How to Cope with an ORA-01000 Error (*maximum open cursors exceeded*)

Version 2.1

Radoslav Rusinov
Radoslav.Rusinov.remove_spam.@gmail.com

Contents

Note: this document uses many quotes from several important Metalink notes and forum postings and is not fully authored by its creator. The role of the author is to make more valuable and full all-in-one source, related to the current problem than existing sources and articles about this topic, and to give appropriate advices and recommendations about it.

1. Introduction

What an Oracle DBA must do when face this kind of error.
Lets first take a look at the documentation:
ORA-01000 maximum open cursors exceeded
Cause: A host language program attempted to open too many cursors. The initialization parameter OPEN_CURSORS determines the maximum number of cursors per user.
Action: Modify the program to use fewer cursors. If this error occurs often, shut down Oracle, increase the value of OPEN_CURSORS, and then restart Oracle.

This error happens a lot in association with some kind of application.
This error also happens at the database level, with just regular inserts, updates, deletes, etc. in PL/SQL or in SQL*Plus, etc.
The reason you receive this error is because Oracle has reached the set limit for open cursors allowed for that executable or that user session. There are two kinds of open cursors: **implicit** and **explicit**. Here is some background on how cursors work.
To process a SQL statement, Oracle opens a work area called a private SQL area. This private SQL area stores information needed to execute a SQL statement. Cursors are stored in this area to keep track of information. An **IMPLICIT** cursor is declared for all data definition and data manipulation statements. These are internal to Oracle. For queries that return more than one row, you must declare an **EXPLICIT** cursor to retrieve all the information. You can tune explicit cursors more easily as you can decide when to open them and close them.
Implicit cursors are harder to tune because they are internal to Oracle. If the application is tuned carefully, it may cut down the number of implicit cursors opened.
Keep in mind that the maximum number of allowed cursors is per session, not per instance.

2. Workarounds

There are two ways to workaround this ORA-01000 error. You can tune cursor usage at the database level and at the application level.

2. 1. Tuning at the DATABASE LEVEL

There is a parameter you can set in the init.ora that determines the number of cursors a user can open in a session: OPEN_CURSORS.
OPEN_CURSORS by default is 50 and usually, this is not high enough. The highest value you can set this parameter to is operating system dependant. To solve the ORA-01000 error, set the OPEN_CURSORS to a higher number. You may need to set it to the maximum of the operating system limit.
In many systems it is set to 1000 (or even more – 3000, 50000) without any problem. But setting of this parameter to values more than 1000 must be discussed, if there is such need, it will means that something is wrong with the application that leads to that error. Then the application code must be revised instead of changing of this parameter to higher values. Even values more than 300 could be considered as bigger.
Consequences to changing this parameter:

This parameter does not affect performance in any way but Oracle will now need a little more memory to store the cursors. It will affect only used memory, not the resources that Oracle will need to support this increased value.

2. 2. Tuning at the APPLICATION LEVEL

For this level is responsible the development team.

3. Monitoring and detecting of the causing problem

The Oracle DBA must try to identify and localize the problem and if it cannot be solved by himself alone then he/she must report it to the development and provide them with all needed information like: frequency of the error, some regularity in appearing of the error, behavior, results from executed scripts and trace files. This will help the problem to be easily identified and solved if it is coming from the application software.

3.1. Using scripts

To be able to resolve the problem, a DBA could use several scripts that will help to him/her in that situation.
1. Use the following view to see detailed information about opened cursors at the moment

```
SELECT * FROM v$open_cursor
/
```

2. Use the following script to find the count of all currently opened cursors and to see the cumulative amount too:

```
SELECT N.NAME, S.VALUE
FROM V$STATNAME N , V$SYSSTAT S
WHERE N.STATISTIC# = S.STATISTIC# AND
S.STATISTIC# in (2,3)
/

Results:
```

1	opened cursors cumulative	207067
2	opened cursors current	748

3. The following SQL displays a list of unique cursors within a session that are opened multiple times. Usually due to the client opening a cursor, using it (iterating through the cursor until end-of-cursor) and not closing it. Then it proceeds to use the same cursor again (with different criteria for the predicate bind variables) and repeat the process.
You will typical find this an issue with clients using ref cursors.

```
SELECT C.SID AS "OraSID",
       C.ADDRESS || ':' || C.HASH_VALUE AS "SQL Address",
       COUNT(C.SADDR) AS "Cursor Copies"
  FROM V$OPEN_CURSOR C
 GROUP BY C.SID, C.ADDRESS || ':' || C.HASH_VALUE
HAVING COUNT(C.SADDR) > 2
 ORDER BY 3 DESC
/

Results:
```

1	21	6B422C38:1053795750	8
2	21	6B3D3040:589337545	7
3	18	6B422C38:1053795750	6
4	27	6B422C38:1053795750	6
5	22	6B3D3040:589337545	5
6	22	6B422C38:1053795750	5
7	36	6B422C38:1053795750	5
8	27	6B3D3040:589337545	4

