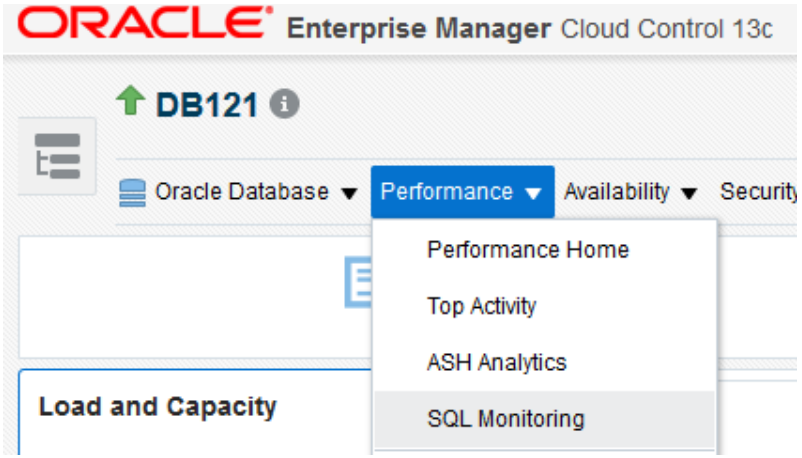


active_report_example.html

trivadis makes IT easier.

Also Accessible in OEM and SQL Developer

- Part of performance tab "SQL Monitoring"
  - Browse through SQL executions, get details for specific executions (active reports)
  - Save reports in HTML

