
# All About Oracle Auditing – Updated for 12C!
# A White Paper

## February 2015

*Mike Dean*
*Sr Staff Consultant*
*Database Specialists, Inc*
*http://www.dbspecialists.com*

# All About Oracle Auditing - Updated for 12C!

Mike Dean, Sr. Staff Consultant, Database Specialists Inc

## ABSTRACT

Many organizations keep their most sensitive and valuable information in an Oracle database. Properly implemented auditing, as part of a defense-in-depth approach, will help keep it secure and keep you in compliance with ever-increasing security requirements. This paper will give you everything you need to know in order to successfully implement Oracle Auditing; from basic configuration to advanced techniques using Fine-Grained Auditing (FGA). It will also discuss new features and changes related to auditing in Oracle 12c, specifically the Unified Audit Trail and Auditing in a Multi-Tenant database.

## TARGET AUDIENCE

Oracle Database Administrators and others involved in securing Oracle databases.

## EXECUTIVE SUMMARY

After reading this paper, the Reader will be able to:

- Understand how to properly configure, manage and review the Oracle Audit Trail
- Understand new Auditing features introduced in Oracle 11 and Oracle 12
- Understand how to use the Audit trail for troubleshooting purposes

## What is Oracle Auditing and why do I need it?

Auditing is the monitoring and recording, either in the database or in OS files, of selected database actions, from both database users and non-database users. In certain industries, you may be required to implement auditing for regulatory compliance. For example, healthcare companies are regulated by HIPAA and publicly traded companies are subject to Sarbanes-Oxley, both of which have extensive auditing requirements. Auditing can also help to ensure accountability by tracking who does what and when in your database. Auditing can deter inappropriate behavior in the first place as people are less likely to break the rules if they know they are being watched. If you do have a security incident of some sort, audit data can be very valuable for investigation purposes. It also allows you to monitor database activities, for example logon activity, table usage, database changes etc. And finally, it can make life easier for the DBA. This may seem counter-intuitive but I will give some examples later on to illustrate this point.

Note: While much of the information presented in this paper is applicable to any version of Oracle, it is important to note that the majority of the syntax and examples shown were

done with Oracle 11 and should work on all versions prior to 11. Towards the end of the paper, I start discussing Oracle 12C and use syntax and examples that would only work with Oracle 12.

## Mandatory Auditing

Mandatory auditing happens automatically in every database and cannot be turned off. It will record when the database is stopped or started as well as record when a user logs on with either SYSDBA or SYSOPER privileges. This data is written to files in a location determined by the AUDIT_FILE_DEST parameter on Unix or to the Event Viewer on Windows. If you have this parameter set to an invalid directory or run out of space in the directory, you won't be able to start the database or connect as SYSDBA.

```
SQL> alter system set audit_file_dest='/bogus'
scope=spfile;
SQL> shutdown immediate
SQL> startup
ORA-09925: Unable to create audit trail file
Linux-x86_64 Error: 2: No such file or directory
Additional information: 9925
```

Depending on how often users connect as SYSDBA and/or SYSOPER, mandatory auditing can generate a lot of files so keep an eye on the free space for this location. You should also schedule periodic reviews and purging of these files.

## Auditing Actions by SYS

By default, the SYS user is exempt from all auditing policies, and this can leave a huge hole in your audit trail. You can change this behavior by setting the parameter AUDIT_SYS_OPERATIONS=TRUE. When that is set to true, Oracle will record ALL statements issued by SYS to files in a location determined by the AUDIT_FILE_DEST parameter on Unix or to the Event Viewer on Windows. Or if AUDIT_SYSLOG_LEVEL is set, then it will write to the Unix syslog. Obviously, if you login as SYS a lot then this can generate a lot of data. In my opinion, you should set this parameter to TRUE so you can monitor SYS activities.

## AUDIT_SYSLOG_LEVEL

This parameter allows you to integrate the database audit trail with the Unix Syslog and even send it to a remote server using remote syslogd functionality. This can be used in highly secure environments to remove the audit data from its source system. It can also be useful to consolidate audit data from multiple databases into a single file. In order to configure this, you specify a facility clause and priority clause in the AUDIT_SYSLOG_LEVEL parameter and then create a corresponding entry in /etc/syslog.conf specifying a filename.