4. To allow the DBA/Analysts to check whether the session cursor cache or open cursors are really a constraint and then increase the parameter **session_cached_cursors** or **open_cursors** accordingly.
SESSION_CACHED_CURSORS lets you specify the number of session cursors to cache.
Repeated parse calls of the same SQL statement cause the session cursor for that statement to be moved into the session cursor cache.
Subsequent parse calls will find the cursor in the cache and do not need to reopen the cursor and even do a soft parse.
The session cursor cache can be constrained by either the **session_cached_cursors** parameter, or the **open_cursors** parameter. This script reports the current maximum usage in any session with respect to these limits.
If either of the Usage column figures approaches 100%, then the corresponding parameter should normally be increased.

```
SELECT
  'session_cached_cursors'  parameter,
  LPAD(value, 5)  value,
  DECODE(value, 0, '  n/a', to_char(100 * used / value, '990') ||
'%')  usage
FROM
  (SELECT
      MAX(s.value)  used
    FROM
      v$statname  n,
      v$sesstat  s
    WHERE
      n.name = 'session cursor cache count' and
      s.statistic# = n.statistic#
  ),
  (SELECT
      value
    FROM
```

```
      v$parameter
    WHERE
      name = 'session_cached_cursors'
  )
UNION ALL
SELECT
  'open_cursors',
  LPAD(value, 5),
  to_char(100 * used / value,  '990') || '%'
FROM
  (SELECT
      MAX(sum(s.value))  used
    FROM
      v$statname  n,
      v$sesstat  s
    WHERE
      n.name in ('opened cursors current', 'session cursor cache
count') and
      s.statistic# = n.statistic#
    GROUP BY
      s.sid
  ),
  (SELECT
      value
    FROM
      v$parameter
    WHERE
      name = 'open_cursors'
  )
/


Results:
```

PARAMETER	VALUE	USAGE
session_cached_cursors	0	n/a
open_cursors	300	19%

Comments:
It tells you a fact (some ratios), doesn't tell you anything more. No more useful then "your buffer cache hit ratio is 99%". That is neither good, nor bad. It is not relevant in itself. There is one good ratio, from statspack -- the soft parse ratio. Not many other ratios mean anything by themselves.
You set open cursors so as to not hit "max open cursors".
You set session cached cursors to help poorly written applications run a bit better.

5. Additional info for open cursors:

```
SELECT b.SID, UPPER(a.NAME), b.VALUE
    FROM v$statname a, v$sesstat b, v$session c
    WHERE a.statistic# = b.statistic#
    AND c.SID = b.SID
    AND LOWER(a.NAME) LIKE '%' || LOWER('CURSOR')||'%'
    AND b.SID=20
UNION
SELECT SID,  'v$open_cursor opened cursor', COUNT(*)
FROM v$open_cursor
WHERE SID=20
GROUP BY SID
ORDER BY SID
/
```

Results (for session with SID=20):

SID	UPPER(A.NAME)	VALUE
20	CURSOR AUTHENTICATIONS	292
20	OPENED CURSORS CUMULATIVE	2247
20	OPENED CURSORS CURRENT	477
20	SESSION CURSOR CACHE COUNT	0
20	SESSION CURSOR CACHE HITS	0
20	v$open_cursor COUNT(*)	16

6. The following script shows the percentage distribution of the total parse calls between hard and soft parses and also reports the percentage of total parse calls satisfied by the session cursor cache.

```
SELECT
  TO_CHAR(100 * sess / calls, '999999999990.00') || '%'
cursor_cache_hits,
 TO_CHAR (100 * (calls - sess - hard) / calls, '999990.00') || '%'
soft_parses,
 TO_CHAR (100 * hard / calls, '999990.00') || '%' hard_parses
FROM
  (SELECT value calls FROM v$sysstat WHERE name = 'parse count
(total)' ),
  (SELECT value hard  FROM v$sysstat WHERE name = 'parse count
(hard)' ),
 (SELECT value sess  FROM v$sysstat WHERE name = 'session cursor
cache hits' )
/
```

Results:

CURSOR_CACHE_HITS	SOFT_PARSES	HARD_PARSES
0.00%	98.84%	1.16%

7. The following script lists the cursors opened by all session connected to the database.
The user executing the script requieres select privileges on views views

sys.v$session, sys.v$sesstat and sys.v$statname. Be aware that this script will create view within the used schema.

```
DROP VIEW user_cursors;
CREATE VIEW user_cursors AS
 SELECT
    ss.username||'('||se.sid||') ' user_process,
SUM(DECODE(name,'recursive calls',value)) "Recursive Calls",
    SUM(DECODE(name,'opened cursors cumulative',value)) "Opened
Cursors",
    SUM(DECODE(name,'opened cursors current',value)) "Current
Cursors"
 FROM v$session ss, v$sesstat se, v$statname sn
 WHERE  se.statistic# = sn.statistic#
               AND (name like '%opened cursors current%'
                              OR name like '%recursive calls%'
                              OR name like '%opened cursors
cumulative%')
               AND se.sid = ss.sid
               AND ss.username is not null
GROUP BY ss.username||'('||se.sid||') ';

ttitle 'Per Session Current Cursor Usage '
column USER_PROCESS format a25;
column "Recursive Calls" format 999,999,999;
column "Opened Cursors"  format 99,999;
column "Current Cursors"  format 99,999;

SELECT * FROM user_cursors
 ORDER BY "Recursive Calls" DESC;
```

8. The following script reports the number of open cursors per user/SID.

```
COLUMN user_name    heading Username
COLUMN num          heading "Open Cursors"
SET lines 80 pages 59
SELECT user_name, sid,COUNT (*) num
  FROM v$open_cursor
 GROUP BY user_name,sid;
CLEAR columns
TTITLE off
SET pages 22
```