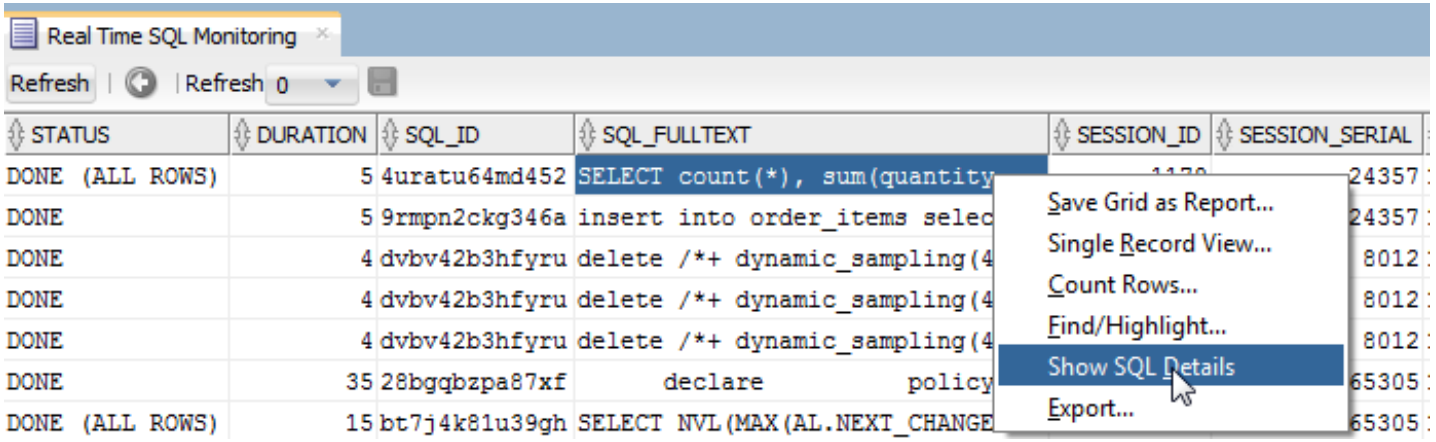


- SQL Developer is also able to do this and also offers active report saving

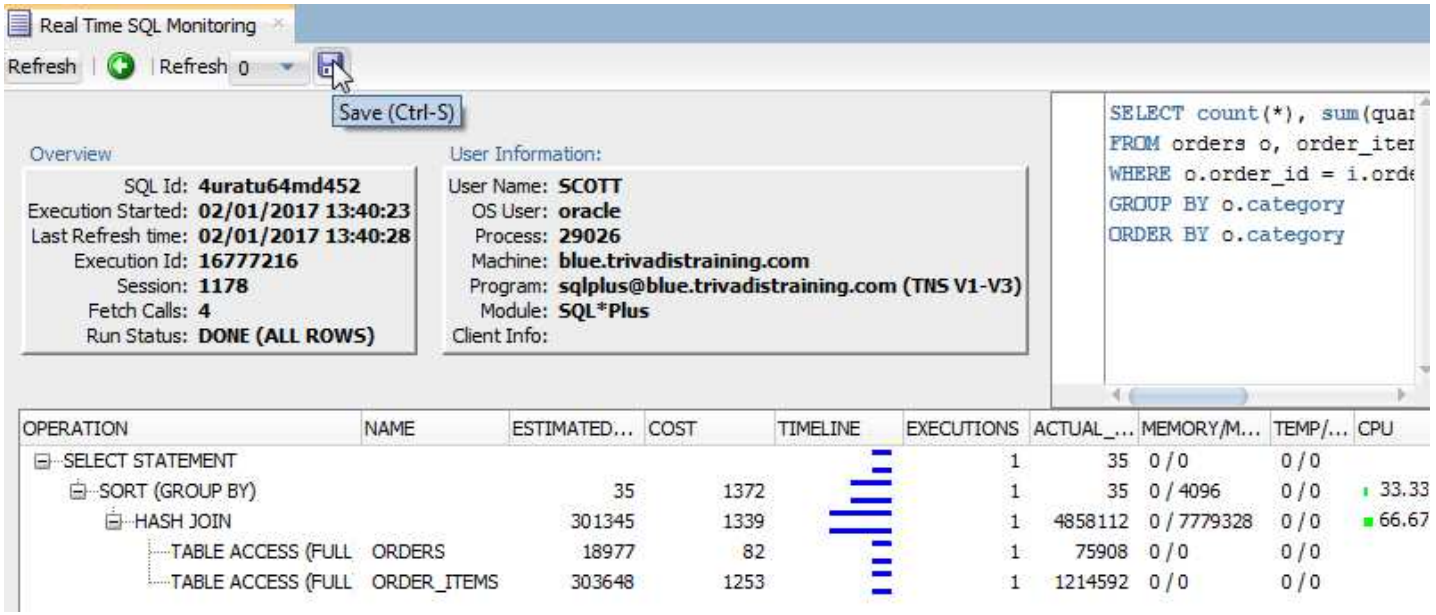
trivadis makes IT easier.

