

Introduction to Simple Oracle Auditing

Pete Finnigan 2003-04-29

Introduction

This article will introduce the reader to the basics of auditing an Oracle database. Oracle's RDBMS is a functionally rich product and there are a number of auditing alternatives available to the reader. Because auditing Oracle is such a huge subject, doing all of it justice would take an entire book, so this paper will cover the basics of why, when and how to conduct an audit. It will also use a couple of good example cases to illustrate how useful Oracle audit can be to an organization.

All of the sample SQL listed in this paper can be downloaded from the author's Web site at <http://www.petefinnigan.com/papers/audit.sql>

The Issues

There are a number of basic issues that should be considered when contemplating using Oracle's auditing features. These are as follows:

- **Why is audit needed in Oracle?**

Is this a strange question? Well, lots of companies don't actually use the internal audit features of Oracle. Or, when they do use them, they are so overwhelmed with choice, they turn on everything for good measure, then realise there is far too much output to read and digest so they quickly turn it all off again. It is quite common to use firewalls, intrusion detection systems (IDS) and other security tools to determine if the network or operating system is being misused or abused. So why not audit what users are doing to the "crown jewels" of an organization, the data. Oracle audit can help detect unauthorized access and internal abuse of the data held in the database.

- **When should Oracle users be audited?**

A simple basic set of audit actions should be active all the time. The ideal minimum is to capture user access, use of system privileges and changes to the database schema structure. This basic set will not show attempted access to specific data that shouldn't be accessed; however, it will give a reasonably simple overview of "incorrect" access and use of privileges. If an employee is

suspected of inappropriate actions or if an attack has been suspected then more detailed audit can be turned on for specific tables. From a data management point of view, auditing data changes for all tables in the database is not really practical and could also affect performance. Monitoring data change access on critical tables (such as salaries in a HR database) should be considered.

- **How can Oracle users be audited?**

The standard audit commands allow all system privileges to be audited along with access at the object level to any table or view on the database for select, delete, insert or update. Audit can be run for either successful or unsuccessful attempts or both. It can be for each individual user or for all users and it can also be done at the session level or access level. At action level a single record is created per action and at session level one record is created for all audit actions per session.

- **What are the performance and complexity issues?**

Audit is generally perceived to be complex and slow. The reason for this is usually ignorance. If many or all options are turned on, then the resultant audit trail produced can be large and difficult to interpret and manage. Furthermore, if audit is used on all tables and views in the database, then this can have an effect on performance. Every time an action performed is auditable a record is written to the database; clearly the more audit is used, the more records will be written to the system tablespace purely for audit. In some cases double the amount of access to the database can be performed: the original write and the audit record being written.

The watchword here is simplicity and caution. Use only the audit that is needed to give an overall view of what is happening and for detailed monitoring of critical data and objects. The simpler the audit trail set-up, the more likely it is that the data will be analyzed and be of some use. It is important to define what actions or abuses are being checked for so that simple reports can be written to filter the audit trail for these actions. A default installation of Oracle has audit turned off by default and Oracle does not come with any standard default audit settings or reports to analyse any audit trail produced. These reasons, and the fact that there are many options available are, in my opinion, why audit is perceived to be complex.

The standard audit commands do not allow audit to be performed at row level. It is also not possible to audit the actions of privileged users such as SYS and "as sysdba" until Oracle 9iR2.

Oracles Audit Facilities

The task of auditing an Oracle database does not have to be limited only to the audit commands; other techniques can be employed as well. Here are some of the main methods that can be used to audit an Oracle database:

- **Oracle audit**

This is really the subject of this paper. All privileges that can be granted to a user or role within the database can be audited. This includes read, write and delete access on objects at the table level. For more detailed audit, the database triggers need to be employed.

- **System triggers**

These were introduced with Oracle 8 and allow the writing of database triggers that fire when system events take place. These include start- up and shutdown of the database, log-on and log-off attempts, and creation, altering and dropping of schema objects. With the aid of autonomous transactions, these allow a log to be written for the above system events.

