ORACLE Certified Professional

ORACLE E-Business Suite 11*i* Certified Professional

Author – ***A.Kishore/Sachin***
http://appsdba.info

Oracle Database Health check:

OPERATING SYSTEM:

1)Physical memory/ Load:

- Free:free command displays amount of total, free and used physical memory (RAM) in the system as well as showing information on shared memory, buffers, cached memory and swap space used by the Linux kernel.
  Usage:
  $ free -m

- vmstat:vmstat reports report virtual memory statistics, which has information about processes, swap, free, buffer and cache memory, paging space, disk IO activity, traps, interrupts, context switches and CPU activity
  Usage:
  $vmstat 5

- top:top command displays dynamic real-time view of the running tasks managed by kernel and in Linux system. The memory usage stats by top command include real-time live total, used and free physical memory and swap memory with their buffers and cached memory size respectively
  Usage:
  $top
- ps :ps command reports a snapshot on information of the current active processes. ps will show the percentage of memory resource that is used by each process or task running in the system. With this command, top memory hogging processes can be identified.
  Usage:
  $ps aux

2) OS Space threshold

Checking the OS space is available in all filesystems,specially the location which is having archive logs ,oracle Database files.We can use the below OS commands:
$df –h
$du –csh *

3) Top 10 process consuming memory:

We can display top 10 memory consuming processes as follows:

ps aux|head -1;ps aux|sort -m

4) Free volumes available:

We have to make sure Sufficient disk space is available on the mount points on each OS servers where the Database is up and running.

$df –h

5)Filesystem space:

Under normal threshold.Check the filesystem in the OS side whether the sufficient space is available at all mount points.

DATABASE :

6) Check extents / Pro active Space addition:

Check each of the Data, Index and temporary tablespaces for extend and blocks allocation details.

```
SET LINES 1000
SELECT SEGMENT_NAME,TABLESPACE_NAME, EXTENTS,BLOCKS
FROM DBA_SEGMENTS;

SELECT SEGMENT_NAME,TABLESPACE_NAME,EXTENTS,BLOCKS
FROM DBA_SEGMENTS WHERE TABLESPACE_NAME='STAR01D';
```

7) Check alert log for ORA- and warn messages:

Checking the alert log file regulary is a vital task we have to do.In the alert log files we have to looks for the following things:

- Look for any of the oracle related errors.
  Open the alert log file with less or more command and search for any ORA – errors. This will give you the error details and time of occurrence.

- Look for the Database level or Tablespace level changes
  Monitor the alert log file and search the file for each Day activities happening In the Database either whether it is bouncing of Database.Increase in the size of the tablespaces,Increase in the size of the Database parameters.In the 11g database we can look for TNS errors in the alert log file.

8) Major wait events (latch/enqueue/Lib cache pin):

We can check the wait events details with the help of below queries:

```
SELECT s.saddr, s.SID, s.serial#, s.audsid, s.paddr, s.user#, s.username,
s.command, s.ownerid, s.taddr, s.lockwait, s.status, s.server,
s.schema#, s.schemaname, s.osuser, s.process, s.machine, s.terminal,
UPPER (s.program) program, s.TYPE, s.sql_address, s.sql_hash_value,
s.sql_id, s.sql_child_number, s.sql_exec_start, s.sql_exec_id,
s.prev_sql_addr, s.prev_hash_value, s.prev_sql_id,
s.prev_child_number, s.prev_exec_start, s.prev_exec_id,
s.plsql_entry_object_id, s.plsql_entry_subprogram_id,
s.plsql_object_id, s.plsql_subprogram_id, s.module, s.module_hash,
s.action, s.action_hash, s.client_info, s.fixed_table_sequence,
s.row_wait_obj#, s.row_wait_file#, s.row_wait_block#,
s.row_wait_row#, s.logon_time, s.last_call_et, s.pdml_enabled,
s.failover_type, s.failover_method, s.failed_over,
s.resource_consumer_group, s.pdml_status, s.pddl_status, s.pq_status,
s.current_queue_duration, s.client_identifier,
s.blocking_session_status, s.blocking_instance, s.blocking_session,
s.seq#, s.event#, s.event, s.p1text, s.p1, s.p1raw, s.p2text, s.p2,
s.p2raw, s.p3text, s.p3, s.p3raw, s.wait_class_id, s.wait_class#,
s.wait_class, s.wait_time, s.seconds_in_wait, s.state,
s.wait_time_micro, s.time_remaining_micro,
s.time_since_last_wait_micro, s.service_name, s.sql_trace,
s.sql_trace_waits, s.sql_trace_binds, s.sql_trace_plan_stats,
s.session_edition_id, s.creator_addr, s.creator_serial#
FROM v$session s
WHERE ( (s.username IS NOT NULL)
AND (NVL (s.osuser, 'x') <> 'SYSTEM')
AND (s.TYPE <> 'BACKGROUND') AND STATUS='ACTIVE'
)
ORDER BY "PROGRAM";
```