SQL Developer 4.1.5 and Real-Time SQL Monitoring

- Get the details of an execution



- View details and save the report

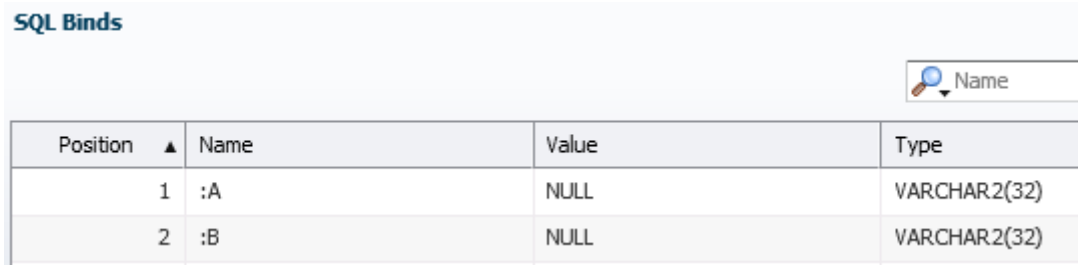


Active SQL Monitoring Reports Rendered by Flash

- OEM and SQL*Developer offer interfaces for generation, presentation and/or saving of active reports
- Active reports are rendered with the help of Adobe flash
  - OEM 13.2
    https://<myoem>/em/database/flex/orarep12102/sqlmonitor/SqlMonitorDetail.swf
  - Active report, saved by OEM 13.2
    http://download.oracle.com/otn_software/emviewers/db_12.1.0.2.0_20160125/em/orarep/sqlmonitor/SqlMonitor.swf
  - Active report, saved by SQL*Developer 4.1.5
    http://download.oracle.com/otn_software/emviewers/sqlmonitor/11/sqlmonitor.swf
  - Generated by dbms_sql_monitor.report_sql_monitor(type=>'ACTIVE')
    http://download.oracle.com/otn_software/emviewers/db_12.1.0.2.0_20160125/em/orarep/sqlmonitor/SqlMonitor.swf
  - The references to the flash resources are included in the generated HTML files, saved reports reference Oracle servers, thus need internet connection

But Beware of Bugs

- Active reports can lie regarding bind variables to be NULL
- This is happening for OEM and saved reports
- You might find in the flash



SQL Binds

Name

Position ▲	Name	Value	Type
1	:A	NULL	VARCHAR2(32)
2	:B	NULL	VARCHAR2(32)

- But the reality is

```
<binds>
  <bind name=":A" pos="1" dty="1" dtystr="VARCHAR2(32)" maxlen="32" csid="873" len="48440"/>
  <bind name=":B" pos="2" dty="1" dtystr="VARCHAR2(32)" maxlen="32" csid="873" len="1"> </bind>
</binds>
```

- Be suspicious about the NULL values !

Nothing Happens If you Try to Open an Active Report Saved by SQL Developer?

- Well – as I said, it references "…/sqlmonitor/11/sqlmonitor.swf" at an Oracle server
- It does so for 11g and 12c databases, but the flash plugin refuses to load XML for versions above 11g
- Solution, just remove the db_version attribute in the report tag

```
perl -pe 's|report db_version="12.*?"|report|' \
  active_report_from_sqldeveloper_9zpzdsjpyh8ub_orig.html \
> active_report_from_sqldeveloper_9zpzdsjpyh8ub_edited.html
```

Which Reports are Available?

- Package dbms_sqltune or dbms_sql_monitor
  - Function report_sql_monitor as seen before
  - Function report_sql_monitor_list as of 11.2 generates a report for the list of statements monitored by oracle (only those in memory in gv$sql_monitor are listed)
  - Function report_sql_detail as of 11.2 should provide even more details, type can only be ‘ACTIVE’ or ‘XML’ (only in dbms_sqltune)

    in these days spooled reports which use report_sql_detail are broken, cause of referenced javascript and flash objects at http://download.oracle.com/otn_software/ not functioning correctly (endless loop)

New Features as of 12c – Report Structure Insights

So What is New as of 12c?

- New views
  - v$sql_monitor_sesstat, v$sql_monitor_statname – I did not find use cases so far
  - dba_hist_reports, dba_hist_reports_details – stored reports of monitored SQL executions in XML format, we will drill in this right now
- New columns in v$sql_monitor
  - report_id: unique ID of the stored XML report

    report_id=0 means not stored yet, other ids can be used to reference a stored report
  - in_dbop_name, in_dbop_exec_id as of 12.2
  - io_cell_uncompressed_bytes, io_cell_offload_eligible_bytes, io_cell_offload_returned_bytes as of 12.2

Stored SQL Monitoring Reports (1)

- Available as of 12c
- The generation and saving is done by MMON /
  MODULE NAME:(MMON_SLAVE),
  ACTION NAME:(Automatic Report Flush)
- Only if control_management_pack_access it set to
  'DIAGNOSTIC+TUNING'
- It uses sys.dbms_auto_report_internal.i_save_report, which calls sys.dbms_report.get_report_with_summary – both are undocumented
- It saves reports in interval (one day) partitioned sys tables wrp$_reports and wrp$_reports_details in tablespace sysaux
- XML report column is a compressed securefile LOB, compression is done by the packages, not by securefile compression
- Only for SQL executions that are not currently executing or queued (status is DONE)
- Reports are automatically purged in conformance to AWR retention period

