Improving Statspack Experience

When you are in Standard Edition, or when you are in Enterprise Edition without the Diagnostic Pack, you cannot use AWR and the performance pages of Enterprise Manager (dbconsole, EM express or Cloud Control) are empty. As we often have to troubleshoot performance problems that occurred in the past, it is recommended to install Statspack, which requires no additional option licensing.

Franck Pachot, dbi services

Installing Statspack is not difficult at all and is documented in the spdoc.txt provided in Oracle Home rdbms/admin directory. But, through the years of using it on different versions and environments, there are few, or more, things I do to enhance its usage. In this article, I would like to share: some best practices, code snippets and ideas to improve your experience with Statspack.

I recommend to have Statspack collecting snapshots at least every hour in any database which is not covered by Diagnostic Pack. When a user comes and tells me that the database was slow this morning, but is back to normal now, I can see nothing without a history of snapshots. But I will probably see lot of system and application statistics if I have snapshots from that time.

Installation

Statspack is not installed by default but is easy to install by running `spcreate.sql` which you find in `ORACLE_HOME/rdbms/admin`.

Here is the script (**Listing 1**) that I use to create a STATSPACK tablespace and install Statspack, to be run when connected as sysdba. As I recommend to put Statspack tables in their specific tablespaces instead of SYSAUX, you just have to customise the file name. I usually start with a 2GB tablespace but you can increase it if you have lot of big SQL statements and want a long retention.

```
set echo on
whenever sqlerror exit failure
create tablespace STATSPACK datafile '+DATA' size 100M autoextend on maxsize 2G;
define default_tablespace='STATSPACK'
define temporary_tablespace='TEMP'
column random new_value perfstat_password noprint
select '"'||dbms_random.string('a',30)||'"' random from dual;
alter session set "_oracle_script"=true;
@?/rdbms/admin/spcreate
alter session set "_oracle_script"=false;
```

LISTING 1- STATSPACK INSTALLATION

You connect as sysdba to run it and if you want to install it in a pluggable database, you first have to ALTER SESSION SET CONTAINER, or connect through the listener to the PDB service. If you want to install Statspack in CDB$ROOT, and I'll explain why later, in 12cR1 you need to set **"_oracle_script"** to true, so that the Statspack schema, the PERFSTAT user, can be created without **C##** prefix. It's an Oracle maintained script anyway, so no problem in doing that. In 12.2.0.1 this setting is in **spcreate.sql** but still missing from **spdrop.sql** (Bug 25233027 opened for this).

You can see that I set a random password. Even if it's only performance data, they include SQL statements, bind values, and information about your application, it's better to protect it. Change the password later or just don't connect with PERFSTAT. You will probably use it from your own DBA user (all tables are accessible via public synonyms) and can then even lock PERFSTAT account.

The creation should finish with 'successful'. In case of error, you can deinstall with **spdrop.sql**.

Filter and Access Predicates

Actually before installing, there's something I sometimes 'hack' in **spcpkg.sql** because the gathering of execution plans bypasses the most interesting information: the where clause predicates. This comes from old bugs in 9i where this gathering used to encounter an ORA-7445 because those columns from **v$sql_plan** are a bit special. The predicate information is not stored as-is and has to do some 'reverse parsing'. However, there is no issue in the current versions, I query those columns very often and don't tend to see problems anymore. Then, you can replace the commented lines:

```
, 0 -- should be max(sp.access_predicates) (2254299)
, 0 -- should be max(sp.filter_predicates)
```

and change them to the ones commented out as 'should be':

```
, max(sp.access_predicates) -- not supported --
, max(sp.filter_predicates) -- not supported --
```

If you had already run the **spcreate.sql** above, just reload the package by running **spcpkg.sql** (set the current schema to **PERFSTAT**) but, it's better to do that before taking any snapshot or you will have some plans that miss the predicates. Be aware that this is not supported, and I'm not responsible for any side effects.

Snapshot Level

By default, Statspack snapshots are taken at level 5, but I want to gather execution plans and gather segment statistics and both are collected only from level 7.

Before setting the default to level 7 and running automatic jobs, I want to be sure that the level 7 gathering only takes a few seconds:

```
SQL> set timing on
SQL> exec statspack.snap(i_snap_level=>7);
PL/SQL procedure successfully completed.
Elapsed: 00:00:03.04
```

I've seen few cases where it takes longer and it's better to fix the issue before setting the default level to 7. One cause can be lot of data in PERFSTAT with no statistics gathering. Another reason is when the SGA is sized too large, then the shared pool may become huge, and long to read.

