# Monitoring Hydra's Database and Applications


- Introduction
- Monitoring Task Execution in Hydra
  - Monitoring Task Runs
- Monitoring Tablespace
- Monitoring Unrated Phone Calls
- Monitoring the Date of the Last CDR Load
- Monitoring the Date of the Last Sessions Update
- Monitoring the Date of the Last Successful Payment Load
- Monitoring Events Execution
- Monitoring Replication Consistency
- Monitoring the Number of Table Rows
  - SD_GOOD_MOVES
  - EX_DATA_COLLECT
  - EX_CALL_DATA_REC
  - SS_SESSION_LOGS
- Monitoring System Applications
  - Web Applications
    - HOPER
    - HUPO
    - HDD
  - Agents
    - hamd
    - hard
    - hcd
    - hid
    - hpd
  - FreeRADIUS
- Application: A Script for a Database User with Required Permissions
- Creating a user for monitoring


## Introduction

As soon as Hydra is launched into production mode you should set up monitoring for Hydra's database instance.

The following parameters are to be monitored:

- Basic parameters of the billing server (load avg, CPU and RAM usage, I/O of the disk array, the number of processes, free space within the disk array).
- Key figures for the database instance (the number of sessions, current load, SGA status, and so on).
- Scheduled task statuses in Hydra.
- Monitoring the amount of unrated CDRs and the date of the last load.
- The number of rows in most important tables such as EX_CALL_DATA_REC, EX_DATA_COLLECT, EX_TRAFFIC_COLLECT_C, SD_GOOD_MOVES.

To monitor Hydra, you can use any popular system which allows receiving data with the help of CLI scripts. Most popular monitoring solutions among providers are free solutions like Zabbix and Nagios.

To monitor key figures of the Hydra's database instance you can use the scripts (from the Oracle section), created especially for Zabbix.

## Monitoring Task Execution in Hydra

In order to set up monitoring for task execution in Hydra, you can use either one of the custom scripts (1) or **hydra_monitoring.sh hydra_job_monitoring.sh**

**hydra_job_monitoring.sh**

```
#!/bin/sh

source /etc/profile

rval=0

ORA_USER="AIS_NET"
ORA_PASS="mypass"
SQLPLUS_PATH="$ORACLE_HOME/bin/sqlplus"

if [ -n "$3" ]; then
       ORA_SID="$3"
       export ORACLE_SID=$ORA_SID
fi

sql=""

case $1 in

'job_state')
    if [ -n "$2" ]; then
        echo "
        SELECT to_char(N_JOB_STATE_ID, 'FM99999999999999990') FROM SS_V_JOBS WHERE N_JOB_ID=$2;
        " | ${SQLPLUS_PATH} -s ${ORA_USER}/${ORA_PASS}@${ORACLE_SID} |
        awk '{  if ($1 == 2034) print "Running"
            else if($1 == 1034) print "Waiting"
            else if($1 == 3034) print "Starting"
            else if($1 == 4034) print "Locked"
            else if($1 == 5034) print "Deleted"
            else if($1 == 6034) print "Error"
            else if($1 == 7034) print "Cant Start"
        }'
    else
        rval=1
        echo "No JOB_ID" >&2
    fi
    ;;
'job_last_start')
    if [ -n "$2" ]; then
        sql="
        SELECT to_char((sysdate - D_LAST_START) * (86400), 'FM99999999999999990') FROM SS_V_JOBS WHERE
N_JOB_ID=$2;
        "
    else
        rval=1
        echo "No JOB_ID" >&2
    fi
    ;;
*)
    echo "Hydra monitoring tool"
    echo "usage:"
    echo "    $0 job_state <JOB_ID> [SID]  -- Check job status."
    echo "    $0 job_last_start <JOB_ID> [SID]  -- Check job last start date/time."
    rval=1
    exit $rval
    ;;
esac

if [ -n "$sql" ]; then
       echo "$sql" | ${SQLPLUS_PATH} -s ${ORA_USER}/${ORA_PASS}@${ORACLE_SID}
fi
rval=$?

exit $rval
```

There are 7 types of statuses:

| Code | Status | Mark |
|---|---|---|
| 1034 | Pending | Y |
| 2034 | Running | Y |
| 3034 | Waiting to start | Y |
| 4034 | Locked | N |
| 5034 | Deleted | N |
| 6034 | Failed to start | N |
| 7034 | Not started | Y |

N stands for an off-normal status. **Exception:** a manually locked task.

