



410.781.6655  
410.781.6625 fax  
6700 Purple Martin Court  
Eldersburg, MD 21784

# **Auditing the Oracle Database**

*(How to find a needle in a haystack)*

**September, 2007**

**Michael C. Capell**  
**Oracle9i and 10g Certified Professional**  
**Principal Engineer**

## Summary

Most discussions of securing one's Oracle Database discuss auditing but not many detail the finer points of what to audit and, more importantly, what to do with the numerous amount of audit data generated. This can be a problem. Customers will immediately answer "everything" if asked what aspects of the database they believe should be audited. While this is a nice approach, the customer does not usually understand the impact of such a statement. Auditing "everything" requires a substantial amount of additional storage to house the audit data and may significantly impact the performance of the system as each statement and transaction will now have the additional overhead of auditing. In addition, after all of this audit data is stored, what does one do with it? Most customers are usually satisfied with auditing their system but few actually do anything with this valuable data. And if nothing is done with it, the additional performance hits and storage were a waste of money, time, and resources.

The purpose of this white paper is to describe a solution to the growing problem of auditing. More and more customers require auditing and this paper will hopefully provide one way to cover the basics and how to actually use the information gathered.

## Configure the Database

In order to take advantage of the auditing capabilities within an Oracle database, several configurations must be made. Oracle provides a script (`cataudit.sql`) which will create a series of audit views which are helpful in monitoring a database. However, the script also grants public access to those views. In order to adhere to most security models, only authorized users should have access to audit information. Thus, the following customized script will run the Oracle-provided SQL code and then revoke the public accesses. It needs to be run as the SYSDBA user and will prompt the user for the name of a log file to record the output:

```
PROMPT Connect as SYSDBA user
CONNECT / AS SYSDBA

SPOOL &&Auditlog;
PROMPT Running CATAUDIT.SQL...
@?/rdbms/admin/cataudit.sql

PROMPT Revoking SELECT privs on audit tables from PUBLIC and
dropping PUBLIC synonyms to audit tables...

REVOKE SELECT ON audit_actions FROM public;
DROP PUBLIC SYNONYM audit_actions;

REVOKE SELECT ON all_def_audit_opts FROM public;
DROP PUBLIC SYNONYM all_def_audit_opts;

REVOKE SELECT ON user_obj_audit_opts FROM public;
DROP PUBLIC SYNONYM user_obj_audit_opts;

REVOKE SELECT ON user_audit_trail FROM public;
DROP PUBLIC SYNONYM user_audit_trail;
```

```
REVOKE SELECT ON user_audit_session FROM public;  
DROP PUBLIC SYNONYM user_audit_session;  
  
REVOKE SELECT ON user_audit_statement FROM public;  
DROP PUBLIC SYNONYM user_audit_statement;  
  
REVOKE SELECT ON user_audit_object FROM public;  
DROP PUBLIC SYNONYM user_audit_object;  
  
SPOOL off;
```

## Protect the Audit Trail

An *audit trail* exists within the Oracle data dictionary as a table named AUD\$. It is designed to store records tracking database statements, privileges, and schema objects. The alternative to storing this information in the database is to use an operating system audit file. Pyxis recommends using the audit trail within the database as Oracle provides predefined views and reports that can access the audit trail. However, with valuable audit information stored in the database, it is imperative that the Database Administrators ensure that no unauthorized users obtain roles or privileges that allow them to tamper with the audit trail. For example, in order to activate auditing, a user needs the AUDIT SYSTEM privilege and in order to delete from the AUD\$ table, a user requires the DELETE\_CATALOG\_ROLE or DELETE ANY TABLE role. Thus, adhering to the ideal of granting least privileges, Database Administrators should make sure that these privileges are granted sparingly and only to users with authorized access.

In Oracle8i and earlier, it was recommended to further protect the audit trail by moving it from the SYSTEM tablespace to a tablespace of its own. Oracle has since altered this recommendation and no longer supports moving the audit trail. Internal Oracle code apparently makes implicit assumptions about the data dictionary tables and the move could cause problems during upgrades or backup/recovery scenarios.

## What to Audit

With the database configured properly, the discussion now moves toward what should be audited. By default, Oracle audits *critical events* and stores the results in the operating system audit file location. If that log is not accessible to Oracle, the default auditing records are written to the audit trail within the database. The critical events as defined by Oracle are:

- **Connections to the instance with administrator privileges** - an audit record is generated that lists the operating system user connecting to Oracle as SYSOPER or SYSDBA.
- **Database Startup** - an audit record is generated that lists the operating system user starting the instance, the user's terminal identifier, the date and time stamp, and whether database auditing was enabled or disabled.
- **Database Shutdown** - an audit record is generated that lists the operating system user shutting down the instance, the user's terminal identifier, and the date and time stamp.