So, a few seconds is ok, then you can set the default level to 7 (**Listing 2**).

```
SQL> exec STATSPACK.MODIFY_STATSPACK_PARAMETER (i_snap_
level=>7, i_instance_number=>null);
PL/SQL procedure successfully completed.
```

LISTING 2 - SET STATSPACK DEFAULT LEVEL TO 7

If you are in RAC, you have to do that for each instance.

Scheduled Jobs

Now it's time to schedule the jobs that will be owned by the PERFSTAT user so that they will be dropped if we de-install Statspack. If you used the random password for the create, you can connect:

```
connect perfstat/&perfstat_password
```

You will be sad to see that I still use the deprecated dbms_job and you are free to use dbms_scheduler, but I have this script (**Listing 3**) that I've used for years which has the advantage to create the job with the instance number. If you are in RAC you have to run it on each instance because statistics gathering and purge is done per instance.

There is one job to take a snapshot every hour and one job to call the purge job every week, keeping 45 days of history.

Check the jobs:

```
SQL> select last_sec,next_date,next_sec,log_user,instance,what from dba_jobs;

LAST_SEC NEXT_DATE NEXT_SEC LOG_USER WHAT
-------- --------- -------- ----------------------------------------------------------------
         29-JAN-17 10:00:00 PERFSTAT statspack.snap;
         06-JAN-17 00:00:00 PERFSTAT statspack.purge(i_num_days=>45,i_extended_purge=>true);
```

```
connect perfstat/&perfstat_password
variable jobno number
variable instno number
begin
    select instance_number into :instno from v$instance;
    dbms_job.submit(:jobno, 'statspack.snap;', trunc(sysdate+1/24,'HH'),
'trunc(SYSDATE+1/24,''HH'')', TRUE, :instno);
    dbms_job.submit(:jobno, 'statspack.purge(i_num_days=>45,i_extended_purge=>true);',
trunc(SYSDATE)+7, 'trunc(SYSDATE)+7', TRUE, :instno);
      commit;
   end;
/
```

LISTING 3 - SCRIPT TO CREATE SNAPSHOT AND PURGE JOBS

```
delete from STATS$IDLE_EVENT;
insert into STATS$IDLE_EVENT
select name from V$EVENT_NAME
where wait_class='Idle';
commit;
```

LISTING 4- REPLACE IDLE EVENT LIST BY THE IDLE WAIT EVENT ONES

```
merge into STATS$SNAPSHOT s
using (
select
 dbid,instance_number,snap_id
 ,ltrim(substr(listagg(name||':'||ltrim(to_char(time_waited_micro/elapsed_micro,'09.9'))||' ')
  within group(order by round(time_waited_micro/elapsed_micro,1) desc,name nulls first),1,21)
  ,':') ucomment
from (
select
  dbid,instance_number,snap_id
  ,decode(name,'DB time','','DB CPU','CPU','User I/O','I/O','System I/O','I/O',substr(name,1,3)) name
  ,time_waited_micro-lag(time_waited_micro)over(partition by dbid,instance_number,name order by snap_id) time_waited_micro
  ,case when lag(snap_time)over(partition by dbid,instance_number,name order by snap_id)-startup_time > 0 then
   (snap_time-lag(snap_time)over(partition by dbid,instance_number,name order by snap_id))*24*60*60*1e6
   end elapsed_micro
 from (
   select dbid,instance_number,snap_id,wait_class name,sum(time_waited_micro) time_waited_micro
   from stats$system_event join v$event_name using(event_id)
   where wait_class not in ('Idle')
   group by dbid,instance_number,snap_id,wait_class
  union all
   select dbid,instance_number,snap_id,stat_name name,value
   from stats$sys_time_model join stats$time_model_statname using (stat_id) where stat_name in ('DB time','DB CPU')
 ) join stats$snapshot using(dbid,instance_number,snap_id) where ucomment is null
) where  elapsed_micro >1e6 and time_waited_micro>1e6
group by dbid,instance_number,snap_id
) l on (s.snap_id=l.snap_id and s.dbid=l.dbid and s.instance_number=l.instance_number)
when matched then update set s.ucomment=l.ucomment;
```

LISTING 5- SET SNAPSHOT COMMENTS WITH AVERAGE DATABASE LOAD

This is an example but you can schedule **`exec statspack.snap`** and **`exec statspack.purge(i_num_days=>45,i_extended_purge=>true)`** with whatever you like for scheduling jobs run as sysdba. You can also do it from crontab or Task Manager. The **`spauto.sql`** script that is provided also uses dbms_job but do not schedule any purge. It is always a bad idea to gather information automatically without thinking about retention. Especially as the Statspack tables are stored in SYSAUX and it will grow.