Results:

USER_NAME	SID	NUM
USER1	21	50
USER2	22	9
USER1	23	53
USER3	30	50

9. To get the session info, you can use the following script:

```
SET PAGESIZE 9999;
SET VERIFY off;
SET FEEDBACK off;
SET LINESIZE 132;
--
COLUMN uname      HEADING  'User|Name'              FORMAT A11;
COLUMN sid        HEADING  'Oracle|SID'             FORMAT 999999;
COLUMN pid        HEADING  'Oracle|PID'             FORMAT 999999;
COLUMN sernum     HEADING  'Serial#'                FORMAT 999999;
COLUMN spid       HEADING  'Server|PID'             FORMAT 999999;
COLUMN suser      HEADING  'Client|OS User'         FORMAT A8;
COLUMN status     HEADING  'Status'                 FORMAT A8;
COLUMN server     HEADING  'Server'                 FORMAT A9;
COLUMN process    HEADING  'Client|Process'         FORMAT A8;
COLUMN smach      HEADING  'Client|Machine'         FORMAT A15   WORD_WRAPPED;
COLUMN sprog      HEADING  'Program'                FORMAT A15   WORD_WRAPPED;
COLUMN cli        HEADING  'Client|Info'            FORMAT A15   WORD_WRAPPED;
SELECT a.username uname,
       a.sid sid,
       b.pid pid,
       a.serial# sernum,
       b.spid spid,
       a.osuser suser,
       a.status status,
       a.server server,
       a.machine smach,
       a.process process,
       a.program sprog,
       a.client_info cli
  FROM v$session a,
       v$process b
 WHERE a.paddr = b.addr
   AND a.type != 'BACKGROUND'
   ORDER BY a.sid
/
```

Results:

UNAME	SID	PID	SERNUM	SPID	SUSER	STATUS	SERVER	…	…	SPROG
IBANK	12	23	1880	2904	SYSTEM	INACTIVE	DEDICATED	…	196:1756	apache.exe
…	…	…	…	…	…	…	…	…	…	…

3.2. Start tracing

For deeper investigation of the problem, you can start **tracing** the problem session or the instance in order to identify the problem. For more details about these actions, see the point **Tracing** in **Advanced Topics** subject bellow.

4. Advanced Topics

4.1. Tracing

Instance-level tracing: If it is proving difficult to identify the reason for the ORA-01000 error then it is possible to get the user session to generate a trace file when the error occurs by adding the following event to the init.ora. See the **Metalink Note 75713.1** (http://metalink.oracle.com/metalink/plsql/showdoc?db=NOT&id=75713.1&blackframe=1) before adding any event to the init.ora file

event="1000 trace name errorstack level 3"

This will cause a trace file to be written by any session when it hits an ORA-01000. Provided MAX_DUMP_FILE_SIZE is large enough this trace should help identify what all cursors in the session are being used for and hence help identify the cause of the ORA-01000.
User-level tracing: If you already know what user activity is causing problem (for example some batch job or performing certain activities via the user interface) then you can start session tracing for the particular end user. You can use the following scripts that are using the dbms_system package. Before run them, find the needed SID and SERIAL# (i.e. 27 and 7180) of the session that must be traced.

```
-- to set timed statistics
exec dbms_system.set_bool_param_in_session(27,7180,'timed_statistics',true);
-- to set maximum trace file size
exec dbms_system.set_int_param_in_session(27,7180,'max_dump_file_size',2147483647);
-- to start trace
exec dbms_system.set_sql_trace_in_session(27,7180,true);
-- to stop trace
exec dbms_system.set_sql_trace_in_session(27,7180,false);
```

For more details, you can use the following Metalink note - **How to Create a SQL Trace from Another Session**:
http://metalink.oracle.com/metalink/plsql/ml2_documents.showDocument?p_database_id=NOT&p_id=100883.1

4.2. PL/SQL Cursor Sharing

From the Oracle documentation:
OPEN_CURSORS specifies the maximum number of open cursors (handles to private SQL areas) a session can have at once. You can use this parameter to prevent a session from opening an excessive number of cursors. This parameter also constrains the size of the PL/SQL cursor cache, which PL/SQL uses to avoid having to reparse as statements are reexecuted by a user.

SESSION_CACHED_CURSORS lets you specify the number of session cursors to cache. Repeated parse calls of the same SQL statement cause the session cursor for that statement to be moved into the session cursor cache. Subsequent parse calls will find the cursor in the cache and do not need to reopen the cursor.

4.2.1. Simple schema of PL/SQL and session cursor sharing

```
              +----------+       +--------+
              |  PL/SQL|      | PL/SQL|   (a)
         +-> |  engine   | -->| cursor   | -+
              |              |      | cache    |   |        +---------+      +--------+
              |+----------+      +--------+     |        | session |       | SQL     |
              |                                     +--->|cursor    |-->| engine |
              |                                              | cache   |      |           |
 connection |                        (b) |                  +---------+      +--------+
------------+------------>------------------+
```

PL/SQL have a cursor cache:
1) Explicit cursors should be closed by CLOSE <cursor> statement as PL/SQL documentation state this.
2) PL/SQL programs can and should be written without considering this cursor cache. **It is not a documented feature**.
3) V$open_cursors shows the open cursors that are in this cache.