The following query provides clues about whether Oracle has been waiting for library cache activities:

```
Select sid, event, p1raw, seconds_in_wait, wait_time
From v$session_wait
Where event = 'library cache pin'
And state = 'WAITING';
```

The below Query gives details of Users sessions wait time and state:

```
SELECT NVL (s.username, '(oracle)') AS username, s.SID, s.serial#, sw.event,
sw.wait_time, sw.seconds_in_wait, sw.state
FROM v$session_wait sw, v$session s
WHERE s.SID = sw.SID
ORDER BY sw.seconds_in_wait DESC;
```

9) Max Sessions:

There should not be more than 6 inactive sessions running for more than 8 hours in a database in order to minimize the consumption of CPU and I/O resources.

a) Users and Sessions CPU consumption can be obtained by below query:

```
Set lines 1000
select ss.username, se.SID,VALUE/100 cpu_usage_seconds
from v$session ss, v$sesstat se, v$statname sn
where se.STATISTIC# = sn.STATISTIC#
and NAME like '%CPU used by this session%'
and se.SID = ss.SID and ss.status='ACTIVE'
and ss.username is not null
order by VALUE desc;
```

b) Users and Sessions CPU and I/O consumption can be obtained by below query:

```
-- shows Day wise,User wise,Process id of server wise- CPU and I/O consumption
set linesize 140
col spid for a6
col program for a35 trunc
select p.spid SPID,to_char(s.LOGON_TIME,'DDMonYY HH24:MI')
date_login,s.username,decode(nvl(p.background,0),1,bg.description, s.program )
program,
```

```
ss.value/100 CPU,physical_reads disk_io,(trunc(sysdate,'J')-trunc(logon_time,'J')) days,
round((ss.value/100)/(decode((trunc(sysdate,'J')-
trunc(logon_time,'J')),0,1,(trunc(sysdate,'J')-trunc(logon_time,'J')))),2) cpu_per_day
from V$PROCESS p,V$SESSION s,V$SESSTAT ss,V$SESS_IO si,V$BGPROCESS bg
where s.paddr=p.addr and ss.sid=s.sid
and ss.statistic#=12 and si.sid=s.sid
and bg.paddr(+)=p.addr
and round((ss.value/100),0) > 10
order by 8;
```

10) Long running Jobs:

We can find out long running jobs with the help of the below query:

```
col username for a20
col message for a50
col remaining for 9999
select username,to_char(start_time, 'hh24:mi:ss dd/mm/yy') started,
time_remaining remaining, message
from v$session_longops
where time_remaining = 0
order by time_remaining desc;
```

11) Invalid objects:

We can check the invalid objects with the help of the below query:

```
select owner||' '||object_name||' '||created||' '||status from dba_objects where
status='INVALID';
```

12) Analyze Jobs (once in a week):

We need to analyze the jobs that are running once in a week as a golden rule.
The below steps can be considered for analyzing jobs.

Analyzing a Running Job

The status of a job or a task changes several times during its life cycle. A job can have the following as its status:
Scheduled: The job is created and will run at the specified time.
Running: The job is being executed and is in progress.
Initialization Error: The job or step could not be run successfully. If a step in a job fails initialization, the job status is Initialization Error.
Failed: The job was executed but failed.
Succeeded: The job was executed completely.
Stopped: The user canceled the job.
Stop Pending: The user has stopped the job. The already running steps are completing execution.
Suspended: This indicates that the execution of the job is deferred.
Inactive: This status indicates that the target has been deleted.
Reassigned: The owner of the job has changed.
Skipped: The job was not executed at the specified time and has been omitted.