To see the list of tasks and their IDs, go to *Administration -> Tasks - Scheduled tasks*.
We highly recommend monitoring all tasks in the system.

For monitoring new Task Execution in the last 15 minutes, you can use the following SQL request:

```
SELECT COUNT(*)
FROM ss_v_job_seances
WHERE d_start > SYSDATE - 15/24/60
AND   c_reason = 'A'
```

If the result is '0', then Task Execution is doesn't work.

## Monitoring Task Runs

Apart from monitoring task statuses, you should also monitor the statuses of completed task runs. With the help of such monitoring, you will be able to track any off-normal statuses of completed task runs, as well as the execution duration for the tasks. To receive details on the last 10 completed task runs, you should use the following SQL request:

```
WITH JOB_SEANCES AS (
  SELECT N_JOB_STATUS,
         VC_JOB_STATUS,
         D_START,
         D_FINISH,
         ROW_NUMBER() OVER (ORDER BY D_START DESC) N_ROW
  FROM   SS_V_JOB_SEANCES
  WHERE  N_JOB_ID = <num_N_JOB_ID>)
SELECT *
FROM   JOB_SEANCES
WHERE  N_ROW <= 10;
```

where:

- `num_N_JOB_ID` — task ID.

There are 5 possible task execution results ( see `N_JOB_STATUS` in the SQL request above):

| Code | Status | Mark |
|---|---|---|
| -2 | Aborted | N |
| -1 | Running | Y |
| 0 | Success | Y |

| | | |
|---|---|---|
| 1 | Warning | N |
| 2 | Error | N |

Off-standard statuses are marked with N.

The `D_START` and `D_FINISH` fields of the SQL request contain the begin date and the end date of the execution. For running tasks, `D_FINISH is set to` **`NULL`**.

To see the list of tasks and their IDs, go to *Administration -> Tasks - Scheduled tasks*.
We highly recommend monitoring all tasks in the system.

## Monitoring Tablespace

You should monitor the statuses of the database tablespaces. Use this SQL request to receive the required details:

```
WITH TBLSP_TOTAL AS (
  SELECT TABLESPACE_NAME,
         ROUND(SUM(BYTES)/(1024*1024)) ALLOCATED_MB,
         ROUND(SUM(DECODE(MAXBYTES, 0, BYTES, MAXBYTES))/(1024*1024)) MAX_MB
  FROM   DBA_DATA_FILES
  GROUP BY TABLESPACE_NAME
  UNION ALL
  SELECT TABLESPACE_NAME,
         ROUND(SUM(BYTES)/(1024*1024)) ALLOCATED_MB,
         ROUND(SUM(DECODE(MAXBYTES, 0, BYTES, MAXBYTES))/(1024*1024)) MAX_MB
  FROM   DBA_TEMP_FILES
  GROUP BY TABLESPACE_NAME),
TBLSP_FREE AS (
  SELECT TABLESPACE_NAME,
         ROUND(SUM(BYTES)/(1024*1024)) FREE_MB
  FROM   DBA_FREE_SPACE
  GROUP BY TABLESPACE_NAME
  UNION ALL
  SELECT TABLESPACE_NAME,
         ROUND(FREE_SPACE/(1024*1024)) FREE_MB
  FROM   DBA_TEMP_FREE_SPACE),
TBLSP_USED AS (
  SELECT TS.TABLESPACE_NAME,
         TS.ALLOCATED_MB - NVL(FS.FREE_MB, 0)                USED_MB,
         NVL(FS.FREE_MB, 0)                                  FREE_MB,
         TS.ALLOCATED_MB                                     TOTAL_MB,
         TS.MAX_MB                                           TOTAL_MAX_MB,
         TS.MAX_MB - (TS.ALLOCATED_MB - NVL(FS.FREE_MB, 0))  FREE_MAX_MB
  FROM   TBLSP_TOTAL TS,
         TBLSP_FREE  FS
  WHERE  FS.TABLESPACE_NAME(+) = TS.TABLESPACE_NAME)
SELECT TABLESPACE_NAME  "TABLESPACE",
       USED_MB          "Used MB",
       FREE_MB          "Free MB",
       TOTAL_MB         "Total MB",
       TOTAL_MAX_MB     "Total Max MB",
       FREE_MAX_MB      "Free Max MB",
       ROUND(100*(FREE_MAX_MB/TOTAL_MAX_MB))  "Pct. Free"
FROM   TBLSP_USED
ORDER BY TABLESPACE_NAME;
```