4.2.2. Oracle 9.2.0.5 and PL/SQL Cursor Sharing

<u>Quotes from 9.2.0.5 Patchset Release notes:</u> Prior to release of the 9.2.0.5.0 patch set, the maximum number of cursors that could be cached for fast lookup by PL/SQL was bounded by the value of the init.ora parameter open_cursors. If you currently have open_cursors set to a high value (for example, greater than 1000), it is likely that this is causing large numbers of PL/SQL cursors to be cached in the shared pool. This could lead to issues with memory management, frequent reloading of library cache objects and ORA-04031 errors.
Patch set 9.2.0.5.0 alleviates the issue by changing the init.ora parameter which determines the upper bound for PL/SQL cursor caching from open_cursors to session_cached_cursors.
Most users will not need to modify the value of either of these parameters. If you already have session_cached_cursors set to a value greater than the open_cursors parameter, then this change will have no performance impact upon your system.
However, if you have session_cached_cursors set to zero, or set at a value significantly lower than the open_cursors parameter, and you are concerned that PL/SQL cursors need to be cached for optimal performance, and then you should ensure that the session_cached_cursors parameter is increased appropriately.
This issue is bug number 3150705.

User comment: On almost all the instances, we have session_cached_cursors set to a low value and open_cursors to an high value (to let the PL/SQL cache everything that it wants to). We must definitely increase session_cached_cursors, otherwise the PL/SQL cursor cache would be, in practice, disabled. We have never experienced "issues with memory management, frequent reloading of library cache objects and ORA-04031 errors", so for sure the current level of caching is safe.

Tom Kyte comment: I'm of the school that PL/SQL cursors *should be cached* (i don't think i like this "fix"). Setting session_cached_cursors to 100 is in general a good rule of thumb in general.

4.2.3. PL/SQL Cached Cursors vs. Session Cached Cursors

Tom Kyte comment: It does seem that the PL/SQL cursors can be in the session cursor cache (in 10g anyhow).

Tom Kyte comment: PL/SQL cached cursors are not the same as session cached cursors at all, they predate that init.ora parameter by many releases. Different concept all together - session_cached_cursors is for 3gls that do explicit cursor management.

Discussion: On a 10g system, let's say open_cursors = 1000, session_cached_cursors = 50.

Question 1: If a 3GL opens up 1000 cursors, can PL/SQL still open up 50 more? Or will PL/SQL get an ORA-1000 error when it tries to open a cursor? In other words, is the actual maximum number of open cursors = open_cursors + session_cached_cursors? Or is the maximum number of open cursors simply = open_cursors, regardless of whether the cursor is a 3GL cursor or a PL/SQL cursor? I would guess the latter, but don't have a 10g system to test it on.

Answer 1: No. PL/SQL will get ORA-01000 due to the fact that the 3gl which does not participate in Oracles cursor cache has all of the slots filled up with it's stuff. The actual max is open_cursors. It is the hard limit.

Question 2: Can open_cursors in 10g still "steal a cursor" from the PL/SQL cursor cache, or did that behavior stop in conjunction with the recent change in the effects of the session_cached_cursors parameter? For example, if PL/SQL opens up 50 cursors, and then a 3GL opens up 950 cursors, what happens when the
3GL tries to open up the 951st cursor? Does the 3GL steal a cursor from the PL/SQL cursor cache? Or does the 3GL get an ORA-1000 error?

Answer 2: It still "steals". 3gl will steal a slot.

Question 3: Is it correct to state: The PL/SQL cursor cache and the session cursor cache are two separate caches. Both caches are controlled by the same parameter, session_cached_cursors. However, the value of session_cached_cursors applies separately to both caches. If session_cached_cursors = 50, there can be 50 open cursors in the PL/SQL cursor cache. Simultaneously, there can be 50 cursors in in the session cursor cache.

In other words, the session_cached_cursors value of 50 does is not a limit on the sum of the cursors in the PL/SQL cursor cache + the cursors in the session cursor cache. Rather, it means that each cache can independently and simultaneously contain up to 50 cursors.

Answer 3: Yes, it is correct to state. However, the size of the PL/SQL cursor cache (which is a more direct link to the cursor than session cache cursors is) is now controlled by the value you set in session cached cursors.