- **Update, delete, and insert triggers**

This is the second line of defence in trying to understand users' actions at a more detailed row level. Database triggers need to be written to capture changes at the column and row level. It is possible to write complete rows of data before and after the change being made to a log table in the database. The use of this type of logging is very resource intensive, as many extra records are written and stored. The one failing with this method is that read access cannot be captured with normal database triggers.

- **Fine-grained audit**

Fine-grained audit solves the problem of capturing read access. This feature is also based on internal triggers that fire when any piece of SQL is parsed. This is very efficient, as the SQL is parsed once for audit and execution. The feature uses predicates that are defined and tested each time the relevant object is accessed. Fine-grained audit is managed by a PL/SQL package called DBMS_FGA. A PL/SQL procedure is executed every time a "match" is made with the predicate. This method allows the audit to be performed down to the row and column level and

to also for read statements. Readers should be forewarned that use of this feature requires programming skills.

• **System logs**

Oracle generates many log files and many of them can provide useful information to assist in auditing the database. One good example is the alert log used by the database to record start-up and shutdown as well as any structural changes such as adding a datafile to the database.

This paper is going to explore only the standard built-in audit commands. The other options will be left for future articles.

Some Examples

Because of the myriad of possibilities, auditing an Oracle database can be a daunting task. In order to try and simplify the discussion of what can be done, we will discuss a couple of simple examples that we will explore and work through.

• **Auditing database access**

This is a fundamental check to find out who accesses the database, from where and when. Log-on failures can be captured as well as log-ons at strange (anomalous) times of the day.

• **Auditing changes to the database structure**

In a production database, no user should ever change the schema structure. DBAs should make changes for upgrades at specific times; any other changes should be regarded as suspicious. Watching for structural changes can turn up indicators of incorrect use of the database.

A third simple example that could have been employed here is to audit any use of system privileges. However, this example is left to the reader to explore.

The final group of audit commands that can be employed is to audit any data changes to objects themselves. Unfortunately, as the requirements are very application and installation specific, this is beyond the scope of this paper.

Audit within Oracle is broken into three areas: statement auditing such as CREATE TABLE or

CREATE SESSION, privilege auditing such as ALTER USER, and object level auditing such as SELECT TABLE.

Basic Configuration

The audit trail can be either written to the database or to the operating system. Writing the audit trail to the operating system is, in some senses, more secure but the implementation is not available on all platforms and is platform specific. In this article we will concentrate on using the database to store the audit trail.

Audit is turned on for writing to the database by adding the following line to the init.ora file. A symbolic link to it can usually be found in \$ORACLE_HOME/dbs

```
audit_trail = db
```

The database now needs to be restarted. A simple check will show that audit is indeed now turned on.

```
SQL> select name,value from v$parameter
      2  where name like 'audit%';
```

NAME	VALUE
-----	-----
audit_trail	DB
audit_file_dest	?/rdbms/audit

```
SQL>
```

No audit actions are captured yet until audit actions are defined; that is, except for privileged access to the database, starting and stopping of the database, and structural changes such as adding a datafile. These are logged to operating system files in \$ORACLE_HOME/rdbms/audit unless audit_file_dest is redefined in the *init.ora* file. On Windows these *events* appear in the Event Viewer. To check if any privilege or statement audit actions are enabled, do the following:

```
SQL> select * from dba_stmt_audit_opts
      2  union
      3  select * from dba_priv_audit_opts;
```

```
no rows selected
```

```
SQL>
```

To find out what objects are being audited, query the view `dba_obj_audit_opts`.

The Worked Examples

Let us now work through our two example cases and see what can be learned. First, turn on audit for the access attempts to the database:

```
SQL> audit create session;
```

```
Audit succeeded.
```

```
SQL>
```

The above command will capture access by all users by access and whether successful or unsuccessful. The default for this command is by access.

Note: The format of all audit commands from the Oracle documentation is as follows:

```
audit {statement_option|privilege_option} [by user] [by  
{session|access}] [ whenever {successful|unsuccessful}]
```

Only the statement_option or privilege_option part is mandatory. The other clauses are optional and enabling them allows audit be more specific.