As a result, you will get a table similar to the example below:

| # | TABLESPACE | Used MB | Free MB | Total MB | Total Max MB | Free Max MB | Pct. Free |
|---|---|---|---|---|---|---|---|
| 1 | HYDRA | 22903 | 27555 | 50458 | 61492 | 38589 | 63 |
| 2 | HYDRA_INDEX | 40621 | 6781 | 47402 | 94292 | 53671 | 57 |
| 3 | SYSAUX | 1829 | 321 | 2150 | 32768 | 30939 | 94 |
| 4 | SYSTEM | 14413 | 67 | 14480 | 32768 | 18355 | 56 |
| 5 | TOOLS | 1 | 31 | 32 | 32 | 31 | 97 |
| 6 | UNDOTBS1 | 8445 | 29078 | 37523 | 65536 | 57091 | 87 |
| 7 | USERS | 1 | 4 | 5 | 32768 | 32767 | 100 |

Fileds stand for:

- *TABLESPACE* — a name of a tablespace
- *Used MB* — amount of used blocks within the used disk volume space
- *Free MB* — amount of free blocks within the used disk volume
- *Total MB* — a total used disk space volume
- *Total Max MB* — a maximum available disk space that can be occupied by the database. It can be greater than the volume of the available disk space.
- *Free Max MB* — a maximum available disk space that can be used for increased data. It can be greater than the volume of the available disk space.
- *Pct. Free* — a free volume percentage. It is calculated as a ratio of Free Max MB to Total Max MB.

You should pay attention to the *Pct. Free* key figure estimated for the `HYDRA` and `HYDRA_INDEX spaces`. If it drops to really low numbers (less than 20%), the system administrator should be automatically informed about the problem concerning the free space. Otherwise, for example, some important system tasks in the database will be brought to a halt with the following error message:

```
   JB_DATA_COLLECT_PKG.EX_DATA_COLLECT_ACCOUNTING
[ORA-01654: unable to extend index AIS_NET.EX_TRAFFIC_COL_C_FIRM_IDX by 8192
in tablespace HYDRA_INDEX]
```

## Monitoring Unrated Phone Calls

The request below returns a number of unrated phone calls over the last hour:

```
SELECT COUNT(*)
FROM   EX_V_CDR
WHERE  N_CDR_TYPE_ID   = SYS_CONTEXT('CONST', 'CDR_TYPE_PhoneCall')
AND    N_CDR_STATE_ID  = SYS_CONTEXT('CONST', 'CDR_Status_Finished')
AND    D_END          >= SYSDATE - 1/24
AND   (N_SUM_A IS NULL AND N_SUM_B IS NULL)
AND    N_DURATION_SEC != 0;
```

The standard value here is 0, i.e. no unrated phone calls at all.

## Monitoring the Date of the Last CDR Load

The following request is used to view the date and time of the last CDR load into Hydra (this option is convenient for analyzing by a person):

```
SELECT DECODE(MAX(D_LOG_CREATE), NULL, 'Never',
       TO_CHAR(MAX(D_LOG_CREATE), 'DD.MM.YYYY HH24:MI:SS')) VC_LAST_LOAD_DATE
FROM  EX_V_CDR
WHERE N_CDR_TYPE_ID = SYS_CONTEXT('CONST', 'CDR_TYPE_PhoneCall');
```