The running jobs can be found out by the help of below query:

```
select sid, job,instance from dba_jobs_running;
```

We can find out the failed jobs and Broken jobs details with the help of the below query:

```
select job||' '||schema_user||' '||Broken||' '||failures||' '||what||' '||last_date||' '||last_sec
from dba_jobs;
```

13) Temp usage / Rollback segment/PGA usage:

We can get information of temporary tablespace usage details with the help of below query:

```
Set lines 1000
SELECT b.tablespace,
ROUND(((b.blocks*p.value)/1024/1024),2)||'M' "SIZE",
a.sid||','||a.serial# SID_SERIAL,
a.username,
a.program
FROM sys.v_$session a,
```

```
sys.v_$sort_usage b,
sys.v_$parameter p
WHERE p.name = 'db_block_size'
AND a.saddr = b.session_addr
ORDER BY b.tablespace, b.blocks;
```

We can get information of Undo tablespace usage details with the help of the below query:

```
set lines 1000
SELECT TO_CHAR(s.sid)||','||TO_CHAR(s.serial#) sid_serial,
NVL(s.username, 'None') orauser,
s.program,
r.name undoseg,
t.used_ublk * TO_NUMBER(x.value)/1024||'K' "Undo"
FROM sys.v_$rollname r,
sys.v_$session s,
sys.v_$transaction t,
sys.v_$parameter x
WHERE s.taddr = t.addr
AND r.usn = t.xidusn(+)
AND x.name = 'db_block_size';
```

We can get the PGA usage details with the help of the below query:

```
select st.sid "SID", sn.name "TYPE",
ceil(st.value / 1024 / 1024/1024) "GB"
from v$sesstat st, v$statname sn
where st.statistic# = sn.statistic#
and sid in
(select sid from v$session where username like UPPER('&user'))
and upper(sn.name) like '%PGA%'
order by st.sid, st.value desc;
Enter value for user: STARTXNAPP
```

14) Redo generation/Archive logs generation details:

We should make sure there should not be frequent log switch happening in a Database. If there are frequent log switches than archive logs might generate more, which may decrease the performance of the Database, however in a production Database log switches could vary depending upon the Server configuration between 5 to 20.

We can the log switch details with the help of the below query:

Redolog switch Datewise and hourwise:

```
------------------------------
set lines 120;
set pages 999;
select to_char(first_time,'DD-MON-RR') "Date",
to_char(sum(decode(to_char(first_time,'HH24'),'00',1,0)),'99') " 00",
to_char(sum(decode(to_char(first_time,'HH24'),'01',1,0)),'99') " 01",
to_char(sum(decode(to_char(first_time,'HH24'),'02',1,0)),'99') " 02",
to_char(sum(decode(to_char(first_time,'HH24'),'03',1,0)),'99') " 03",
to_char(sum(decode(to_char(first_time,'HH24'),'04',1,0)),'99') " 04",
to_char(sum(decode(to_char(first_time,'HH24'),'05',1,0)),'99') " 05",
to_char(sum(decode(to_char(first_time,'HH24'),'06',1,0)),'99') " 06",
to_char(sum(decode(to_char(first_time,'HH24'),'07',1,0)),'99') " 07",
to_char(sum(decode(to_char(first_time,'HH24'),'08',1,0)),'99') " 08",
to_char(sum(decode(to_char(first_time,'HH24'),'09',1,0)),'99') " 09",
to_char(sum(decode(to_char(first_time,'HH24'),'10',1,0)),'99') " 10",
to_char(sum(decode(to_char(first_time,'HH24'),'11',1,0)),'99') " 11",
to_char(sum(decode(to_char(first_time,'HH24'),'12',1,0)),'99') " 12",
to_char(sum(decode(to_char(first_time,'HH24'),'13',1,0)),'99') " 13",
to_char(sum(decode(to_char(first_time,'HH24'),'14',1,0)),'99') " 14",
to_char(sum(decode(to_char(first_time,'HH24'),'15',1,0)),'99') " 15",
to_char(sum(decode(to_char(first_time,'HH24'),'16',1,0)),'99') " 16",
to_char(sum(decode(to_char(first_time,'HH24'),'17',1,0)),'99') " 17",
to_char(sum(decode(to_char(first_time,'HH24'),'18',1,0)),'99') " 18",
to_char(sum(decode(to_char(first_time,'HH24'),'19',1,0)),'99') " 19",
to_char(sum(decode(to_char(first_time,'HH24'),'20',1,0)),'99') " 20",
to_char(sum(decode(to_char(first_time,'HH24'),'21',1,0)),'99') " 21",
to_char(sum(decode(to_char(first_time,'HH24'),'22',1,0)),'99') " 22",
to_char(sum(decode(to_char(first_time,'HH24'),'23',1,0)),'99') " 23"
from v$log_history
group by to_char(first_time,'DD-MON-RR')
order by 1
/
```

Archive logs generations is directly proportional to the number of log switches happening in a Database. If there are frequent log switches than archive logs might generate more which can affect the performance of Database.

