

## **ORACLE database monitoring : In praise of the dumb agent**

*By Steven Wolfe, Development Manager, Oriole Corporation.*

Most products which monitor Oracle databases on a network boast 'intelligent agents'. In fact, as anybody knows, in real life cleverness can sometimes be more of a liability than an asset. This article shows how to use dumb agents (intelligently) and the flexibility they can bring to a fast-evolving organization. Our purpose was to set up Oracle monitoring on a WAN - with databases initially in London, New York, Paris and subsequently in Tokyo, Milan, Amsterdam and others. We wanted to monitor both 'classical' Oracle indicators (such as filling rates of tablespaces and hit-ratios) and also some application related values, to be stored in a specially created monitoring database (MDB) before generating HTML pages. As the applications were new, precisely what to monitor was unclear when we went in production and we wanted to be as flexible as possible.

Several options were available; the most obvious one was to have a big monitor.sql script and to run it from the monitoring machine through SQL\*Net and a bevy of database links. Here we had three concerns :

- Security - Monitoring usually requires connecting to accounts which can 'see' a lot and transferring passwords on the network could be a concern.
- Maintenance - Each time a new database goes live, the monitoring database must be modified (tnsnames.ora file, new database link, etc.)
- But chiefly, scheduling. Some monitoring queries can be pretty heavy and are better run when users are quietly sleeping in their beds. With a network on which the sun never sets, proper scheduling becomes mind-numbing. Moreover, we wouldn't have just one monitor.sql to run but in fact a collection of scripts to be run at different intervals (it's usually pointless to check tablespaces more than once a day but some application-related values may have to be collected every hour or half-hour).

Another possibility was to install and schedule (using local time) a monitoring tool on each database and, in a 'push' fashion as opposed to the 'pull' fashion described above, have a database automatically register itself and send data to the MDB when it goes live. Very nice, except that when you decide to modify something in what you monitor, you have to apply the modification everywhere. Not very pleasant when you are as lazy as this author and, of course, error-prone.

Therefore we pushed the preceding idea a little farther: why not have every database not only push the result of the monitoring queries to the MDB but pull the text of the query as well from the MDB beforehand? Using the DBMS\_SQL package, queries can be dynamically executed. Each time a new query is added, or a query changes, it is automatically taken into account by the databases we want to monitor. Given a few simple rules, the monitored database doesn't need to know anything about what it's doing - it just gets statements, executes them and stores the result in the MDB. Hence the dumb agent paradigm (smart word, paradigm).

There are not very many tables used and a minimum subset of what we implemented can be described as follows (a ✓ in column N means NOT NULL) (note: scripts to create these tables and the PL/SQL procedures referred to in this paper can all be freely downloaded from [www.oriolcorp.com](http://www.oriolcorp.com)) :

<b>REPORT_QUERIES</b>		<b>Definition of monitoring queries to be run at remote databases</b>	
<b>Column</b>	<b>N</b>	<b>Type</b>	
REPORT_CODE	✓	CHAR(3)	
REPORT_TITLE		VARCHAR2(100)	For displaying the results
REPORT_SHORT_TITLE		VARCHAR2(15)	Ditto
REPORT_HEADER		VARCHAR2(100)	Column names, separated by tabs
REPORT_FREQUENCY	✓	NUMBER	Frequency (in days or fractions of day) at which the query must be run.
REPORT_FIRST_TIME		DATE	To be sure that the query runs at a given time.
REPORT_EXEC_TYPE		CHAR(1)	NULL if to be executed by all databases, P if project-specific, B if database-specific (refers to tables not mentioned here).

<b>REPORT_QUERIES_TEXT</b>		<b>Monitoring queries to be run at remote databases</b>	
<b>Column</b>	<b>N</b>	<b>Type</b>	
REPORT_CODE	✓	CHAR(3)	
LINE#	✓	NUMBER	
TEXT		VARCHAR2(100)	Must return two columns, a number and a string <= 1000 characters in which fields are separated by tabs. Can be ordered.

REPORT_RESULT	Table where result of monitoring queries on remote databases is stored	
Column	N	Type
HOSTNAME	✓	VARCHAR2(64)
DBNAME	✓	VARCHAR2(9)
INSTANCE_NAME	✓	VARCHAR2(16)
TIMESTAMP	✓	DATE
REPORT_CODE		CHAR(3)
LINE#	✓	NUMBER
TEXT		VARCHAR2(1000) Tab separated result

Queries must obey a few very simple rules, as explained in the comments for REPORT\_QUERIES\_TEXT above:

they return two columns only,

one numeric (which may be used for ordering, for instance if we want to list tables by decreasing size) and

one which is a string up to 1000 characters long, which is also used as the minor sorting key.

This string will contain all the actual columns, separated by tabs. As the number and type of the columns is fixed, it becomes quite easy to use the DBMS\_SQL package to execute the queries dynamically without having the least idea of the data they are getting.

Now an example to see how all this works. Firstly we will consider the MDB side.

If we want the MDB to be accessed by the monitored databases, we must first create a special account for the database links and give this account the SELECT privilege on all the above tables, plus INSERT on REPORT\_RESULT and we must also create synonyms (public or private on the database link account).