As a result, the response to the request will contain either the exact date in the following format:

```
LAST_DATE_LOAD
-------------------
29.01.2013 11:05:54
```

or the «*Never*» row, if there is no CDR in the database:

```
LAST_DATE_LOAD
-------------------
Never
```

The following request returns result containing either the date and time of the last CDR load (calculated in seconds since the date of the last load up to the current moment) or «*-1*» in case there is no CDR in the database. The following request is a good option for processing within monitoring systems:

```
SELECT DECODE(MAX(D_LOG_CREATE), NULL, -1, TO_CHAR((SYSDATE-MAX(D_LOG_CREATE))*(60*60*24), 'FM99999999999999990')
) N_LAST_LOAD_SEC
FROM  EX_V_CDR
WHERE N_CDR_TYPE_ID = SYS_CONTEXT('CONST', 'CDR_TYPE_PhoneCall');
```

Triggers for monitoring should be set up depending on either the interval for loading CDR or RADIUS accounting.

## Monitoring the Date of the Last Sessions Update

The following request is used to monitor the date of the last sessions of updating customer data in Hydra (this option is convenient for analysing by a person):

```
SELECT DECODE(MAX(D_END), NULL, 'Never',
       TO_CHAR(MAX(D_END), 'DD.MM.YYYY HH24:MI:SS')) VC_LAST_UPDATE
FROM  EX_V_CDR
WHERE N_CDR_TYPE_ID IN (SYS_CONTEXT('CONST', 'CDR_TYPE_PPP_WithCharging'), SYS_CONTEXT('CONST', 'CDR_TYPE_PPP_WOC
harging'));
```

As a result, the response to the request will contain either the exact date in the following format:

```
LAST_DATE_LOAD
-------------------
26.11.2013 10:01:38
```

or the «*Never*» string, if there are no sessions in the database:

```
LAST_DATE_LOAD
-------------------
Never
```

The following request returns a response containing either the date and time of the last sessions update (calculated in seconds since the date of the last update up to the current moment) or «*-1*» in case there are no sessions in the database. This request is a good option for processing within monitoring systems:

```
SELECT DECODE(MAX(D_END), NULL, -1, TO_CHAR((SYSDATE-MAX(D_END))*(60*60*24), 'FM99999999999999990'))
N_LAST_UPDATE_SEC
FROM  EX_V_CDR
WHERE N_CDR_TYPE_ID IN (SYS_CONTEXT('CONST', 'CDR_TYPE_PPP_WithCharging'), SYS_CONTEXT('CONST', 'CDR_TYPE_PPP_WOC
harging'));
```

Triggers for monitoring should be set up depending on the interval for loading RADIUS accounting.

## Monitoring the Date of the Last Successful Payment Load

The following request is used to monitor the date of the last successful payment load from external payment systems into Hydra (this option is convenient for analyzing by a person):

```
SELECT DECODE(MAX(D_LOAD), NULL, 'Never',
       TO_CHAR(MAX(D_LOAD), 'DD.MM.YYYY HH24:MI:SS')) VC_LAST_LOAD
FROM  EX_V_PAYMENTS
WHERE N_RATING = 100;
```

As a result, the response to the request will contain either the exact date in the following format:

```
LAST_DATE_LOAD
-------------------
25.03.2014 08:09:43
```

or the «*Never*» row, if there are no successful payments from external systems in the database:

```
LAST_DATE_LOAD
-------------------
Never
```

To view the date of the last successful load for a certain payment system you should use the following request:

```
SELECT DECODE(MAX(D_LOAD), NULL, 'Never',
       TO_CHAR(MAX(D_LOAD), 'DD.MM.YYYY HH24:MI:SS')) VC_LAST_LOAD
FROM  EX_V_PAYMENTS
WHERE N_RATING        = 100
AND  N_TO_ACCOUNT_ID = <num_N_TO_ACCOUNT_ID>;
```

where:

- `num_N_TO_ACCOUNT_ID` — stands for the payment system account ID of the provider organization.