SNAP_ID Sequence

The snapshots are identified with a SNAP_ID and some people like to see this number without a gap when they query the Statspack views. I don't need that because I use the LAG() analytic function, but if you prefer no gap, no problem, the sequence is not read frequently: **`SQL> ALTER SEQUENCE perfstat.stats$snapshot_id NOCACHE;`**

When you join Statspack tables, don't forget to include DBID and INSTANCE_NUMBER in addition to SNAP_ID, just in case you use your scripts later on a database with new DBID (after a RMAN DUPLICATE for example) or in RAC.

Idle Events

The list of idle events in STATS$IDLE_EVENT often lags behind the new events introduced in new releases. This leads to very misleading wait events as shown at http://blog.dbi-services.com/statspack-idle-events/ and my preferred solution, even if it's not the supported one, is to replace this list of idle events by the idle event wait class (**Listing 4**).

Date Format

Your locale date format may be larger than what was defined in spreport, such as when in the French language:

```
                                                          Snap
Instance     DB Name        Snap Id   Snap Started      Level Comment
------------ ------------ --------- ------------------ ----- --------
CDB          CDB                531 04 Juil. 2016 00:     7
                                    00
                                541 04 Juil. 2016 01:     7
                                    00
```

In that case it's better to set NLS_LANG=AMERICAN_AMERICA before running spreport.sql.

Snapshot Comment

Statspack has a feature that does not exist in AWR: a comment can be associated with a snapshot. For example, when you take a manual snapshot, you can tag it with a comment:

```
SQL> exec statspack.snap(i_ucomment=>'before load test #15');
```

But the automatic snapshots do not have any comment. I use the script in **Listing 5** to set all null comments to one that shows

```
                                                  Snap
Instance     DB Name        Snap Id   Snap Started    Level Comment
------------ ------------ --------- ----------------- ----- --------------------
cdb1         CDB                330 12 Jul 2016 17:00     7 20.0 I/O:15.1 CPU:01
                                332 12 Jul 2016 18:00     7 21.4 I/O:14.4 Sch:02
                                334 12 Jul 2016 19:00     7 20.1 I/O:14.1 CPU:01
                                337 12 Jul 2016 20:00     7 19.0 I/O:14.1 CPU:01
                                338 12 Jul 2016 21:00     7 18.4 I/O:13.2 CPU:01
                                340 12 Jul 2016 22:00     7 21.1 I/O:15.0 CPU:01
                                342 12 Jul 2016 23:00     7 22.6 I/O:13.3 Sch:03
                                345 13 Jul 2016 00:00     7 18.6 I/O:13.2 CPU:01
                                347 13 Jul 2016 01:00     7 19.0 I/O:14.2 CPU:01
                                349 13 Jul 2016 02:00     7 18.9 I/O:14.6 CPU:01
                                350 13 Jul 2016 03:00     7 17.8 I/O:13.9 CPU:01
                                352 13 Jul 2016 04:00     7 18.4 I/O:14.1 CPU:01
```

LISTING 6 - OUTPUT OF SPREPORT.SQL WITH AAS COMMENTS

```
create or replace view PERFSTAT.DELTA$SNAPSHOT as select
  e.SNAP_ID
 ,DBID
 ,INSTANCE_NUMBER
 ,SNAP_TIME
 ,lag(e.SNAP_ID)over(partition by DBID,INSTANCE_NUMBER,STARTUP_TIME order by e.snap_id) BEGIN_SNAP_ID
 ,ROUND((SNAP_TIME-lag(SNAP_TIME)over(partition by DBID,INSTANCE_NUMBER,STARTUP_TIME order by e.snap_id))*24*60*60) SNAP_
SECONDS
 ,STARTUP_TIME,PARALLEL, VERSION, DB_NAME, INSTANCE_NAME, HOST_NAME
FROM  PERFSTAT.STATS$SNAPSHOT e
 join stats$database_instance i using(STARTUP_TIME,DBID,INSTANCE_NUMBER)
/

create or replace view PERFSTAT.DELTA$SYSSTAT as select
 n.snap_time,n.snap_seconds
 ,e.SNAP_ID
 ,e.DBID
 ,e.INSTANCE_NUMBER
 ,e.STATISTIC#
 ,e.NAME
 ,e.VALUE-b.VALUE VALUE
 ,n.startup_time instance_startup_time,n.db_name,n.instance_name,n.host_name
from PERFSTAT.DELTA$SNAPSHOT n join PERFSTAT.STATS$SYSSTAT e
 on(e.snap_id=n.snap_id and e.dbid=n.dbid and
 e.instance_number=n.instance_number)
 join PERFSTAT.STATS$SYSSTAT b
 on(n.begin_snap_id=b.snap_id AND e.dbid=b.dbid AND
 e.instance_number=b.instance_number and e.NAME=b.NAME)
/
```