To enable additional auditing and specify the operation system audit file location, several *static* initialization parameters need to be set. That is, after these parameters are set, the database requires a reboot for them to take effect.

```
AUDIT_SYS_OPERATIONS = TRUE
AUDIT_FILE_DEST = <OS Audit File Location>
AUDIT_TRAIL = DB
```

Setting `AUDIT_SYS_OPERATIONS` to `TRUE` ensures that the administrators within the database are audited in addition to the critical actions described above. All of the audit records of the critical events as well as the administrative functions are sent to the operating system audit file defined by the `AUDIT_FILE_DEST` parameter. Pyxis recommends that this file be stored along with the database's other administrative files. Lastly, the `AUDIT_TRAIL` parameter enables and disables *standard* or user-defined auditing throughout the database. The default is `NONE` and equivalent to disabling standard auditing (Oracle will continue to audit the critical events despite the value of `AUDIT_TRAIL`). Pyxis recommends enabling this parameter by setting it to `DB` which directs all standard audit records to the database audit trail.

After the default auditing is configured properly and additional auditing has been enabled, Database Administrators must decide what other events to audit. Pyxis suggests that, in order to maintain a secure system, auditing should be applied to all DBA activities, failed user logons, user profile modifications, and audit trail access. Oracle provides the capability to audit of all these actions easily and the necessary audits can be enabled as follows:

```
SPOOL &&ActivateActionAuditLog;
PROMPT Begin Auditing All DBA Activities...
AUDIT DBA;

PROMPT Begin Auditing Failed User Logons...
AUDIT SESSION WHENEVER NOT SUCCESSFUL;

PROMPT Begin Auditing Profile Modifications...
AUDIT PROFILE;

PROMPT Begin Auditing Audit Access to Audit Trail...
AUDIT ALL ON sys.aud$ BY ACCESS;

SPOOL off;
```

Oracle is now auditing all critical events as well as some basic actions recommended for most secure systems. The customer must now define additional requirements for auditing based upon their security guidelines keeping in mind that every additional audit can impact the amount of storage necessary as well as the overall performance of the system. For security-conscious systems, one additional audit event that might be necessary is the connection of every user whether successful or not. This essentially acts as a log to track user access to the database and can be enabled as follows:

```
PROMPT Begin Auditing All User Logons...
AUDIT CONNECT;
```

The caveat here is that many applications are written using a *proxy user* to connect to the database and not an individual user account. That is, users connect to an application as themselves and then the application connects to the database as APPUSER or some other similar userid. Thus, the audit record would indicate only that a proxy user connected to the database at a certain time, but not contain any specific information about the individual user. Oracle is smart enough, however, to log additional pertinent information such as workstation identification numbers, hostnames, the program that spawned the connection, and most importantly, the *client identifier* of the connection. The client identifier is a value set by the application to accurately track each user within a database. Database Administrators should make sure that when an application connects with a proxy user, the client identifier is set. To do this, the following statement can be added to a database trigger (such as a logon trigger) or a procedure called within the application:

```
EXEC DBMS_SESSION.SET_IDENTIFIER('<CID>');
```

The variable <CID> should be replaced with the client identifier of the user that connected to the application. Whether or not a proxy user connects to the database, Oracle can now accurately track actions within the database to the correct identity.

In addition to logons, Oracle can audit statements, privileges, and objects. Consult the customer to gather requirements for which, if any, of these actions require additional auditing. Once again, keep in mind that each additional action audited will have an effect on performance and storage utilization.

## Process and Monitor the Audit Trail

The next piece of the puzzle is to process all of the information collected. The following `audit.sql` script generates a customized report of the audit trail. The first section of the report details all users in the system and what privileges are audited. The second section details each audit event and includes the timestamp of the event, the userid of the event, the action performed, the privilege required, and the client identifier. The report should be run as SYSDBA:

```
-- AUDIT.SQL
SET PAGESIZE 55;
SET LINESIZE 110;
SPOOL &&AuditTrail
TTITLE 'Audit Options'
COL "User" FORMAT a13;
COL "Privilege" FORMAT a20;
SELECT user_name "User", privilege "Privilege", success,
       failure FROM dba_priv_audit_opts
UNION ALL
SELECT user_name "User", audit_option "Privilege", success,
       failure FROM dba_stmt_audit_opts;

TTITLE 'Audit Trail'
COL "Timestamp" FORMAT a17;
COL "Logoff" FORMAT a17;
COL "UNIX ID" FORMAT a7;
COL "DB User" FORMAT a7;
COL "Client ID" FORMAT a15;
COL "Action" FORMAT a17;
```

```
COL "Priv/Error" FORMAT a17;
SELECT TO_CHAR(timestamp, 'MM/DD/YYYY HH24:MI') "Timestamp",
       TO_CHAR(logoff_time, 'MM/DD/YYYY HH24:MI') "Logoff",
       os_username "UNIX ID", username "DB User",
       client_id "Client ID",
       CASE WHEN action_name = 'SELECT'
            THEN action_name || '(' || obj_name || ')'
            ELSE action_name END "Action",
       CASE WHEN returncode=0 THEN priv_used ELSE '*ORA-' ||
            returncode || '*' END "Priv/Error"
FROM sys.dba_audit_trail ORDER BY timestamp;

SPOOL off;
TTITLE off;
```