For example:
```
AUDIT_SYSLOG_LEVEL = 'LOCAL1.WARNING';
```

/etc/syslog.conf:
```
LOCAL1.WARNING /var/log/dbaudit.log
```

## Standard Auditing

Standard auditing is enabled by setting the AUDIT_TRAIL parameter and is configured with the AUDIT/NOAUDIT commands. It can write its records to either the SYS.AUD$ table or to OS files depending on the value of AUDIT_TRAIL. The following are valid values for this parameter:

```
NONE = disables standard auditing
OS = writes audit records to an OS file
DB = writes audit records to the SYS.AUD$ table in the database
DB_EXTENDED = writes audit records to the SYS.AUD$ and includes the
complete SQL statement that was executed along with any bind values
XML = writes audit records to an OS file in XML format
XML_EXTENDED = writes audit records to an OS file in XML format plus
records the SQL statement and bind values to SYS.AUD$
```

The database will need to be restarted for it to take effect. With Standard Auditing, you can track Statements, Privileges and Objects

### *Privilege auditing:*

audit **select any table**;

### *Statement auditing:*

audit **select table**;

### *Object auditing:*

audit select **on SCOTT.SALARY**;

You can specify if you want to track only failed actions or only successful actions or both. You can specify either "by session" or "by access" which determines the granularity at which the audit record is written. You can also track activities by certain users. The details of your auditing config can be found in dba_stmt_audit_opts, dba_priv_audit_opts and dba_obj_audit_opts. You turn off the auditing with the NOAUDIT command.

```
audit select table by session | by access;
audit select table whenever not successful
audit select table by SCOTT;
```

```
noaudit select table;
```

Oracle recommends that you audit BY ACCESS and not BY SESSION in your AUDIT statements. The audit records generated through the BY ACCESS audit option have more information, such as execution status (return code), date and time of execution, the privileges used, the objects accessed, the SQL text itself and its bind values. In addition, the BY ACCESS audit option captures the SCN for each execution and this can help flashback queries.

You can audit the use of any system privilege. Privilege auditing does not occur if the action is already permitted by the existing owner and object privileges. Privilege auditing is triggered only if the privileges are insufficient, that is, only if what makes the action possible is a system privilege. For example, suppose that user SCOTT has been granted the SELECT ANY TABLE privilege and the SELECT ANY TABLE is being audited. If SCOTT selects his own table (for example, SCOTT.EMP), then the SELECT ANY TABLE privilege is not used. Because he performed the SELECT statement within his own schema, no audit record is generated. On the other hand, if SCOTT selects from another schema (for example, the HR.EMPLOYEES table), then an audit record is generated. Because SCOTT selected a table outside his own schema, he needed to use the SELECT ANY TABLE privilege. If Scott has been given direct grants on the other schema's table (as he should be), then he would not need SELECT ANY TABLE and would not be audited.

## Fine Grained Auditing

Fine-Grained Auditing (FGA) is an Enterprise Edition only feature and enables you to create policies that define specific conditions that must take place for the audit to occur, allowing you to monitor data access based on content. It provides granular auditing of queries, and INSERT, UPDATE, and DELETE operations. For example, a CFO must track access to financial records to guard against employee snooping, with enough detail to determine what data was accessed. It is not enough to know that SELECT privilege was used by a specific user on a particular table. Fine-grained auditing provides this deeper functionality.

In general, fine-grained audit policies are based on simple, user-defined SQL predicates on table objects as conditions for selective auditing. During fetching, whenever policy conditions are met for a row, the query is audited.

For example, you can use fine-grained auditing to audit the following types of actions:

- Accessing a table outside of normal working hours
- Logging in from a particular IP address
- Selecting or updating a particular table column

Fine-grained auditing records are stored in the SYS.FGA_LOG$ table. To find the records have been generated for the audit policies that are in effect, you can query the DBA_FGA_AUDIT_TRAIL view. The DBA_COMMON_AUDIT_TRAIL view combines both

standard and fine-grained audit log records. The DBA_AUDIT_TRAIL view contains standard Auditing records.