For a user to define audit statements, the privilege "AUDIT SYSTEM" needs to have been granted first. The users that have this privilege can be checked as follows:

```
SQL> select *  
  2  from dba_sys_privs  
  3  where privilege like '%AUDIT%';
```

```
GRANTEE
```

```
PRIVILEGE
```

```
ADM
```

```
-----  
CTXSYS          AUDIT ANY          NO  
CTXSYS          AUDIT SYSTEM      NO  
DBA             AUDIT ANY          YES  
DBA             AUDIT SYSTEM      YES  
IMP_FULL_DATABASE  AUDIT ANY          NO  
MDSYS           AUDIT ANY          YES  
MDSYS           AUDIT SYSTEM      YES  
WKSYS           AUDIT ANY          NO  
WKSYS           AUDIT SYSTEM      NO
```

9 rows selected.

SQL>

The above results are for a 9i database, the default users MDSYS, CTXSYS and WKSYS would likely be good targets for attackers, as any audit actions could be turned off as one of these users to hide any actions undertaken.

Now that audit will capture all access attempts, we need to wait for some users to log in to do work. So while they do that, let's set up the audit to capture alterations to the schema. For the sake of brevity, in this example, not all schema object changes will be captured. Changes to *tables, indexes, clusters, views, sequences, procedures, triggers, libraries* and many more can be captured. In this example, audit will be enabled on an example set. Turning on the audit can be performed as a two-stage process, generate the audit commands and then run them as follows:

```
set head off  
set feed off  
set pages 0  
spool aud.lis  
select 'audit '||name||';'  
from system_privilege_map  
where (name like 'CREATE%TABLE%'  
or name like 'CREATE%INDEX%'
```

```
or name like 'CREATE%CLUSTER%'
or name like 'CREATE%SEQUENCE%'
or name like 'CREATE%PROCEDURE%'
or name like 'CREATE%TRIGGER%'
or name like 'CREATE%LIBRARY%')
union
select 'audit '||name||';'
from system_privilege_map
where (name like 'ALTER%TABLE%'
or name like 'ALTER%INDEX%'
or name like 'ALTER%CLUSTER%'
or name like 'ALTER%SEQUENCE%'
or name like 'ALTER%PROCEDURE%'
or name like 'ALTER%TRIGGER%'
or name like 'ALTER%LIBRARY%')
union
select 'audit '||name||';'
from system_privilege_map
where (name like 'DROP%TABLE%'
or name like 'DROP%INDEX%'
or name like 'DROP%CLUSTER%'
or name like 'DROP%SEQUENCE%'
or name like 'DROP%PROCEDURE%'
or name like 'DROP%TRIGGER%'
or name like 'DROP%LIBRARY%')
union
select 'audit '||name||';'
from system_privilege_map
where (name like 'EXECUTE%INDEX%'
or name like 'EXECUTE%PROCEDURE%'
or name like 'EXECUTE%LIBRARY%')
/
spool off
@@aud.lis
```

This will generate a set of audit commands that can be captured to a spool file, which is then

run to enable the audit commands.

Another solution would be to audit the actual permissions granted to users by generating the audit commands from the database view *dba_sys_privs*. While this may seem to be a better solution and potentially involve less audit commands, it would not allow for the case when new permissions are granted to users. In this case, audit would also need to be enabled at the time the privileges are granted.

Now that all of the sample audit is now enabled, the settings can be viewed with this SQL:

```

1  select audit_option,success,failure
2  from dba_stmt_audit_opts
3  union
4  select privilege,success,failure
5* from dba_priv_audit_opts
SQL> /

```

AUDIT_OPTION	SUCCESS	FAILURE
ALTER ANY CLUSTER	BY ACCESS	BY ACCESS
ALTER ANY INDEX	BY ACCESS	BY ACCESS
ALTER ANY INDEXTYPE	BY ACCESS	BY ACCESS
ALTER ANY LIBRARY	BY ACCESS	BY ACCESS
?		
EXECUTE ANY LIBRARY	BY SESSION	BY SESSION
EXECUTE ANY PROCEDURE	BY SESSION	BY SESSION

38 rows selected.

SQL>

Every time a user attempts anything in the database where audit is enabled the Oracle kernel checks to see if an audit record should be created or updated (in the case of a session record) and generates the record in a table owned by the SYS user called AUD\$. This table is, by default, located in the SYSTEM tablespace. This in itself can cause problems with potential denial of service attacks. If the SYSTEM tablespace fills up, the database will hang.