Stored SQL Monitoring Reports (2)

- Only for the top 5 SQL executions per minute
- You can use dbms_auto_report.start_report_capture and dbms_auto_report.finish_report_capture to store reports for all monitored statements
  Basically this changes the execution mode for the MMON action from *regular* to *full_capture* and vice versa, which can also be seen in dba_hist_reports_control
- Documented views dba_hist_reports, dba_hist_reports_details can be used to access XML reports, these views transparently decompress the data
- dba_feature_usage_statistics shows an active usage of “Real-Time SQL Monitoring”, if this saving of reports is done my MMON
  - I have seen this already in newly created databases just by the effects of catalog.sql, catproc.sql etc.
    → modify control_management_pack_access parameter already when creating databases, if you do not have tuning pack license to avoid unwanted discussions

SQL Monitoring XML Report Example

- Get report the way the MMON gets it from memory by using attributes from gv$sql_monitor: sql_id, sql_exec_id, etc.

```
SET DEFINE OFF LONG 100000
SELECT
  dbms_report.get_report_with_summary(
'/orarep/sqlmonitor/main?sql_id=6wxq34z70p2r6&sql_exec_id=16777216&' ||
'sql_exec_start=11:09:2016 15:49:54&last_refresh_time=11:09:2016 15:51:36&' ||
'inst_id=1&session_id=258&session_serial=45606&key=2&con_id=0'
 ) FROM DUAL
```

- For readability you can format the XML a bit

```
SELECT XMLSERIALIZE( DOCUMENT XMLTYPE(
dbms_report.get_report_with_summary('/ora...&con_id=0')) AS CLOB INDENT) report
FROM DUAL;
```

Get a Stored Report from dba_hist-Views

- Stored reports can be accessed via dba_hist_reports_details
- Decompression is done transparently via PL/SQL package functions

```
SELECT
  XMLSERIALIZE( DOCUMENT
  XMLTYPE (
      d.report
  ) as CLOB INDENT) report
FROM dba_hist_reports r, dba_hist_reports_details d
WHERE r.report_id = d.report_id
AND r.component_name='sqlmonitor'
AND r.report_id = &report_id
```

- Dba_hist_reports contains overview information – column “report_summary” XML tag “report_repository_summary”
- Dba_hist_reports_details contains actual reports
- Multitenant
  - if connected to CDB$ROOT → access to all reports of all containers
  - if connected to a PDB → access to local container only

XML Reports Look Like This

```
<report>
  <report_id><![CDATA[/orarep/sqlmonitor/main%3finst_id%3d1%26session_id%3d273%26session_serial%3d6537%26
  <sql_monitor_report version="4.0" sysdate="11/21/2016 15:10:13">
    <report_parameters>
      <sql_id>6xscskqnwsg53</sql_id>
      <sql_exec_id>16777216</sql_exec_id>
      <session_id>273</session_id>
      <session_serial>6537</session_serial>
      <sql_exec_start>11/21/2016 15:09:40</sql_exec_start>
      <bucket_count>35</bucket_count>
      <interval_start>11/21/2016 15:09:40</interval_start>
      <interval_end>11/21/2016 15:10:14</interval_end>
    </report_parameters>
    <target instance_id="1" session_id="273" session_serial="6537" sql_id="6xscskqnwsg53" sql_exec_start='
      <user_id>0</user_id>
      <program>oracle@zam36 (M000)</program>
      <module>MMON_SLAVE</module>
      <action>AWR Auto-Purge Slave Action</action>
      <service>SYS$BACKGROUND</service>
      <sql_fulltext is_full="Y">delete from WRH$_SYSMETRIC_HISTORY tab where (dbid = :dbid)    and snap_i
      <status>DONE</status>
      <refresh_count>11</refresh_count>
      <first_refresh_time>11/21/2016 15:09:57</first_refresh_time>
      <last_refresh_time>11/21/2016 15:10:13</last_refresh_time>
      <duration>33</duration>
      <optimizer_env>
        <param name="active_instance_count">1</param>
        <param name="is_recur_flags">39</param>
        <param name="optimizer_mode">choose</param>
        <param name="parallel_autodop">0</param>
        <param name="parallel_ddl_mode">enabled</param>
        <param name="parallel_ddldml">0</param>
        <param name="parallel_degree">0</param>
```