LISTING 7 - CUSTOM 'DELTA$' VIEWS FOR STATS$SNAPSHOT AND STATS$SYSSTAT INFORMATION

the Average Active Sessions (AAS), and main timed event.

With the comments set with **Listing 5** the `spreport.sql` list of snapshots id looks like the output in **Listing 6**. In this example, you can see that the Average Active Session is between 17 and 21 and the highest wait classes: 'I/O' for 'User I/O', 'CPU' for 'DB CPU', 'Sch' for 'Scheduler', etc.

Running these scripts helps to identify the period of time where you have activity.

Delta Values

The Statspack tables store only the cumulative values from the start of the instance. They make sense only when calculating the delta values between two snapshots, which is what `spreport.sql` does.

When you want to go beyond the report, you can query the tables, but then you need to self-join the tables to get the previous snapshot and calculate the difference. An example of this for STATS$SNAPSHOT and STATS$SYSSTAT is in **Listing 7**. DELTA$SNAPSHOT adds the elapsed time (SNAP_SECONDS) and previous SNAP_ID (BEGIN_SNAP_ID) using the analytic function LAG(). DELTA$SNAPSHOT joins to DELTA$SNAPSHOT and joins to itself to get previous snapshot values to subtract.

I didn't want to write those kind of views for each Statspack table and maintain them with Statspack evolution, so 10 years ago I created a script (see http://www.dba-village.com/village/dvp_scripts.ScriptDetails?ScriptIdA=3128) to generate them (see **Listing 8**). It calculates the delta value for all nullable number datatype columns, and joins on primary key. The script generates the view creation in `delta.tmp` and runs it.

Pack Management

When you are in Enterprise Edition and don't have Diagnostic Pack, you will use Statspack, but by default, AWR is still activated:

```
SQL> show parameter pack

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
control_management_pack_access       string      DIAGNOSTIC+TUNING
```

```
define owner=PERFSTAT
define prefix=DELTA

whenever sqlerror exit failure
whenever  oserror exit failure

CREATE OR REPLACE VIEW &owner..&prefix.$SNAPSHOT AS
SELECT
  e.SNAP_ID
 ,DBID
 ,INSTANCE_NUMBER
 ,SNAP_TIME
 ,lag(e.SNAP_ID)over(partition by DBID,INSTANCE_NUMBER,STARTUP_TIME order by e.snap_id) BEGIN_SNAP_ID
 ,ROUND((SNAP_TIME-lag(SNAP_TIME)over(partition by DBID,INSTANCE_NUMBER,STARTUP_TIME order by e.snap_id))*24*60*60) SNAP_
SECONDS
 ,STARTUP_TIME,PARALLEL, VERSION, DB_NAME, INSTANCE_NAME, HOST_NAME
FROM  &owner..STATS$SNAPSHOT e
 join stats$database_instance i using(STARTUP_TIME,DBID,INSTANCE_NUMBER)
/
```

```
BEGIN
 FOR v IN (
  SELECT table_name , '&prefix.'||SUBSTR(table_name,INSTR(table_name,'$')) view_name FROM ALL_TAB_COLUMNS
 WHERE table_name LIKE 'STATS$%' AND table_name NOT IN ('STATS$SNAPSHOT','STATS$DATABASE_INSTANCE')
 AND nullable='N' AND column_name IN ('SNAP_ID','DBID','INSTANCE_NUMBER') GROUP BY table_name HAVING COUNT(*) = 3
 ) LOOP
  dbms_output.put_line('');
  dbms_output.put_line('create or replace view &owner..'||v.view_name||' as select n.snap_time,n.snap_seconds');
  FOR c IN (
   SELECT * FROM ALL_TAB_COLUMNS
  WHERE NOT(nullable='Y' AND data_type  IN ('NUMBER')) AND table_name=v.table_name AND owner='&owner.'
  ) LOOP
  dbms_output.put_line(' ,e.'||c.column_name);
  END LOOP;
  FOR c IN (
   SELECT * FROM ALL_TAB_COLUMNS
  WHERE nullable='Y' AND data_type  IN ('NUMBER') AND table_name=v.table_name AND owner='&owner.'
  ) LOOP
  dbms_output.put_line(' ,e.'||c.column_name||'-b.'||c.column_name||' '||c.column_name);
  END LOOP;
  dbms_output.put_line(' ,n.startup_time instance_startup_time,n.db_name,n.instance_name,n.host_name');
  dbms_output.put_line('from &owner..&prefix.$SNAPSHOT n join &owner..'||v.table_name||' e ');
  dbms_output.put_line(' on(e.snap_id=n.snap_id and e.dbid=n.dbid and e.instance_number=n.instance_number)');
  dbms_output.put_line(' join &owner..'||v.table_name||' b'  );
  dbms_output.put(' on(n.begin_snap_id=b.snap_id AND e.dbid=b.dbid AND e.instance_number=b.instance_number');
  FOR c IN (
   SELECT * FROM ALL_CONS_COLUMNS join ALL_CONSTRAINTS USING(owner,constraint_name)
  WHERE constraint_type='P' AND column_name NOT IN ('SNAP_ID','DBID','INSTANCE_NUMBER')
  AND owner='&owner.' AND ALL_CONSTRAINTS.table_name=v.table_name
  ) LOOP
  dbms_output.put(' and e.'||c.column_name||'=b.'||c.column_name);
  END LOOP;
  dbms_output.put_line(')');
  dbms_output.put_line('/');
 END LOOP;
 dbms_output.put_line('');
END;
.

set feedback off serveroutput on size 100000
spool delta.tmp
/
spool off
set feedback on echo on
start delta.tmp
```