The AUD\$ table is rare, as it is the only SYS owned table from which Oracle allows records to be deleted. If the audit trail is turned on and written to the database, then the numbers of records in this table need to be monitored carefully to ensure it doesn't grow too fast and fill the system tablespace. A purging strategy needs to be adopted to keep the size of the table in check and, if needed, to archive off audit trail records for future reference. One tactic could be to copy the records to summary tables that allow specific checks for abuse to be performed offline. These summary tables can be in a separate database for added security. Once copied, sys.aud\$ can be truncated.

SYS.AUD\$ can be moved to a different tablespace other than SYSTEM but check with Oracle support first, as this action is no longer supported.

Only users who have been granted specific access to SYS.AUD\$ can access the table to read, alter or delete from it. This is usually just the user SYS or any user who has had permissions. There are two specific roles that allow access to SYS.AUD\$ for select and delete, these are DELETE_CATALOG_ROLE and SELECT_CATALOG_ROLE. These roles should not be granted to general users.

Back to the examples, our users have been logging on and working throughout the day and created some audit records. These audit records can be viewed in a number of ways:

- By selecting from SYS.AUD\$ - This is the raw audit trail
- By selecting from dba_audit_trail - This is a DBA view showing the raw audit trail.
- By selecting from dba_audit_session - This view shows just log-on and log-off actions.

A simple piece of SQL can show details of the connection attempts:

```
SQL> get check_create_session
1  --
2  -- check_create_session.sql
3  --
4  col username for a15
5  col terminal for a6
6  col timestamp for a15
7  col logoff_time for a15
8  col action_name for a8
```

```

 9  col returncode for 9999
10  select      username,
11             terminal,
12             action_name,
13             to_char(timestamp, 'DDMMYYYY:HHMISS') timestamp,
14             to_char(logoff_time, 'DDMMYYYY:HHMISS') logoff_time,
15             returncode
16* from        dba_audit_session
SQL> /

```

USERNAME	TERMIN	ACTION_N	TIMESTAMP	LOGOFF_TIME	RETURNCODE
SYS	pts/1	LOGOFF	09042003:051046	09042003:051641	0
ZULIA	pts/1	LOGON	09042003:051641		1017
SYS	pts/1	LOGOFF	09042003:051649	09042003:053032	0
SYS	pts/2	LOGOFF	09042003:052622	09042003:053408	0
ZULIA	pts/1	LOGON	09042003:053032		1017

There are a number of simple abuses that can be checked for in the area of user access to the database. As examples for this paper we will look at the following:

• Failed log-on attempts

This can indicate fat fingers or attackers' attempts to gain unauthorized access the database. The following SQL highlights this:

```

SQL> select count(*),username,terminal,to_char(timestamp, 'DD-MON-YYYY')
 2  from dba_audit_session
 3  where returncode<>0
 4  group by username,terminal,to_char(timestamp, 'DD-MON-YYYY');

```

COUNT(*)	USERNAME	TERMIN	TO_CHAR(TIM
1	BILL	pts/3	09-APR-2003
3	FRED	pts/3	09-APR-2003
4	ZULIA	pts/1	09-APR-2003

SQL>

This shows two possible abuses, the first is the user Zulia attempting to log on and failing four times on the same day. This could be a forgotten password or it could be someone trying to guess his or her password. A change to the SQL as follows gives a bit more detail:

```
SQL> select count(*),username,terminal,to_char(timestamp,'DD-MON-YYYY'),returncode
  2  from dba_audit_session
  3  group by username,terminal,to_char(timestamp,'DD-MON-YYYY'),returncode;
```

COUNT(*)	USERNAME	TERMIN	TO_CHAR(TIM	RETURNCODE
1	BILL	pts/3	09-APR-2003	1017
1	EMIL	pts/1	09-APR-2003	0
1	EMIL	pts/2	09-APR-2003	0
1	EMIL	pts/3	09-APR-2003	0
1	EMIL	pts/4	09-APR-2003	0
3	FRED	pts/3	09-APR-2003	1017
3	SYS	pts/1	09-APR-2003	0
1	SYS	pts/2	09-APR-2003	0
1	SYSTEM	pts/5	09-APR-2003	0
4	ZULIA	pts/1	09-APR-2003	1017
1	ZULIA	pts/1	09-APR-2003	0

11 rows selected.