4.3. Cursor reuse in PL/SQL static SQL

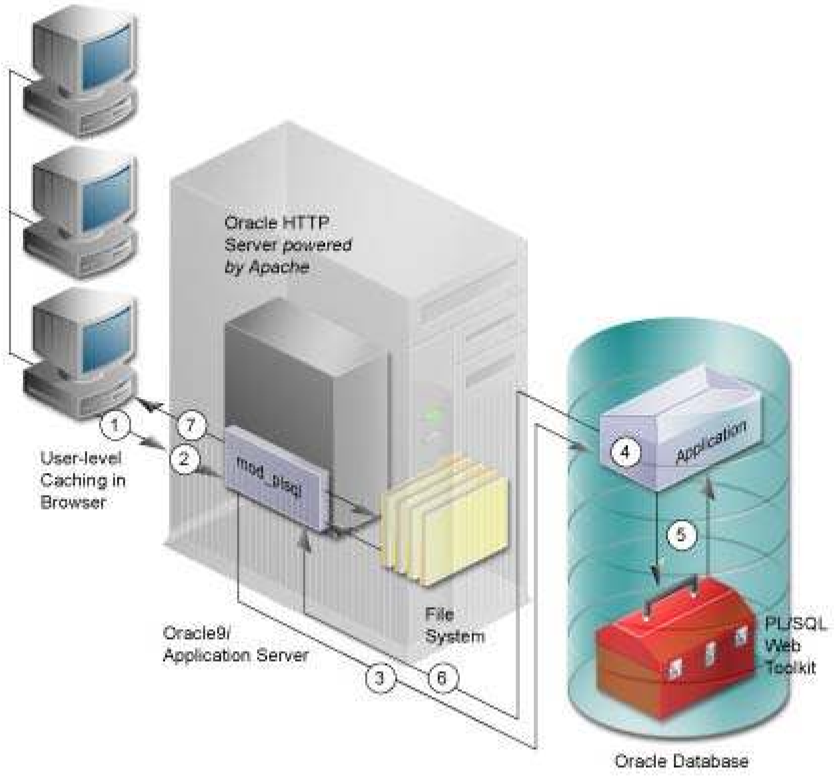
The Oracle instance maintains a system-wide LRU cache (the shared cursor cache) - exposed via v$sqlarea - of previously encountered SQL statements and appropriate derived information so that when the next SQL statement is submitted for parsing it is checked for match against the cached ones.
When the current statement is matched, the stored derived information (parse tree, execution plan, etc) is reused and computational cost is saved.
Moreover, the above processing has to be done in the context of a cursor which itself has to be opened and associated with the SQL statement in question, again at some cost. A given session might have one or several concurrently open cursors. Information on these, including the foreign key reference to **v$sqlarea** is exposed via **v$open_cursor**.
Programmers whose requirements can be satisfied by static SQL in a PL/SQL programming environment enjoy the benefits of an implicit implementation of this cost-saving paradigm while writing simple, easy-to-maintain, code.
Static SQL constructs in PL/SQL, using both explicit and implicit cursors, give you the benefits of the cost-saving paradigm without the effort of programming it.
This is achieved because the PL/SQL runtime system does not actually close your cursor (in the sense of the actions implemented by DBMS_SQL.CLOSE_CURSOR, (a hard-close) when the PL/SQL **CLOSE** statement is executed, or when an implicit cursor statement completes. Rather, it just soft-closes the cursor you think you've closed, *i.e.* it marks it as a candidate for later hard-close in the PL/SQL cursor cache, a LRU cache of potentially re-usable open cursors exposed via **v$open_cursor**.
It is possible to subvert this by careless programming, but it's easy to diagnose and correct such errors by monitoring v$open_cursor.

- Native dynamic SQL (EXECUTE IIMEDIATE), which often implies the use of a REF CURSOR opened with a dynamic string, does not implement the cost-saving paradigm since it's current exposure in PL/SQL (up to Oracle9*i*) does not give language constructs to distinguish between opening and parsing on the one hand and binding, executing and fetching on the other. However (as sated above) this penalty is offset by the greater efficiency due to its tighter integration.
- There is no reason ever to omit the close statement to balance the open for any of the above flavors of cursor construct. If you do omit it, you've programmed a potential memory leak: this is simply bad code! Nevertheless, the PL/SQL runtime system generally rescues you from such mistakes. However, you should not rely on this.

4.4. Connection pooling in MOD_PLSQL

4.4.1. Introduction

Usually when v$session is queried, all session that are coming from clients of the application (via web browser) are displayed as coming from "apache.exe" (in "program" column). The DBA must be aware of that when he/she is playing around with this or other problem that expect some knowledge about the application. The following information could be useful when some "user session" tracing is performed and during daily DBA activities. Monitoring of user sessions is important point for troubleshooting of the ORA-01000 error. In Windows OS, the opened sessions are at least equal to the number of opened sessions via web browser windows. This behavior is different in an UNIX environment. For more information, see the link from point 13 in the "Reference" section below.
From Oracle documentation: *Mod_plsql* is an Apache plug-in that communicates with the database. It maps browser requests into database stored procedure calls over a SQL*Net connection. It is generally indicated by a "*/pls*" virtual path.
The following scenario provides an overview of what steps occur when a server receives a client request:


Oracle HTTP Server powered by Apache
mod_plsql
User-level Caching in Browser
1
2
7
4
Application
5
PL/SQL Web Toolkit
File System
Oracle9i Application Server
3
6
Oracle Database

1. The Oracle HTTP Server receives a PL/SQL Server Page request from a client browser.
2. The Oracle HTTP Server routes the request to mod_plsql.
3. The request is forwarded by mod_plsql to the Oracle Database. By using the configuration information stored in your DAD, mod_plsql connects to the database.
4. Mod_plsql prepares the call parameters, and invokes the PL/SQL procedure in the application.
5. The PL/SQL procedure generates an HTML page using data and the PL/SQL Web Toolkit accessed from the database.
6. The response is returned to mod_plsql.
7. The Oracle HTTP Server sends the response to the client browser.

The procedure that mod_plsql invokes returns the HTTP response to the client. To simplify this task, mod_plsql includes the PL/SQL Web Toolkit, which contains a set of packages called the owa packages. Use these packages in your stored procedure to get information about the request, construct HTML tags, and return header information to the client. Install the toolkit in a common schema so that all users can access it.