This report will detail every event in the database audit trail. Here is a sample:

```
Mon Jan 1                                     page 1
                                     Audit Options

User          Privilege          SUCCESS          FAILURE
-----
AUDIT SYSTEM          BY ACCESS          BY ACCESS
CREATE SESSION        BY ACCESS          BY ACCESS
SYSTEM AUDIT          BY ACCESS          BY ACCESS
CREATE SESSION        BY ACCESS          BY ACCESS
USER                 BY ACCESS          BY ACCESS
PUBLIC SYNONYM        BY ACCESS          BY ACCESS
PUBLIC DATABASE LINK  BY ACCESS          BY ACCESS
ROLE                  BY ACCESS          BY ACCESS
PROFILE              BY ACCESS          BY ACCESS
SYSTEM GRANT          BY ACCESS          BY ACCESS

10 rows selected
```

```
Mon Jan 1                                     page 1
                                     Audit Trail

Timestamp      Logoff          UNIX ID DB User Client ID ACTION          Priv/Error
-----
01/01/2007 09:00 01/01/2007 09:05 appuser APPUSER          LOGOFF          CREATE SESSION
01/01/2007 09:02          appuser APPUSER user1    SELECT (AUD$) *ORA-2004*
```

The first section of the report details the audit options in the database. Since only system events are being audited in this example, and those events apply to the entire database, the `User` column is left blank. The sequence of events depicted in the second section above shows a user (`user1`) connecting to the database via a proxy user (`appuser`) and attempting to spy on the database audit trail. The first record shows the initial connection to the database using the `CREATE SESSION` privilege. No Client Identifier is shown since the application has not yet specified the user (that happens after the initial connection) but the ultimate logoff date is shown to track the user. The second record shows the attempt to issue a `SELECT` statement on the database audit trail (`AUD$`). Since the user has no access, an `ORA-2004` is reported by Oracle in the Privilege/Error column. Here the Client Identifier is clearly visible (`user1`) as is a formatted description of the action the user performed: `SELECT (AUD$)`.

As one might expect, the audit trail will become very large over time as it continually grows. Pyxis recommends taking snapshots of the audit trail using the script above, backing those snapshots up, and

then truncating the audit trail so a new snapshot can be taken at a later date. The following customized shell script (`auditmgr`) will take the log generated from `audit.sql` and move it to a backup location. The name of the file generated by `audit.sql`, as well as a new location for the audit trail, are required as parameters to the script:

```
#!/usr/bin/sh
# Rename Audit Trail and Move to Backup Location
# Usage: auditmgr <Audit Trail> <Backup Directory>

#Verify the correct number of arguments
if [ $# -lt 2 ]
then
    echo "Renames Audit Trail and Moves to Backup Location" 1>&2
    echo "Usage: $0 <Audit Trail> <Audit Trail Backup Directory>"
1>&2
    exit 1
fi

# Get current date and create filename based on date
date=`date +%m%d%y%H%M%S`
filename=$2/audit$date

# Move Audit Trail to Audit Trail Backup Dir with new filename
echo "Moving Audit Trail to $filename..."
mv $1 $filename

#Change privs to Audit Trail as per Oracle Security Guidelines
chmod 700 $filename
```

Armed with `auditmgr` and `audit.sql`, Database Administrators can now implement the Pyxis solution by adding the scripts to their daily regimen. As part of the periodic or nightly backup routine, `audit.sql` should be called followed by `auditmgr` to save the daily audit trail. One might also consider implementing a similar plan for the operating system audit files containing all of the critical event and system operations audit records. In addition, the audit trail should be truncated essentially resetting it for the next snapshot as follows:

```
CONNECT / AS SYSDBA
TRUNCATE TABLE SYS.AUD$;
```

With these scripts and commands running nightly or periodically, a Database Administrator or Security Officer can then review the audit trail in a timely manner. The files will be smaller and more manageable. The more frequent the review, the less overwhelming the amount of data shall be. In addition, extraction programs or filters can be applied to the report to highlight events of more concern. Pyxis recommends a daily review of the audit trail as part of the Database Administrator's normal routine. That is, every morning after grabbing a cup of coffee, the Database Administrator normally checks the status of each database to make sure it is online and available. In addition, logs of the previous night are reviewed to confirm successful backups of all of the pertinent data. Among these tasks, the daily audit trail report should be reviewed for any suspicious activity.



410.781.6655  
410.781.6625 fax  
6700 Purple Martin Court  
Eldersburg, MD 21784

## Conclusion

The customized scripts and recommended commands listed in this paper detail the Pyxis Engineering solution to the overwhelming problem of auditing. This solution is scalable and flexible enough to be tailored to each client's specific needs or domains. Most of the work is done automatically by Oracle and, after the solution is implemented, the only additional task is to review the customized report periodically. This periodic review will prevent the common mistake of collecting audit data but never using it. And Database Administrators will no longer have to worry about trying to find a needle in a haystack.

## References

*Oracle9i Database Administrator's Guide Release 2 (9.2), March 2002, Part No. A96521-01*