SQL>

This reveals that the user successfully logged on on the same terminal on the same day. A number of failed log-ons should be agreed as part of these checks and the above SQL run every day. Those users with failure numbers above the threshold should be investigated.

. Attempts to access the database with non-existent users

One interesting extension to the above SQL is to find attempts to log in where the user doesn't exist. An audit record is still created in this case. The following SQL illustrates:

```
SQL> select username,terminal,to_char(timestamp,'DD-MON-YYYY HH24:MI:SS')
 2  from dba_audit_session
 3  where returncode<>0
 4  and not exists (select 'x'
 5     from dba_users
 6     where dba_users.username=dba_audit_session.username)
SQL> /
```

USERNAME	TERMIN	TO_CHAR(TIMESTAMP, 'D
FRED	pts/3	09-APR-2003 17:31:47
FRED	pts/3	09-APR-2003 17:32:02
FRED	pts/3	09-APR-2003 17:32:15
BILL	pts/3	09-APR-2003 17:33:01

SQL>

This is probably abuse. All attempts to log on with a user that doesn't exist should be checked each day and investigated.

• Attempts to access the database at unusual hours

Checks should be made for any attempts to access the database outside of working hours. These accesses could be genuine overtime work or maintenance but they could just as easily be unauthorized access attempts and should be checked as follows:

```
SQL> select      username,
 2     terminal,
 3     action_name,
 4     returncode,
 5     to_char(timestamp,'DD-MON-YYYY HH24:MI:SS'),
 6     to_char(logoff_time,'DD-MON-YYYY HH24:MI:SS')
 7  from dba_audit_session
```

```

8  where to_date(to_char(timestamp,'HH24:MI:SS'),'HH24:MI:SS') <
to_date('08:00:00','HH24:MI:SS')
9  or to_date(to_char(timestamp,'HH24:MI:SS'),'HH24:MI:SS') >
to_date('19:30:00','HH24:MI:SS')
SQL> /

```

```

USERNAME      TERMIN ACTION_N RETURNCODE TO_CHAR(TIMESTAMP,'D TO_CHAR(LOGOFF_TIME,
-----
SYS            pts/1   LOGOFF          0 09-APR-2003 20:10:46 09-APR-2003 20:16:41
SYSTEM        pts/5   LOGOFF          0 09-APR-2003 21:49:20 09-APR-2003 21:49:50
ZULIA         pts/5   LOGON           0 09-APR-2003 21:49:50
EMIL          APOLLO  LOGON           0 09-APR-2003 22:49:12

```

```
SQL>
```

The above SQL shows any connections before 8:00 AM and after 7:30 PM. Any connections, particularly those made by privileged users such as SYS and SYSTEM, should be investigated. Particular attention can be made to the location from which the access was made. For instance, if privileged access is made from machines that are not in the administrator department, the administrator needs to find out why.

- **Check for users sharing database accounts**

The following SQL looks for users who are potentially sharing database accounts:

```

SQL> select count(distinct(terminal)),username
2  from dba_audit_session
3  having count(distinct(terminal))>1
4  group by username
SQL> /

```

```

COUNT(DISTINCT(TERMINAL)) USERNAME
-----
4 EMIL
3 SYS
3 ZULIA

```

```
SQL>
```

This shows that three users have accessed their accounts from more than one location. A further check could be to add a time component to see if they are accessed simultaneously and also to restrict the check per day. The above SQL gives some idea of the potential without complicating it too much. Again, these accounts and users should be investigated.

- **Multiple access attempts for different users from the same terminal**

The final example checks to find where multiple database accounts have been used from the same terminal. The SQL is again simple and could be extended to group by day and also to print out the users per terminal. This is a simple test to illustrate the abuse idea:

```
SQL> select count(distinct(username)),terminal
  2  from dba_audit_session
  3  having count(distinct(username))>1
  4  group by terminal
SQL> /
```

```
COUNT(DISTINCT(USERNAME)) TERMINAL
-----
3 pts/1
2 pts/2
3 pts/3
3 pts/5
```

```
SQL>
```

This could indicate someone trying to gain access by trying many accounts and passwords, or it could indicate legitimate users sharing accounts for certain aspects of their work. In either case, the admin should investigate further.