For XML schema have a look at
$ORACLE_HOME/rdbms/xml/em/orarep/sqlmonitor/sqlmonitorSch.xsd

dbms_sql_monitor.report_sql_monitor produces XML where inner node contents are encoded (looks like base64)

report_from_db.xml

XML Reports Contain Even More

- XML reports can be visualized with the help of the flash plugins → active report
- But the XML itself offers even more, which is not shown in the browser:
  - sql_exec_id, session_id, session_serial, program, module, action, service
  - Optimizer environment
  - Outline of the execution plan
  - Peeked binds
  - …
- You can access the XML by using SQL/XPath within the database or by simply reading through spooled XML files (only for non-encoded XML)

Structure of Active Report HTML File (1)

- It starts like this

```
<html>
 <head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
  <script language="javascript" type="text/javascript">
   <!--
      var version = "12.1.0.2.0";
     var swf_base_path = "http://download.oracle.com/otn_software/";

         document.write('<script language="javascript" type="text/javascript" ' +
                        'src="' + swf_base_path + 'emviewers/scripts/activeReportInit.js?' +
                        Math.floor((new Date()).getTime()/(7*24*60*60*1000)) +
                       '"></' + 'script>');
     -->
  </script>
 </head>
 <body onload="sendXML();">
  <script type="text/javascript">
       writeIframe();
    </script>
  <script id="fxtmodel" type="text/xml">
   <!--FXTMODEL-->
   <report db_version="12.1.0.2.0" elapsed_time="0.29" cpu_time="0.27" cpu_cores="2" hyperthread="N"
    <report_id><![CDATA[/orarep/sqlmonitor/main?inst_id=1&session_id=38&session_serial=11284&sql_exec
                eAHtXX1zGjnS/38/xdw8W3t2lWMDw2sWU4thknDhxQc4u7mrq6kxjB02wBAYkng/
/fNrvcxoYN7Azm5yl63KAlJL6m51t7qlllzffJhbC3c589y1tXZW7trTPjrrzcxd
XurF85yubR42U9tzLvV84SJvXBRy+bJWMJ7ncs+LZb3xq6bVeTNrZa/theOhMZWi
```

Structure of Active Report HTML File (2)

- It ends like this

```
dzSjeaXOYixQiERFZjJoYj7drYmZIXUWjVAibaRbg79LFNORQmI8UIhEhaFZSEz3
a2IwU0gsQdwjhVmdxXyGNbEcCxQiUVl6spCY7tjEzJBCYjkXwwaVxEoWQY0FegyJ
WZ5NOom1dM/GyFex6RI51YGgJgA9hsRU1ybdohZypdSF3zBqcWZXITEe6DEkpvo2
cX5LIKiFiqHYONUwB4JarBRVl0QFCkhMAMJ+gQjrcCkqRi0UNhRrCtPD+ip/8T/m
xLLMaYep8QMie9zxDf+x6cYP/w+qSWcZ
            </report>
   <!--FXTMODEL-->
  </script>
 </body>
</html>
```

Ending

Getting List of Available Reports from Memory

- Just query gv$sql_monitor

```
SELECT
  report_id,
  con_id,
  inst_id,
  status,
  username,
  module,
  action,
  service_name,
  program,
  sid,
  session_serial#,
  elapsed_time,
  -- gives up to 2000 characters of SQL statement
  sql_text,
  sql_id,
  sql_exec_start,
  sql_exec_id,
  sql_plan_hash_value,
  first_refresh_time,
  last_refresh_time,
  refresh_count
FROM gv$sql_monitor
WHERE sql_text IS NOT NULL
```