We can use the below queries for archive logs generation details:

a) Archive logs by dates:

```
set lines 1000
select to_char(first_time,'DD-MON-RR') "Date",
to_char(sum(decode(to_char(first_time,'HH24'),'00',1,0)),'99') " 00",
to_char(sum(decode(to_char(first_time,'HH24'),'01',1,0)),'99') " 01",
to_char(sum(decode(to_char(first_time,'HH24'),'02',1,0)),'99') " 02",
to_char(sum(decode(to_char(first_time,'HH24'),'03',1,0)),'99') " 03",
to_char(sum(decode(to_char(first_time,'HH24'),'04',1,0)),'99') " 04",
to_char(sum(decode(to_char(first_time,'HH24'),'05',1,0)),'99') " 05",
to_char(sum(decode(to_char(first_time,'HH24'),'06',1,0)),'99') " 06",
to_char(sum(decode(to_char(first_time,'HH24'),'07',1,0)),'99') " 07",
to_char(sum(decode(to_char(first_time,'HH24'),'08',1,0)),'99') " 08",
to_char(sum(decode(to_char(first_time,'HH24'),'09',1,0)),'99') " 09",
to_char(sum(decode(to_char(first_time,'HH24'),'10',1,0)),'99') " 10",
to_char(sum(decode(to_char(first_time,'HH24'),'11',1,0)),'99') " 11",
to_char(sum(decode(to_char(first_time,'HH24'),'12',1,0)),'99') " 12",
to_char(sum(decode(to_char(first_time,'HH24'),'13',1,0)),'99') " 13",
to_char(sum(decode(to_char(first_time,'HH24'),'14',1,0)),'99') " 14",
to_char(sum(decode(to_char(first_time,'HH24'),'15',1,0)),'99') " 15",
to_char(sum(decode(to_char(first_time,'HH24'),'16',1,0)),'99') " 16",
to_char(sum(decode(to_char(first_time,'HH24'),'17',1,0)),'99') " 17",
to_char(sum(decode(to_char(first_time,'HH24'),'18',1,0)),'99') " 18",
to_char(sum(decode(to_char(first_time,'HH24'),'19',1,0)),'99') " 19",
to_char(sum(decode(to_char(first_time,'HH24'),'20',1,0)),'99') " 20",
to_char(sum(decode(to_char(first_time,'HH24'),'21',1,0)),'99') " 21",
to_char(sum(decode(to_char(first_time,'HH24'),'22',1,0)),'99') " 22",
to_char(sum(decode(to_char(first_time,'HH24'),'23',1,0)),'99') " 23"
from v$log_history
group by to_char(first_time,'DD-MON-RR')
order by 1
/
```

b) Archive log generation details Day-wise :

```
select to_char(COMPLETION_TIME,'DD-MON-YYYY'),count(*)
from v$archived_log group by to_char(COMPLETION_TIME,'DD-MON-YYYY')
order by to_char(COMPLETION_TIME,'DD-MON-YYYY');
```

c) Archive log count of the day:

```
select count(*)
from v$archived_log
where trunc(completion_time)=trunc(sysdate);
```