There are, of course, many other scenarios that could indicate possible abuses. Checking for those is as simple as the cases depicted above. It will be left to the reader to experiment. Let me know what you find useful.

The second example case that audit actions were set for is to detect changes made to the

database schema. This could include new objects being added or attempts to change existing objects within the database.

A simple piece of SQL will show any audit trail items that relate to objects being created or changed as follows:

```
col username for a8
col priv_used for a16
col obj_name for a22
col timestamp for a17
col returncode for 9999
select  username,
        priv_used,
        obj_name,
        to_char(timestamp, 'DD-MON-YYYY HH24:MI') timestamp,
        returncode
from dba_audit_trail
where priv_used is not null
and priv_used<>'CREATE SESSION'
/
SQL> @check_obj.sql
```

ZULIA	CREATE TABLE	STEAL_SALARY	09-APR-2003 20:07	0
PETE	CREATE PROCEDURE	HACK	09-APR-2003 20:42	0

This simple example shows that the user ZULIA has created a table and the user PETE has been writing PL/SQL procedures. Any changes such as this that are found should be investigated in a production database. Many more specific abuses can be checked for in relation to object and schema changes but, in general, no user should be able to alter the database schema in a production database. As a result, the check can remain pretty simple.

Protecting the Database Against These Abuses

The two examples given are just two of many possible scenarios that could be detected using Oracle's auditing facilities. Turning on and managing audit is one of the first steps to securing

the database. Using audit should be part of an overall organization security plan and policy that includes Oracle. The database should be audited regularly for misconfiguration or known vulnerabilities that could allow security breaches to take place.

Because of its complex nature and vast number of different ways it can be used and configured, the best approach to securing Oracle will always be to follow the principle of least privilege. Once the database is part of the overall security plan and is configured correctly and checked regularly, then auditing it should be considered an important part of the strategy.

In general, do not grant any privileges to general users in a production database, remove most of the PUBLIC privileges and delete or lock and change the passwords of any default accounts. Ensure that users obey password policies and that the password management features of Oracle are employed.

It is important that the audit actions are planned from a performance and usability point of view and that the audit trail is managed. It is also important that the audit trail data is understood in terms of detecting abuse.

The author's recent book by the SANS Institute "Oracle security step- by-step - A survival guide for Oracle security" gives excellent guidelines on how to configure Oracle securely.

Conclusions

Oracle's auditing features are very powerful and sometimes seem very complex. As we saw in the introduction, there is more than one option available for auditing an Oracle database. It is possible to audit almost everything in the Oracle RDBMS with the standard features but not at the row level. If a high-level audit is needed, use the standard features to get a view of overall activity and then home in on the area of concern in more detail.

Because it is possible to audit almost any type of action in an Oracle database using the standard audit features, the reader should experiment with the most useful audit actions for their organization. Keep it simple and do not try to use everything. Above all, predetermine what data will be generated in the audit trail and the abuses that can be checked for. Write reports to check the audit trail and purge it regularly. Finally, monitor the reports each day and take the appropriate action.

For more detailed auditing, use database triggers and fine grained auditing. Keep in mind that

both of these methods need programming skills to implement and report on, so they should be considered carefully. A lot of useful information can be gathered without resorting to row-level audit. Above all, employ least privilege principle to avoid any users making changes or reading data that they should not.

References

Oracle security step-by-step -A survival guide for Oracle security,
Pete Finnigan 2003, published by SANS Institute

Oracle security handbook - Aaron Newman and Marlene Theriault,
published by Oracle Press.

Pete Finnigan is the author of the recently published book "Oracle security step-by-step - A survival guide to Oracle security" published in January 2003 by the SANS Institute (see <http://store.sans.org>). Pete Finnigan is the founder and CTO of PeteFinnigan.com Limited (<http://www.petefinnigan.com>) a UK based company that specialises in all things Oracle Security related.

[Privacy Statement](#)

Copyright 2006, SecurityFocus