4.4.2 Connection Pooling

MOD_PLSQL has built-in connection pooling. It can be set via the MOD_PLSQL configuration file. This setting and filename differs between the old pre 10G MOD_PLSQL and the 10G MOD_PLSQL (which also works with 9i databases).

Pre-10G MOD_PLSQL
config file: $ORACLE_HOME/Apache/modplsql/cfg/wdbsvr.app
parameter: reuse

10G MOD_PLSQL
config files:
- $ORACLE_HOME/Apache/modplsql/conf/plsql.conf
- $ORACLE_HOME/Apache/modplsql/conf/dads.conf

parameters:
- PlsqlIdleSessionCleanupInterval
- PlsqlMaxRequestsPerSession

It is also important to understand just exactly how Apache works and impacts the MOD_PLSQL pool. Apache itself has a pool, which services web browser requests. Let's say there are 100 processes in the pool. Apache process 1 gets a request from user A that must be serviced by MOD_PLSQL. MOD_PLSQL creates a pooled connection and logs on to the database X. That pooled connection is local to Apache process 1. None of 99 other processes can see it.
If user A sends another MOD_PLSQL request and it is serviced by Apache process 2, that process will cause another MOD_PLSQL pooled connection to database X.
The issue with this architecture (and any other app tier architecture) is that a pool of a pool is created.
The best way to address this from the Oracle side is to use MTS (Multithreaded Server).
The best way to address this from the app tier perspective is to close idle pooled connections after a certain time. And to use connections that can be re-used again and again, irrespective of who the web user is (i.e. single application connection to Oracle). This of course has a security/auditing impact that must be noted. MOD_PLSQL supports both these.

<u>From Oracle documentation:</u> The connection pooling logic in *mod_plsql* can be best explained with an example. Consider the following typical scenario:

1. The Oracle9*i* Application Server listener is started. There are no database connections in the connection pool maintained by mod_plsql.
2. A browser makes a mod_plsql request (R1) for Database Access Descriptor (DAD) D1.
3. One of the Oracle HTTP Server processes (httpd process P1) starts servicing the request R1.
4. mod_plsql in process P1 checks its connection pool and finds that there are no database connections in its pool for that user request.

5. Based on the information in DAD D1, mod_plsql in process P1 opens a new database connection, services the PL/SQL request, and adds the database connection to its pool.
6. From this point on, all subsequent requests to process P1 for DAD D1 can now make use of the database connection pooled by mod_plsql.
7. If a request for DAD D1 gets picked up by another process (process P2), then mod_plsql in process P2 opens its own database connection, services the request, and adds the database connection to its pool.
8. From this point on, all subsequent requests to process P2 for DAD D1 can now make use of the database connection pooled by mod_plsql.
9. Now, assume that a request R2 is made for DAD D2 and this request gets routed to process P1.
10. mod_plsql in process P1 does not have any database connections pooled for DAD D2, and a new database session is created for DAD D2 and pooled after servicing the request. Process P1 now has two database connections pooled, one for DAD D1 and another for DAD D2.

The important details in the previous example are:

- Each Oracle HTTP Server process serves all types of requests, such as static files requests, servlet requests, and mod_plsql requests. There is no control on which Oracle HTTP Server process services the next request.
- One Oracle HTTP Server process cannot use or share the connection pool created by another process.
- Each Oracle HTTP Server process pools at most one database connection for each DAD.
- User sessions are switched within a pooled database connection for a DAD. For DADs based on Oracle9*i*AS Single Sign-On (SSO), proxy authentication is used to switch the user session. For non-SSO users, using HTTP basic authentication with the username and password not in the DAD, users are re-authenticated on the same connection.
- Multiple DADs may point to the same database instance, but database connections are not shared across DADs even within the same process.
- Unused DADs do not result in any database connections.

In the worst case scenario, the total number of database connections that can be pooled by mod_plsql is a factor of the total number of active DADs multiplied by the number of Oracle HTTP Server (httpd) processes running at any given time for a single Oracle9*i* Application Server instance. If you have configured the Oracle HTTP Server processes to a high number, you need to configure the backend database to handle a corresponding amount of database sessions.
For example, if there are three Oracle9*i*AS instances configured to spawn a maximum of 50 httpd processes each, plus two active DADs, you need to set up the database to allow 300 (3*50*2) sessions. This number does not include any sessions that are needed to allow other applications to connect.
Because database connections cannot be shared across httpd processes, process-based platforms have more of a *Connection Reuse* feature than *Connection Pooling*. Note that this is an artifact of the process-model in Oracle HTTP Server. Whenever Oracle HTTP Server becomes threaded in the future, mod_plsql will allow for true connection pooling. If the number of database sessions is a concern, then refer to the "Two-Listener Strategy" for details on how to address this problem.

4.4.3. Closing Pooled Database Sessions

Pooled database sessions are closed under the following circumstances:

- When a pooled connection has been used to serve a configured number of requests

By default each connection pooled by mod_plsql is used to service a maximum of 1000 requests and then the database connection is shut down and re-established. This is done to make sure that any resource leaks in the PL/SQL application, or in the Oracle client/server side, do not adversely affect the system. This default of 1000 can be changed by tuning the DAD configuration parameter PlsqlMaxRequestsPerSession.