The request below is followed by a response containing either the date and time of the last successful payment load (calculated in seconds since the date of the last load up to the current moment), or «*-1*» in case there are no successful payments from external payment systems in the database. This request can be used for processing within monitoring systems:

```
SELECT DECODE(MAX(D_LOAD), NULL, -1, TO_CHAR((SYSDATE-MAX(D_LOAD))*(60*60*24), 'FM99999999999999990'))
N_LAST_LOAD_SEC
FROM  EX_V_PAYMENTS
WHERE N_RATING = 100;
```

A similar request when monitoring by payment system:

```
SELECT DECODE(MAX(D_LOAD), NULL, -1, TO_CHAR((SYSDATE-MAX(D_LOAD))*(60*60*24), 'FM99999999999999990'))
N_LAST_LOAD_SEC
FROM  EX_V_PAYMENTS
WHERE N_RATING        = 100
AND  N_TO_ACCOUNT_ID = <num_N_TO_ACCOUNT_ID>;
```

where:

- `num_N_TO_ACCOUNT_ID` — stands for the payment system account ID of the provider organization.

Triggers for monitoring should be set up depending on how often payments arrive.

## Monitoring Events Execution

The following request is used to monitor the number of errors and warnings appeared in events queue over the last hour:

```
SELECT COUNT(*)
FROM   SS_V_EVENTS_QUEUE
WHERE  N_EVENT_STATE_ID IN (SYS_CONTEXT('CONST', 'EVENT_QUE_STATE_Warning'),
                            SYS_CONTEXT('CONST', 'EVENT_QUE_STATE_Error'))
AND    D_ACK            >= TRUNC(SYSDATE, 'HH');
```

The standard value here is 0, i.e. no errors or warnings at all.

## Monitoring Replication Consistency

In order to check replication, you should use the following request at both databases on behalf of a user with the SYSDBA permission (otherwise it will not be executed on the standby server).

```
SELECT SEQUENCE# FROM V$LOG_HISTORY WHERE RECID = (SELECT MAX(RECID) FROM V$LOG_HISTORY);
```

The result must be the same for both servers. It is permitted to have a difference of not more than 1-2 versions due to replication delay.

To set up monitoring for replication at /etc/sudoers you should add a permission for the hzabbix user to run Oracle script that will access the database on behalf of SYSDBA.

```
zabbix       ALL=(oracle) NOPASSWD:/etc/zabbix-agent/monrep.sh
```

To run the script

```
zabbix@hydra ~ $ sudo -u oracle /etc/zabbix-agent/monrep.sh
```

The monrep.sh script containing a request for obtaining max(sequence#)

```
#!/bin/bash

. /etc/profile
. /etc/environment
export ORAENV_ASK=NO
. oraenv > /dev/null
sql="select max(sequence#) from v\$log_history;"
echo -e $sql | sqlplus -s / as sysdba
```

## Monitoring the Number of Table Rows

You must monitor the number of table rows because a great number of rows in certain tables will inevitably cause performance issues. Such issues are typically located only when the number of table rows has exceeded all tolerable limits due to the incorrect configuration of the system. As a result, it is impossible to delete exceeded data in the short run.

See below an example of a request for checking the number of table rows:

```
SQL> SELECT COUNT(*)
FROM SD_GOOD_MOVES;
```

The table of threshold values

| Table | Use | Threshould |
|---|---|---|
| EX_CALL_DATA_REC | CDR and PPP-sessions | 15M |
| EX_DATA_COLLECT | PPP sessions traffic | average number of completed PPP sessions per month multiplied by the number of traffic classes<br><br>*For example, for 500K sessions per month and 4 traffic classes involved in collecting statistics (local traffic both inbound and outbound, Internet traffic both inbound and outbound), the threshold is calculated as 500K*4= 2M* |
| EX_TRAFFIC_COLLECT_C | Unaccounted traffic | 2M |
| SD_GOOD_MOVES | Charge logs and Invoices contents | 15M |
| SS_SESSION_LOGS | System session logs | 5M |