Getting The Same for Stored Reports

- Query dba_hist_reports, but a little bit more complicated by using report summaries XML and XPath expressions

```
SELECT
  report_id,
  con_id,
  instance_number inst_id,
  extractValue(XMLTYPE(report_summary), '/report_repository_summary/sql/status') status,
  extractValue(XMLTYPE(report_summary), '/report_repository_summary/sql/user') username,
  extractValue(XMLTYPE(report_summary), '/report_repository_summary/sql/module') module,
  extractValue(XMLTYPE(report_summary), '/report_repository_summary/sql/action') action,

  ... left out a lot for readability

  -- gives only first 100 characters of SQL statement
  extractValue(XMLTYPE(report_summary), '/report_repository_summary/sql/sql_text') sql_text,
  extractValue(XMLTYPE(report_summary), '/report_repository_summary/sql/@sql_id') sql_id,
  ...

  to_number(extractValue(XMLTYPE(report_summary),
            '/report_repository_summary/sql/@sql_exec_id')) sql_exec_id,
  ...
  to_number(extractValue(XMLTYPE(report_summary),
            '/report_repository_summary/sql/refresh_count')) refresh_count
FROM dba_hist_reports
WHERE component_name='sqlmonitor'
ORDER BY report_id
```

TaRTesMon

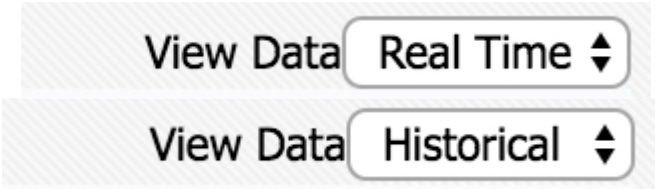
Getting Readable Output for a Stored Report (1)

- Report is still in memory (gv$sql_monitor) → use dbms_sql_monitor.report_sql_monitor
- Report is only available on disk → dbms_sql_monitor.report_sql_monitor does not work
- Empty report is created if you try it the naïve way

```
SQL> SELECT
  2    dbms_sql_monitor.report_sql_monitor (
  3      sql_id => '9xa63mc99x00u',
  4      sql_exec_id  => 16777216,
  5      type => 'XML'
  6    )
  7  FROM dual;
<report db_version="12.1.0.2.0" elapsed_time="0.02" cpu_time="0.02" cpu_cores="2"
hyperthread="N" timezone_offset="3600" packs="2">
<report_id><![CDATA[/orarep/sqlmonitor/main?sql_exec_id=16777216&sql_id=9xa63mc99x00u]]>
</report_id>
</report>
```

Getting Readable Output for a Stored Report (2)

- OEM is one valid solution



  OEM is able to render the XML internally on its own
  of course this is only offered for 12c databases

- No OEM – no readable report ? No!
- But a little more complicated
- The representations of ‘TEXT’ and ‘HTML’ cannot be traced back to the raw XML, but for the ‘ACTIVE’ report we can, a saved HTML contains
  - HTML header with reference to javascript and flash objects on download.oracle.com
  - Raw XML report
  - Ending of the HTML document
- So this we can run also in the opposite direction

Getting Readable Output for a Stored Report (3)

- Approach:
  - Create an HTML-file with
    - the static prolog
    - the raw XML of our report from dba_hist_reports_details
    - and the static epilog
  - Borrow the static stuff from the best sources, so from reports generated with dbms_sql_monitor.report_sql_monitor

```
SELECT &prolog FROM dual;

SELECT XMLSERIALIZE( DOCUMENT XMLTYPE ( report ) as CLOB INDENT) report
FROM dba_hist_reports_details
WHERE report_id = 4711;

SELECT &epilog FROM dual;
```