- When a pooled connection has been idle for an extended period of time.
  By default, each pooled connection gets automatically cleaned up after 15 minutes of idle time. This operation is performed by the cleanup thread in mod_plsql. For heavily loaded sites, each connection could get used at least once every 15 minutes and the connection cleanup might not happen for a long period of time. In such a case, the connection would get cleaned up based on the configuration of PlsqlMaxRequestsPerSession. This default of 15 minutes can be changed by tuning the mod_plsql configuration parameter PlsqlIdleSessionCleanupInterval. Consider increasing the default for better performance in cases where the site is not heavily loaded.
- When the Oracle HTTP Server process goes down.
  The Oracle HTTP Server configuration parameter MaxRequestsPerChild governs when an Oracle HTTP Server process will be shut down. For example, if this parameter is set to 5000, each Oracle HTTP Server process would serve exactly 5000 requests before it is shut down. Oracle HTTP Server processes could also start up and shut down as part of Oracle HTTP Server maintenance based on the configuration parameters MinSpareServers, MaxSpareServers, and MaxClients. For mod_plsql connection pooling to be effective, it is extremely important that Oracle HTTP Server in Oracle9*i*AS be configured such that each Oracle HTTP Server process remains active for some period of time. An incorrect configuration of Oracle HTTP Server could result in a setup where Oracle HTTP Server processes are heavily started up and shut down. Such a configuration would require that each new Oracle HTTP Server process replenish the connection pool before subsequent requests gain any benefit of pooling.

4.4.4. Connection Pooling and Oracle HTTP Server Configuration

- Creating a new database connection is an expensive operation and it is best if every request does not have to open and close a database connection. The optimal technique is to make sure that database connections opened in one request are reused in subsequent requests. In some rare situations, where a database is accessed very infrequently and performance is not a major concern, connection pooling can be disabled. For example, if the administrator accesses a site infrequently to perform some administration tasks, then the DAD used to access the administrator applications can choose to disable connection pooling. This reduces the number of database sessions at the expense of performance.
- Oracle HTTP Server configuration should be properly tuned so that once processes are started up, the processes remains up for a while. Otherwise, the connection pooling in mod_plsql is rendered useless. The Oracle9*i*AS listener should not have to continually start up and shut down processes. A proper load analysis should be performed of the site to determine what the average load on the Web site. The Oracle HTTP Server configuration should be tuned such that the number of httpd processes can handle the average load on the system. In addition, the configuration parameter MaxClients in the httpd.conf file should be able to handle random load spikes as well.
- Oracle HTTP Server processes should be configured so that processes are eventually killed and restarted. This is required to manage any possible memory leaks in various components accessed through the Oracle HTTP Server. This is specifically required in mod_plsql to ensure that any

database session resource leaks do not cause a problem. Make sure that MaxRequestsPerChild configuration parameter is set to a high number. For mod_plsql applications, this should not be set to 0.
- For heavily loaded sites, the Oracle HTTP Server configuration parameter KeepAlive should be disabled. This ensures that each process is available to service requests from other clients as soon as a process is done with servicing the current request. For sites which are not heavily loaded, and where it is guaranteed that the number of Oracle HTTP Server processes are always greater than the number of simultaneous requests to the Oracle9*i*AS listener, enabling the KeepAlive parameter results in performance improvements. In such cases, make sure to tune the KeepAliveTimeout parameter appropriately.
- You may want to lower down the value of Timeout in the Oracle HTTP Server configuration. This ensures that Oracle HTTP Server processes are freed up earlier if a client is not responding in a timely manner. Do not set this value too low, otherwise slower responding clients may start getting timed out.

4.4.5. Tuning the Number of Database Sessions

- The processes parameter in the Oracle init$SID.ora configuration file should be set so that Oracle is able to handle the maximum number of database sessions. This number should be proportional to the number of DADs, maximum number of Oracle HTTP Server processes, and the number of Oracle9*i*AS instances.
- Using a two-listener strategy or using Multi Threaded Server (MTS) reduces the number of database sessions. See "Two-Listener Strategy".
- On Unix platforms, the connection pool is not shared across Oracle HTTP Server processes. For this reason, it is recommended that the application use as few DADs as possible.
- Front ending your Oracle HTTP Server with Oracle9*i*AS Web Cache reduces the requirement to have a high number of processes for your HTTP configuration, resulting in lesser number of database sessions.

Table 1-1 Database Access Descriptor (DAD) Parameters

Parameter	**Setting**
PlsqlAlwaysDescribeProcedure	Set this to "Off" for best performance.
PlsqlDatabaseConnectString	Use the host:port:sid format instead of a TNS entry
PlsqlFetchBufferSize	Default=128 For multi-byte character sets like Japanese, Chinese, setting this to 256 will give better performance
PlsqlIdleSessionCleanupInterval	Default=15 (minutes) Increasing this parameter allows a pooled database connection to stay around for longer times
PlsqlLogEnable	Default=off This parameter should be set to "Off" unless recommended by Oracle support for debugging purposes

Parameter	Setting
PlsqlMaxRequestsPerSession	Default=1000 If the PL/SQL application does not leak resources/memory, this parameter can be tuned higher (for example, 5000)
PlsqlNLSLanguage	Setting this parameter to match the database NLS language will disable overheads in character set conversions occurring in Oracle Net Services.
PlsqlSessionStateManagement	Set this parameter to "StatelessWithFastResetPackageState" if the database is 8.1.7.2 or above. Oracle9*i*AS Portal is not yet certified with the mode StatelessWithFastResetPackageState. For Oracle9*i*AS Portal, set this parameter to the value StatelessWithResetPackageState.