Then, we must define what type of information we want to collect. Let's say that we want to list the users currently connected and (say) the first 50 characters of the statements they are executing. If we were to run the query under SQL\*Plus, we would normally write it as :

```
select s.username, s.status
      ,substr(a.sql_text, 1, 50) stmt
  from v$session s
      ,v$sqlarea a
 where s.username is not null           -- Ignore the Oracle
                                         -- background processes
      and s.audsid <> userenv('SESSIONID') -- Ignore the current
                                         -- session

      and s.sql_address = a.address
      and s.sql_hash_value = a.hash_value
 order by s.status                      -- to have active sessions first
```

Here, our conventions force us to write the query in a slightly different way (in fact, only the SELECT LIST and the ORDER BY clause differ) :

```
select decode(s.status, 'ACTIVE', 1, 2), -- NUMBER column for ordering
```

```

        s.username||chr(9)||s.status||chr(9)||
        substr(a.sql_text, 1, 50)           -- Single text column
from v$session s,
     v$sqlarea a
where s.username is not null
     and s.audsid <> userenv('SESSIONID')
     and s.sql_address = a.address
     and s.sql_hash_value = a.hash_value
order by 1, 2

```

Now, if we want the query to be executed every two hours on all the databases we can then store the following values in REPORT\_QUERIES :

```

insert into report_queries
(REPORT_CODE,REPORT_TITLE,REPORT_SHORT_TITLE
,REPORT_HEADER,REPORT_FREQUENCY,REPORT_FIRST_TIME,REPORT_EXEC_TYPE)
values
('WHO', 'Currently connected users', 'Sessions'
,'USERNAME      STATUS STATEMENT', 1/12, NULL, NULL);

```

and in REPORT\_QUERIES\_TEXT :

```

insert into report_queries_text(report_code, line#, text)
values('WHO', 1, 'select decode(s.status, 'ACTIVE', 1, 2),');
insert into report_queries_text(report_code, line#, text)
values('WHO', 2, 's.username||chr(9)||s.status||chr(9)||
  substr(a.sql_text, 1, 50)');
insert into report_queries_text(report_code, line#, text)
values('WHO', 3, 'from v$session s,');
insert into report_queries_text(report_code, line#, text)
values('WHO', 4, '      v$sqlarea a');
insert into report_queries_text(report_code, line#, text)
values('WHO', 5, 'where s.username is not null');
insert into report_queries_text(report_code, line#, text)
values('WHO', 6, '  and s.audsid <> userenv('SESSIONID')');
insert into report_queries_text(report_code, line#, text)
values('WHO', 7, '  and s.sql_address = a.address');
insert into report_queries_text(report_code, line#, text)
values('WHO', 8, '  and s.sql_hash_value = a.hash_value');
insert into report_queries_text(report_code, line#, text)
values('WHO', 9, 'order by 1, 2');

```

(a number of typical queries are also available at [www.oriolecorp.com](http://www.oriolecorp.com))

Basically, that's all you need on the server.

Now, let's turn our attention to a monitored database. Here, you have :

- 1 To create the database link required to connect to the MDB; of course, the MDB must be accessible through SQL\*Net.
- 2 Create the monitoring procedure check\_db by running check\_db.sql on the suitable account (typically, an externally identified DBA account)
- 3 **IMPORTANT** : SYS must grant SELECT ... WITH GRANT OPTION to the account above on ALL its views which are likely to be queried. When you create a stored procedure, you must have this privilege (at least) on all the tables referred to in the procedure and this holds true when tables are referred to in dynamically executed queries as well. Beware of the dynamic views too, which must not be

referred to as V\$VIEWNAME but as V\_\$VIEWNAME when granting privileges. So, if you want our preceding query to execute

```
grant SELECT on V_$SQLAREA to <suitable user> with grant option;
```

```
grant SELECT on V_$SESSION to <suitable user> with grant option;
```

- 4 Schedule the execution of check\_db at a rate at least equal to the highest frequency you may have. This can be done by an Oracle job (check file dbmsjob.sql in your \$ORACLE\_HOME/rdbms/admin directory if you use a Unix server), or an operating system utility such as crontab.

That's all. Detailing the source code for check\_db would be rather boring but the algorithm is quite obvious : it looks for all the queries which are supposed to be run on the current database and which have not been run recently (recently meaning later than SYSDATE - REPORT\_FREQUENCY). For each of the queries found, the text is obtained from REPORT\_QUERIES\_TEXT, the different lines are concatenated into a single string which is then executed. The result of the query is then stored to REPORT\_RESULT with a timestamp.

Now the result just has to be extracted from the MDB and neatly formatted but that's another story...

The Author.

Steven Wolfe is Development Manager of Oriole Corporation, a company which provides a range of fast, easy to use, efficiency enhancing Oracle database administration tools. Full details of the range of tools and a selection of free scripts is available on the Oriole webpage at [www.oriolcorp.com](http://www.oriolcorp.com).

Steven can be reached by e-mail at [swolfe@oriolcorp.com](mailto:swolfe@oriolcorp.com).