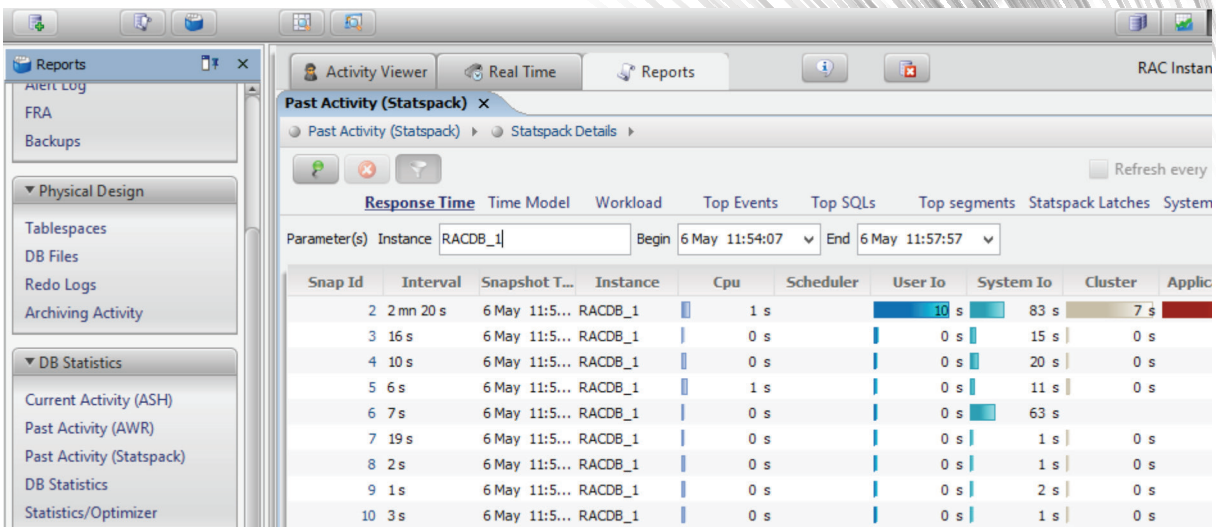
LISTING 8 - CREATE DELTA$ VIEWS WITH SNAPSHOT INFORMATION AND DELTA VALUES FOR ALL METRICS

It is an overhead and in addition to that it is a risk that someone uses it and gets it flagged in DBA_FEATURE_USAGE_STATISTICS, which will be a problem in the case of a license audit. My recommendation is that you set `control_management_pack_access` to `none`. Oracle recommends to leave it set because some functions accessible without the licensed option are based on AWR framework. Some examples are segment advisors and undo advisors. In my opinion the risk to pay millions because of a recorded usage is more important. It's different in 12.2 multitenant because lockdown profiles help finer control on features used.