count of archived logs generated today on hourly basis:

```
select to_char(first_time,'DD-MON-RR') "Date",
to_char(sum(decode(to_char(first_time,'HH24'),'00',1,0)),'99') " 00",
to_char(sum(decode(to_char(first_time,'HH24'),'01',1,0)),'99') " 01",
to_char(sum(decode(to_char(first_time,'HH24'),'02',1,0)),'99') " 02",
to_char(sum(decode(to_char(first_time,'HH24'),'03',1,0)),'99') " 03",
to_char(sum(decode(to_char(first_time,'HH24'),'04',1,0)),'99') " 04",
to_char(sum(decode(to_char(first_time,'HH24'),'05',1,0)),'99') " 05",
to_char(sum(decode(to_char(first_time,'HH24'),'06',1,0)),'99') " 06",
to_char(sum(decode(to_char(first_time,'HH24'),'07',1,0)),'99') " 07",
to_char(sum(decode(to_char(first_time,'HH24'),'08',1,0)),'99') " 08",
to_char(sum(decode(to_char(first_time,'HH24'),'09',1,0)),'99') " 09",
to_char(sum(decode(to_char(first_time,'HH24'),'10',1,0)),'99') " 10",
to_char(sum(decode(to_char(first_time,'HH24'),'11',1,0)),'99') " 11",
to_char(sum(decode(to_char(first_time,'HH24'),'12',1,0)),'99') " 12",
to_char(sum(decode(to_char(first_time,'HH24'),'13',1,0)),'99') " 13",
to_char(sum(decode(to_char(first_time,'HH24'),'14',1,0)),'99') " 14",
to_char(sum(decode(to_char(first_time,'HH24'),'15',1,0)),'99') " 15",
to_char(sum(decode(to_char(first_time,'HH24'),'16',1,0)),'99') " 16",
to_char(sum(decode(to_char(first_time,'HH24'),'17',1,0)),'99') " 17",
to_char(sum(decode(to_char(first_time,'HH24'),'18',1,0)),'99') " 18",
to_char(sum(decode(to_char(first_time,'HH24'),'19',1,0)),'99') " 19",
to_char(sum(decode(to_char(first_time,'HH24'),'20',1,0)),'99') " 20",
to_char(sum(decode(to_char(first_time,'HH24'),'21',1,0)),'99') " 21",
to_char(sum(decode(to_char(first_time,'HH24'),'22',1,0)),'99') " 22",
to_char(sum(decode(to_char(first_time,'HH24'),'23',1,0)),'99') " 23"
from v$log_history
where to_char(first_time,'DD-MON-RR')='16-AUG-10'
group by to_char(first_time,'DD-MON-RR')
order by 1
/
```

15) I/O Generation:

We can find out CPU and I/O generation details for all the users in the Database with the help of the below query:

```
-- Show IO per session,CPU in seconds, sessionIOS.
set linesize 140
col spid for a6
col program for a35 trunc
select p.spid SPID,to_char(s.LOGON_TIME,'DDMonYY HH24:MI')
date_login,s.username,decode(nvl(p.background,0),1,bg.description, s.program )
program,
ss.value/100 CPU,physical_reads disk_io,(trunc(sysdate,'J')-trunc(logon_time,'J'))
days,
round((ss.value/100)/(decode((trunc(sysdate,'J')-
trunc(logon_time,'J')),0,1,(trunc(sysdate,'J')-trunc(logon_time,'J')))),2) cpu_per_day
from V$PROCESS p,V$SESSION s,V$SESSTAT ss,V$SESS_IO si,V$BGPROCESS bg
where s.paddr=p.addr and ss.sid=s.sid
and ss.statistic#=12 and si.sid=s.sid
and bg.paddr(+)=p.addr
and round((ss.value/100),0) > 10
order by 8;
```

To know what the session is doing and what kind of sql it is using:

```
-- what kind of sql a session is using
set lines 9999
set pages 9999

select s.sid, q.sql_text from v$sqltext q, v$session s
where q.address = s.sql_address
and s.sid = &sid order by piece;
```

16) Sync arch:

In a dataguard environment we have to check primary is in sync with the secondary database.This we can check as follows:

The V$ MANAGED_STANDBY view on the standby database site shows you the activities performed by both redo transport and Redo Apply processes in a Data Guard environment.

```
SELECT PROCESS, CLIENT_PROCESS, SEQUENCE#, STATUS FROM V$MANAGED_STANDBY;
```

In some situations, automatic gap recovery may not take place and you will need to perform gap recovery manually. For example, you will need to perform gap recovery

manually if you are using logical standby databases and the primary database is not available.
The following sections describe how to query the appropriate views to determine which log files are missing and perform manual recovery.

On a physical standby database:

To determine if there is an archive gap on your physical standby database, query the V$ARCHIVE_GAP view as shown in the following example:
SQL> SELECT * FROM V$ARCHIVE_GAP;

If it displays no rows than the primary Database is in sync with the standy Database.If it display any information with row than manually we have to apply the archive logs.

After you identify the gap, issue the following SQL statement on the primary database to locate the archived redo log files on your primary database (assuming the local archive destination on the primary database is LOG_ARCHIVE_DEST_1):
Eg:
SELECT NAME FROM V$ARCHIVED_LOG WHERE THREAD#=1 AND DEST_ID=1 AND SEQUENCE# BETWEEN 7 AND 10;
Copy these log files to your physical standby database and register them using the ALTER DATABASE REGISTER LOGFILE statement on your physical standby database. For example:
SQL> ALTER DATABASE REGISTER LOGFILE
'/physical_standby1/thread1_dest/arcr_1_7.arc';
SQL> ALTER DATABASE REGISTER LOGFILE
'/physical_standby1/thread1_dest/arcr_1_8.arc';

After you register these log files on the physical standby database, you can restart Redo Apply. The V$ARCHIVE_GAP fixed view on a physical standby database only returns the next gap that is currently blocking Redo Apply from continuing. After resolving the gap and starting Redo Apply, query the V$ARCHIVE_GAP fixed view again on the physical standby database to determine the next gap sequence, if there is one. Repeat this process until there are no more gaps.