To define the number of completed PPP sessions per month you can use the following request:

```
SELECT COUNT(*)
FROM   EX_V_CDR
WHERE  N_SERVICE_ID = 40223001 -- a service ID here
AND    D_END BETWEEN TO_DATE('01.05.2013 00:00:00', 'DD.MM.YYYY HH24:MI:SS') AND TO_DATE('31.05.2013 23:59:59',
'DD.MM.YYYY HH24:MI:SS') -- for the period
AND    N_CDR_STATE_ID IN (SYS_CONTEXT('CONST', 'CDR_Status_Finished'),
                          SYS_CONTEXT('CONST', 'CDR_Status_FinForced'));
```

## SD_GOOD_MOVES

This table contains charge log and invoice lines.
When the threshold value is exceeded you need to set up data aggregation and archiving in the system. See the Hydra Billing Admin Guide, *Using the System->Administration->Tasks->Standard tasks-> Archiving charge logs*

## EX_DATA_COLLECT

This table contains statistics of PPP sessions traffic. When the threshold value is exceeded you should go to the settings for the Deleting old CDRs tasks and decrease the value in the EX_DATA_COLLECT records aging period parameter.

## EX_CALL_DATA_REC

This table contains data on CDRs and PPP sessions.
When the threshold value is exceeded you must:

1. Use the following script to define which type of data has the majority

```
SQL> SELECT SI_REF_PKG_S.GET_NAME_BY_ID(N_CDR_TYPE_ID) VC_CDR_NAME,
       COUNT(*)
FROM   EX_CALL_DATA_REC
GROUP BY N_CDR_TYPE_ID;

VC_CDR_NAME              COUNT(*)
----------------------------------------------------------------------
Phone call       107552
PPP session (unrated)  3774012
```

2. In case of a majority of CDRs (phone calls), you should contact technical support for exporting CDRs in a file.
3. In case of a majority of PPP sessions, you should decrease the value of the *Timeout for deleting old CDRs* parameter in network services which PPP sessions are created for.

### SS_SESSION_LOGS

This table contains data on system session logs that are created with the help of the `MAIN.INIT` procedure. When the threshold value is exceeded you should contact technical support for deleting old session records.

# Monitoring System Applications

## Web Applications

### HOPER

When starting a process, a PID file is created.

- For a version 3.3 and higher, its location is specified in the config file, typically at /var/run/hydra/hoper/unicorn.pid.
- For a version below 3.3, its location cannot be changed. It is located at shared/unicorn.pid of the application installation root.

The following coding returns **0** if the process exists, or **1**, if either it does not exist or the PID-file of the process cannot be found (by the **root** user):

```
root@sever:~# PIDFILE="/var/run/hydra/hoper/unicorn.pid" ; if [ -f $PIDFILE ] ; then kill -0 `cat $PIDFILE` >
/dev/null 2>&1 ; echo $? ; else echo "1" ; fi
```

### HUPO

When starting a process, a PID file is created.

- For a version 3.3 and higher, its location is specified in the config file, typically at /var/run/hydra/hupo/unicorn.pid .
- For a version below 3.3, its location cannot be changed. It is located at shared/pids/unicorn.pid of the application installation root.

The following coding returns **0** if the process exists, or **1**, if either it does not exist or the PID-file of the process cannot be found (by the **root** user):

```
root@sever:~# PIDFILE="/var/run/hydra/hupo/unicorn.pid" ; if [ -f $PIDFILE ] ; then kill -0 `cat $PIDFILE` >
/dev/null 2>&1 ; echo $? ; else echo "1" ; fi
```

### HDD

When starting, a hdd_default and a PID file are created.

- For a version 3.3 and higher, its location is specified in the config file, typically at /var/run/hydra/hdd/hdd_default.pid .
- For a version below 3.3, its location cannot be changed. It is located at tmp/ of the application installation root.

The following coding returns **0** if the process exists, or **1**, if either it does not exist or the PID-file of the hdd_default process cannot be found (by the **root** us er):

```
root@sever:~# PIDFILE="/var/run/hydra/hdd/hdd_default.pid" ; if [ -f $PIDFILE ] ; then kill -0 `cat $PIDFILE` >
/dev/null 2>&1 ; echo $? ; else echo "1" ; fi
```

## Agents

### hamd

When starting a process, a PID file is created, with its location specified in the config file (/etc/hamd/hamd.conf). Typically, a PID file is located along the /var/run/hydra/hamd.pid path.