Introducing TaRTeSMon

- Well it needed a name: Trivadis Real-Time SQL Monitor, please consider it as – maybe – temporary
- SQL*Plus script
  - Spool an HTML overview report with accessible real-time sql monitor reports
  - Spool the individual reports
  - Optionally apply a filter (recommended)
- What you need
  - A host with SQL*Plus connection to db in question, ideally with internet connection (for the active reports and/or for TaRTeSMon source)
  - Connected user needs CREATE SESSION and SELECT privileges on the mentioned views (gv$sql_monitor, dba_hist_reports, dba_hist_reports_details)
- You can use it to get additional information (active real-time SQL monitoring reports) for your performance tuning session

Starting TaRTeSMon

Example session

```
SQL> @http://zarick.de/tartesmon
you can create a filter here based on following attributes:
report_id           NUMBER
con_id              NUMBER
inst_id             NUMBER
...
still_in_memory     VARCHAR2

Press enter if you do not want to apply a filter
Enter a filter here: elapsed_time > 100000000
generation finished, output directory is tartesmon_PHYS_20170202_172407
```

Other filter examples

```
# username != 'SYS' AND sql_exec_start >= sysdate – 1
# refresh_count >= 75 OR upper(sql_text) LIKE '%SELECT%ORDER_ITEMS%';
# sql_exec_start > to_date('01.01.2017','DD.MM.YYYY') AND
  sql_exec_start < to_date('07.01.2017','DD.MM.YYYY')
# still_in_memory = 'N'
```

Filters and TaRTeSMon

- To keep the runtime short you should use filters
- Runtime example – no filters
  - Database 12.1.0.2, production knowledge management system of a logistics company in Austria
  - AWR retention 30 days
  - 7570 active reports
  - Generation runtime 10 minutes
  - 18 MB compressed archive (tar.gz)

Use Cases for TaRTeSMon

- 12c, Release 1 and 2
- EE + Diagnostics and Tuning pack needs to be available
- If OEM is unavailable, TaRTeSMon might fill this gap to browse real-time monitoring reports
- Also useful is to take reports for offline analysis

- Get more insights in performance tuning sessions
- Find bind variables for a particular SQL run
- Drill into execution plans, e.g. compare estimated and actual rows
- See resource usages of individual runs (PGA, CPU, IOs)

Example UseCase for TaRTeSMon

- A known business report typically runs for 10 minutes.
  But there are exceptions, sometimes it takes about 1 hour.

- Approach with TaRTeSMon:
  Get all reports for this sql_id and analyze

- Find the different elapsed times which we got

```
grep "stat name=\"elapsed" sqlmonitor_3gvhd1pfbdkfd_*.html
```

- Find the different bind variables which were used

```
grep "bind name" sqlmonitor_3gvhd1pfbdkfd_*.html
```

- Or just use the browser ☺

Conclusion

Real-Time SQL Monitoring Reports

- **Persistent real-time SQL monitor information**
  as of 12c but only with diagnostics and tuning pack
- **Flash reports are quite useful**
  but beware of bugs, do not trust in the NULLs as binds

- **Pure XML contains even more**
  e.g. optimizer environment
- **TaRTeSMon**
  Well – it is just some queries to spool the report information to be used for offline analysis, for rendering and more

Further Information …

- http://blog.trivadis.com/b/mathiaszarick/archive/2017/03/17/real-time-sql-monitoring-report-history-in-12c.aspx
- https://oracle-base.com/articles/11g/real-time-sql-monitoring-11gr1
- https://bdrouvot.wordpress.com/2013/04/29/bind-variable-peeking-retrieve-peeked-and-passed-values-per-execution-in-oracle-11-2/
- http://www.oracle.com/technetwork/database/manageability/sqlmonitor-084401.html
- http://www.oracle.com/technetwork/database/manageability/owp-sql-monitoring-128746.pdf
- http://docs.oracle.com/database/121/TGSQL/toc.htm
- http://docs.oracle.com/database/121/ARPLS/d_sqltun.htm

trivadis
makes IT easier.

Questions And Answers

Mathias Zarick
Principal Consultant

Mathias.Zarick@trivadis.com

trivadis
makes IT easier.