On a logical standby database:
To determine if there is an archive gap, query the DBA_LOGSTDBY_LOG view on the logical standby database. For example, the following query indicates there is a gap in the sequence of archived redo log files because it displays two files for THREAD 1 on the logical standby database. (If there are no gaps, the query will show only one file for each thread.) The output shows that the highest registered file is sequence number 10, but there is a gap at the file shown as sequence number 6:

```
SQL> COLUMN FILE_NAME FORMAT a55
SQL> SELECT THREAD#, SEQUENCE#, FILE_NAME FROM DBA_LOGSTDBY_LOG L
2> WHERE NEXT_CHANGE# NOT IN
3> (SELECT FIRST_CHANGE# FROM DBA_LOGSTDBY_LOG WHERE L.THREAD# = THREAD#)
4> ORDER BY THREAD#,SEQUENCE#;

THREAD# SEQUENCE# FILE_NAME
---------- ---------- -----------------------------------------------
1 6 /disk1/oracle/dbs/log-1292880008_6.arc
1 10 /disk1/oracle/dbs/log-1292880008_10.arc
```

Copy the missing log files, with sequence numbers 7, 8, and 9, to the logical standby system and register them using the ALTER DATABASE REGISTER LOGICAL LOGFILE statement on your logical standby database. For example:
SQL> ALTER DATABASE REGISTER LOGICAL LOGFILE '/disk1/oracle/dbs/log-1292880008_10.arc';

After you register these log files on the logical standby database, you can restart SQL Apply.

The DBA_LOGSTDBY_LOG view on a logical standby database only returns the next gap that is currently blocking SQL Apply from continuing. After resolving the identified gap and starting SQL Apply, query the DBA_LOGSTDBY_LOG view again on the logical standby database to determine the next gap sequence, if there is one. Repeat this process until there are no more gaps.
Monitoring Log File Archival Information:

Step 1 : Determine the current archived redo log file sequence numbers.
Enter the following query on the primary database to determine the current archived redo log file sequence numbers:

SQL> SELECT THREAD#, SEQUENCE#, ARCHIVED, STATUS FROM V$LOG
WHERE STATUS='CURRENT';

Step 2 : Determine the most recent archived redo log file.

Enter the following query at the primary database to determine which archived redo log file contains the most recently transmitted redo data:

SQL> SELECT MAX(SEQUENCE#), THREAD# FROM V$ARCHIVED_LOG GROUP BY THREAD#;

Step 3 Determine the most recent archived redo log file at each destination.

Enter the following query at the primary database to determine which archived redo log file was most recently transmitted to each of the archiving destinations:

SQL> SELECT DESTINATION, STATUS, ARCHIVED_THREAD#, ARCHIVED_SEQ#
2> FROM V$ARCHIVE_DEST_STATUS
3> WHERE STATUS <> 'DEFERRED' AND STATUS <> 'INACTIVE';

The most recently written archived redo log file should be the same for each archive destination listed. If it is not, a status other than VALID might identify an error encountered during the archival operation to that destination.

Step 4 Find out if archived redo log files have been received.

You can issue a query at the primary database to find out if an archived redo log file was not received at a particular site. Each destination has an ID number associated with it. You can query the DEST_ID column of the V$ARCHIVE_DEST fixed view on the primary database to identify each destination's ID number.

Assume the current local destination is 1, and one of the remote standby destination IDs is 2. To identify which log files are missing at the standby destination, issue the following query:

```
SQL> SELECT LOCAL.THREAD#, LOCAL.SEQUENCE# FROM
2> (SELECT THREAD#, SEQUENCE# FROM V$ARCHIVED_LOG WHERE DEST_ID=1)
3> LOCAL WHERE
4> LOCAL.SEQUENCE# NOT IN
5> (SELECT SEQUENCE# FROM V$ARCHIVED_LOG WHERE DEST_ID=2 AND
6> THREAD# = LOCAL.THREAD#);

THREAD# SEQUENCE#
--------- ---------
1 12
1 13
1 14
```