The following coding returns **0** if the process exists, or **1**, if either it does not exist or the PID-file of the process cannot be found (by the **root** user):

```
root@server:~# PIDFILE="/var/run/hydra/hamd.pid" ; if [ -f $PIDFILE ] ; then kill -0 `cat $PIDFILE` > /dev/null
2>&1 ; echo $? ; else echo "1" ; fi
```

We also recommend checking for the process at the port. The following coding returns **0** if the process can listen to the required port, otherwise, the value is **1**:

```
root@server:~# PORT=8889; lsof -i :$PORT -n > /dev/null ; echo $?
```

### hard

When starting a process, a PID file is created, with its location specified in the config file (/etc/hard/hard.conf). Typically, a PID file is located along the /var /run/hydra/hard.pid path.
The following coding returns **0** if the process exists, or **1**, if either it does not exist or the PID-file of the process cannot be found (by the **root** user):

```
root@sever:~# PIDFILE="/var/run/hydra/hard.pid" ; if [ -f $PIDFILE ] ; then kill -0 `cat $PIDFILE` > /dev/null
2>&1 ; echo $? ; else echo "1" ; fi
```

We also recommend checking for the process at the port. The following coding returns **0** if the process can listen to the required port, otherwise, the value is **1**:

```
root@sever:~# PORT=11080; lsof -i :$PORT -n > /dev/null ; echo $?
```

### hcd

When starting a process, a PID file is created, with its location specified in the config file (/etc/hcd/hcd.conf). Typically, a PID file is located along the /var /run/hydra/hcd.pid path.
The following coding returns **0** if the process exists, or **1**, if either it does not exist or the PID-file of the process cannot be found (by the **root** user):

```
root@sever:~# PIDFILE="/var/run/hydra/hcd.pid" ; if [ -f $PIDFILE ] ; then kill -0 `cat $PIDFILE` > /dev/null
2>&1 ; echo $? ; else echo "1" ; fi
```

We also recommend checking for the process at the port. The following coding returns **0** if the process can listen to the required port, otherwise, the value is **1**:

```
root@sever:~# PORT=8888; lsof -i :$PORT -n > /dev/null ; echo $?
```

### hid

When starting a process, a PID file is created, with its location specified in the config file (/etc/hid/hid.conf). Typically, a PID file is located along the /var/run /hydra/hid.pid path.

The following coding returns **0** if the process exists, or **1**, if either it does not exist or the PID-file of the process cannot be found (by the **root** user):

```
root@sever:~# PIDFILE="/var/run/hydra/hid.pid" ; if [ -f $PIDFILE ] ; then kill -0 `cat $PIDFILE` > /dev/null
2>&1 ; echo $? ; else echo "1" ; fi
```

We also recommend checking for the process at the port. The following coding returns **0** if the process can listen to the required port, otherwise, the value is **1**:

```
root@sever:~# PORT=10080; lsof -i :$PORT -n > /dev/null ; echo $?
```

### hpd

When starting a process, a PID file is created, with its location specified in the config file (/etc/hpd/hpd.conf). Typically, a PID file is located along the /var /run/hydra/hpd.pid path.
The following coding returns **0** if the process exists, or **1**, if either it does not exist or the PID-file of the process cannot be found (by the **root** user):

```
root@sever:~# PIDFILE="/var/run/hydra/hpd.pid" ; if [ -f $PIDFILE ] ; then kill -0 `cat $PIDFILE` > /dev/null
2>&1 ; echo $? ; else echo "1" ; fi
```

We also recommend checking for the process at the port. The following coding returns **0** if the process can listen to the required port, otherwise, the value is **1:**

```
root@sever:~# PORT=9080; lsof -i :$PORT -n > /dev/null ; echo $?
```