In Standard Edition, the `control_management_pack_access` is set to none by default and it is not recommended to change that.

Graphical View

What is missing with Statspack are the graphics that you can see on EM Express or Cloud Control performance pages. They are unfortunately only based on AWR so you have nothing when you don't have the Diagnostic Pack license. An alternative is to build your own queries, using the 'DELTA' views I've built above, and display the result on SQL Developer reports, or on Excel.




```
column "SNAP_ID" new_value end_snap noprint
column "LAG(SNAP_ID)OVER(ORDERBYSNAP_TIME)" new_value begin_snap noprint
select snap_id,lag(snap_id)over(order by snap_time) from stats$snapshot order by snap_time;
define report_name=sp_last.txt
@?/rdbms/admin/spreport
exit
```

LISTING 9 - SPLASTREP.SQL TO GENERATE THE LATEST REPORT

There are also third party tools that can do that and I recommend Orachrome Lighty which can graphically display what you get from Statspack reports:

And Lighty can go further with a job that simulates ASH and then can really simulate AWR. I've described how to test it in a blog post: http://blog.dbi-services.com/exploring-oracle-se-a-ee-performance-statistics-with-orachrome-lighty/.

Other Possible Enhancements

The nice thing about Statspack is that we have the source, which is very useful to understand where the statistics come from and how ratios are calculated. We can also do some changes, for example, for 12*c* you can gather snapshots for V$EVENT_HISTOGRAM_MICRO and add a section in the report. The enhancement request to add it for AWR was accepted for 12.2 but not as yet in Statspack.

I often use the following `splastrep.sql` script to automatically generate a report on the two latest snapshots. When the `begin_snap,end_snap` and `report_name` variables are defined, then `spreport.sql` does not require any user input. Run **Listing 9** with sqplus (`spreport.sql` formatting is not compatible with sqlcl).

You can also try to `set markup html` on, on some sections if you prefer html reports over text ones but you will see that some sections are better displayed with preformatted text.

Multitenant

In multitenant you install Statspack in each container. Oracle does not support installing it at CDB$ROOT, but my opinion is different because you may want to capture activity of sessions that switched to CDB$ROOT through metadata or data links. Having a report that covers the whole instance may be a good start for system-wise performance analysis. In each PDB, the statistics are related to the container (what you see locally from V$ views), but be careful as some statistics have a meaning only at CDB level (redo for example).

Conclusion

Many features that are available with options, like AWR, have their counterparts, which you need to spend time customising. The good thing is that Statspack is very similar to AWR. AWR was an enhancement of Statspack (which itself was an enhancement of UTLBSTAT/UTLESTAT which are still there in ORACLE_HOME). They are based on the same metrics: statistics and wait events, so the interpretation can be done with the same knowledge and the help of Oracle Database reference documentation.

Statspack has several interesting features that are not well known. You can baseline some snapshots to keep them beyond the purge retention (`statspack.make_baseline`). You can export/import Statspack repository with Data Pump to a centralised one. When you upgrade the database you have to upgrade Statspack, or better: export the old repository elsewhere and re-create it in new version.

If you are running Active Data Guard you can use Statspack to analyse performance on the standby. You install it in the primary (with `sbcreate.sql and sbaddins.sql`) and snapshot gathering uses database links to query the standby performance

views. This has been available since 11*g*R1 and Statspack is the only way, even with Diagnostic Pack, because the support of AWR for Active Data Guard appears only in 12*c*R2.

Final word: remember that in addition to the snapshots you schedule (hourly for example) you can gather snapshots manually. Always try to analyse a report that covers a homogenous sample of your performance issue. With a good method to analyse them, the 20 years old Statspack reports are more powerful than most tuning tools found in other RDBMS. ■

ABOUT THE AUTHOR

Franck Pachot
Principal Consultant, dbi-services

Franck Pachot is principal consultant, trainer and technology leader at dbi-services in Switzerland. He has over 20 years of experience in Oracle databases in all areas from development, data modeling, performance, administration and training, Franck is an Oracle Certified Master 11*g* and 12*c* and an Oracle ACE Director. Franck is co-author of 'Oracle 12*c*R2 Multitenant' (https://www.amazon.com/Oracle-Database-Release-Multitenant-Press/dp/1259836096).

Blog: http://blog.pachot.net
https://ch.linkedin.com/in/franckpachot
@FranckPachot