5. References (recommended and used articles, forum posts and comments)

1. Maximum open cursors exceeded (Metalink forum post) – URL: http://metalink.oracle.com/metalink/plsql/ml2_documents.showDocument?p_database_id=FOR&p_id=580829.993
2. Overview of ORA-1000 Maximum Number of Cursors Exceeded (Metalink Note) – URL: http://metalink.oracle.com/metalink/plsql/showdoc?db=NOT&id=1012266.6&blackframe=1
3. ORA-1000 maximum open cursors exceeded (Metalink Note) – URL: http://metalink.oracle.com/metalink/plsql/ml2_documents.showDocument?p_database_id=NOT&p_id=18591.1
4. Maximum number of cursors (asktom.oracle.com forum post) – URL: http://asktom.oracle.com/pls/ask/f?p=4950:8:15383759053304836584::NO::F4950_P8_DISPLAYID,F4950_P8_CRITERIA:2298290920735
5. Open cursors exceeded (asktom.oracle forum post) – URL: http://asktom.oracle.com/pls/ask/f?p=4950:8:15383759053304836584::NO::F4950_P8_DISPLAYID,F4950_P8_CRITERIA:1041031921901
6. PL/SQL cursor cache changes in 9.2.0.5 (asktom.oracle forum post) – URL: http://asktom.oracle.com/pls/ask/f?p=4950:8:::::F4950_P8_DISPLAYID:17989406187750
7. Followup Maximum number of cursors (asktom.oracle forum post) – URL: http://asktom.oracle.com/pls/ask/f?p=4950:8:15383759053304836584::NO::F4950_P8_DISPLAYID,F4950_P8_CRITERIA:2299599531923
8. Does PL/SQL Implicitly Close Cursors? (Article from Jonathan Gennick) – URL: http://www.gennick.com/open_cursors.html
9. When does PL/SQL close the cursors (Metalink note) – URL: http://metalink.oracle.com/metalink/plsql/ml2_documents.showDocument?p_database_id=NOT&p_id=166336.1

10. Cursor reuse in PL/SQL static SQL (Article from Bryn Llewellyn, PL/SQL Product Manager, Oracle Corp.) – URL: http://www.oracle.com/technology/sample_code/tech/pl_sql/htdocs/x/Cursors/start.htm
11. Connection pooling in MOD_PLSQL (Metalink forum post) – URL: http://metalink.oracle.com/metalink/plsql/ml2_documents.showDocument?p_database_id=FOR&p_id=536618.994
12. Connection pooling (Metalink forum post) – URL: http://metalink.oracle.com/metalink/plsql/ml2_documents.showDocument?p_database_id=FOR&p_id=275797.996
13. Optimizing Mod_plsql Database Connections on Unix (Metalink Note) – URL: http://metalink.oracle.com/metalink/plsql/ml2_documents.showDocument?p_database_id=NOT&p_id=150419.1
14. PL/SQL Connection Pooling in OAS vs. 9iAS (Metalink Note) – URL: http://metalink.oracle.com/metalink/plsql/showdoc?db=NOT&id=143430.1&blackframe=1
15. Oracle9*i* Application Server Performance Guide – URL: http://download-uk.oracle.com/docs/cd/A97329_03/core.902/a95102/toc.htm
16. Oracle9*i* Application Server mod_plsql User's Guide - URL: http://download-uk.oracle.com/docs/cd/A97329_03/web.902/a90855/toc.htm
17. Maximum Open Cursors Exceeded (Metalink forum post) http://metalink.oracle.com/metalink/plsql/ml2_documents.showDocument?p_database_id=FOR&p_id=3881.996
18. leaking recursive cursor (Metalink forum post) – URL: http://metalink.oracle.com/metalink/plsql/ml2_documents.showDocument?p_database_id=FOR&p_id=287121.999
19. Script to report open cursors by user (Metalink script) – URL: http://metalink.oracle.com/metalink/plsql/showdoc?db=NOT&id=111415.1
20. Script for Listing User Session Cursor Usage (Metalink script) – URL: http://metalink.oracle.com/metalink/plsql/showdoc?db=NOT&id=133936.1
21. SCRIPT - to Gauge the Impact of the SESSION_CACHED_CURSORS Parameter (Metalink script) – URL: http://metalink.oracle.com/metalink/plsql/showdoc?db=NOT&id=208918.1
22. SCRIPT - to Tune the 'SESSION_CACHED_CURSORS' and 'OPEN_CURSORS' Parameters (Metalink script) – URL: http://metalink.oracle.com/metalink/plsql/showdoc?db=NOT&id=208857.1
23. Important Customer Information about numeric EVENTS (Metalink Note) – URL: http://metalink.oracle.com/metalink/plsql/showdoc?db=NOT&id=75713.1&blackframe=1
24. How to Create a SQL Trace from Another Session (Metalink Note) – URL: http://metalink.oracle.com/metalink/plsql/ml2_documents.showDocument?p_database_id=NOT&p_id=100883.1