## FreeRADIUS

When starting a process, a PID file is created, with its location specified in the config file (typically, /etc/freeradius/radiusd.conf). Typically, a PID file is located along the /var/run/radiusd/radiusd.pid path.
The following coding returns **0** if the process exists, or **1**, if either it does not exist or the PID-file of the process cannot be found (by the **root** user):

```
root@sever:~# PIDFILE="/var/run/radiusd/radiusd.pid" ; if [ -f $PIDFILE ] ; then kill -0 `cat $PIDFILE` > /dev
/null 2>&1 ; echo $? ; else echo "1" ; fi
```

We also recommend checking for the process at the port (most often, the UDP port 1812 is used for authorizing). The following coding returns **0** if the process can listen to the required port, otherwise, the value is **1**:

```
root@sever:~# PORT=1812; lsof -i :$PORT -n > /dev/null ; echo $?
```

# Application: A Script for a Database User with Required Permissions

The following script contains commands for creating a user with the permissions required for monitoring the system and Oracle, you should run it on behalf of the **SYS** user:

## Creating a user for monitoring

```
CREATE USER &&username PROFILE DEFAULT IDENTIFIED BY &&password
DEFAULT TABLESPACE USERS TEMPORARY TABLESPACE TEMP
ACCOUNT UNLOCK;
/
GRANT SELECT ON V_$LOG_HISTORY TO &&username;
GRANT SELECT ON V_$PARAMETER TO &&username;

GRANT CONNECT TO &&username;
GRANT RESOURCE TO &&username;

GRANT SELECT ON SS_V_JOBS TO &&username;
GRANT SELECT ON SI_V_USERS TO &&username;
GRANT SELECT ON SS_V_JOB_SEANCES TO &&username;
GRANT SELECT ON SS_V_MANAGER_JOBS TO &&username;

GRANT EXECUTE ON SI_SUBJECTS_PKG_S TO &&username;
GRANT EXECUTE ON SI_OBJECTS_PKG_S TO &&username;
GRANT EXECUTE ON SI_REF_PKG_S TO &&username;


-- count.q*
GRANT SELECT ON SS_V_EVENTS_QUEUE to &&username;

-- count.dbc
GRANT SELECT ON V_$DATABASE_BLOCK_CORRUPTION TO &&username;
-- count.uretenop
GRANT SELECT ON V_$UNDOSTAT TO &&username;
-- count.cdr; count.lastcdr
GRANT SELECT ON EX_V_CDR TO &&username;
-- count.gm
GRANT SELECT ON SD_GOOD_MOVES TO &&username;
-- count.ecrd
GRANT SELECT ON EX_CALL_DATA_REC TO &&username;
-- count.edc
GRANT SELECT ON EX_DATA_COLLECT TO &&username;
-- count.etcc
GRANT SELECT ON EX_TRAFFIC_COLLECT_C TO &&username;
-- count.active
GRANT SELECT ON SI_SUBJ_GOODS TO &&username;
-- tblspace.discovery
GRANT SELECT ON DBA_SEGMENTS TO &&username;
-- tblspace.pcf
GRANT SELECT ON DBA_DATA_FILES TO &&username;
GRANT SELECT ON DBA_FREE_SPACE TO &&username;
GRANT SELECT ON DBA_TEMP_FILES TO &&username;
GRANT SELECT ON DBA_TEMP_FREE_SPACE TO &&username;

GRANT SELECT ON V_$TEMP_EXTENT_POOL TO &&username;
GRANT SELECT ON DBA_UNDO_EXTENTS TO &&username;

-- checkactive
GRANT SELECT ON V_$INSTANCE TO &&username;
-- rcachehit
GRANT SELECT ON V_$SYSSTAT TO &&username;
-- activeusercount
GRANT SELECT ON V_$SESSION TO &&username;
-- dbsize
GRANT SELECT ON DBA_FREE_SPACE TO &&username;
GRANT SELECT ON DBA_TABLESPACES TO &&username;
-- lastarclog
GRANT SELECT ON V_$LOG TO &&username;
-- freebufwaits
GRANT SELECT ON V_$SYSTEM_EVENT TO &&username;
GRANT SELECT ON V_$EVENT_NAME TO &&username;
GRANT SELECT ON EX_V_PAYMENTS TO &&username;
GRANT SELECT ON SS_SESSION_LOGS TO &&username;
/
QUIT;
```