To create a fine-grained audit policy, use the `DBMS_FGA.ADD_POLICY` procedure. This procedure creates an audit policy using the supplied predicate as the audit condition. Oracle Database executes the policy predicate with the privileges of the user who created the policy. The maximum number of fine-grained policies on any table or view object is 256. Oracle Database stores the policy in the data dictionary table, but you can create the policy on any table or view that is not in the `SYS` schema. After you create the fine-grained audit policy, it does not reside in any specific schema, although the definition for the policy is stored in the `SYS.FGA$` data dictionary table. You cannot modify a fine-grained audit policy after you have created it. If you need to modify the policy, drop it and then recreate it.

The following example will create an FGA policy to record all updates to the SALARY column unless it is done by the owner of the table MIKE. I am using the SYS_CONTEXT function to find information about the user and using that to determine if auditing should occur.

```
begin
DBMS_FGA.ADD_POLICY(
object_schema      => 'MIKE',
object_name        => 'EMPLOYEE',
policy_name        => 'salary_change',
audit_column       => 'SALARY',
audit_condition    =>
'SYS_CONTEXT(''USERENV'',''SESSION_USER'') <> ''MIKE'' ',
enable             =>  TRUE,
statement_types    => 'UPDATE',
audit_trail        =>  DBMS_FGA.DB + DBMS_FGA.EXTENDED);
end;
/
```

One advantage of FGA over Standard Auditing is that you don't need to set the AUDIT_TRAIL parameter for it to work. This means that you can enable auditing without restarting the database.

## What should you be auditing?

In general, you should audit changes to the database (alter database, alter system), DDL, system/object privileges, logon/logoff and unsuccessful operations. See the script at the end of this paper for my recommended audit settings.

More specifically, you should audit data that is sensitive and/or important to your organization. (salaries, classified data, financial info, etc). This requires you to understand your data and is where Fine Grained Auditing may be most appropriate.

## What to Look for in the Audit Trail?

Unless you are looking for something specific, you basically want to keep an eye out for anomalies in the Audit Trail. Of course, what is an anomaly in one database may not be in another. Here are some things you may want to look for:

- DDL not during a scheduled build
- Activity outside of normal working hours
- Failed attempts to access data or exceed privileges
  - ORA-00942: "table or view does not exist"
  - ORA-02004: "security violation"
  - ORA-01031: "insufficient privileges“
- Excessive failed login attempts
- Login attempts by non-existent users
- Alter system/Alter database commands
- Unauthorized privilege/object grants
- Unsuccessful operations – (where aud$.returncode != 0)

## 11g new Auditing features

Oracle 11g introduced a host of changes to the Auditing functionality. One change that could catch you off guard is Oracle 11 now implements auditing by default by setting the AUDIT_TRAIL=DB whenever the Database Configuration Assistant (DBCA) is used to create a database. The script that gets run by DBCA is $ORACLE_HOME/rdbms/admin/secconf.sql. To turn off these audit settings, run the undoaud.sql file in that same directory.

A summary of the changes in Oracle 11:

- Beware of default auditing! (11.1)
- Changes to AUDIT BY SESSION (11.2.0.1) - can dramatically increase the size of the audit trail if you had been auditing by session.
- Separate records for LOGON, LOGOFF now. In Oracle 10, the LOGON record would be updated when the user logged off. Now there are two records.
- DB_EXTENDED is now DB,Extended (11.2.0.1)
- BY ACCESS now the default (11.2.0.2)
- Audit Trail Cleanup process has been improved with DBMS_AUDIT_MGMT (11.2.0.1)
- Caution when upgrading to 11G with auditing turned on. Refer to Note 1329590.1

Oracle Database 11g audits the following privileges by default:

```
ALTER ANY PROCEDURE     CREATE ANY LIBRARY      DROP ANY TABLE
```

| | | |
|---|---|---|
| ALTER ANY TABLE | CREATE ANY PROCEDURE | DROP PROFILE |
| ALTER DATABASE | CREATE ANY TABLE | DROP USER |
| ALTER PROFILE | CREATE EXTERNAL JOB | EXEMPT ACCESS POLICY |
| ALTER SYSTEM | CREATE PUBLIC DATABASE LINK | GRANT ANY OBJECT PRIVILEGE |
| ALTER USER | CREATE SESSION | GRANT ANY PRIVILEGE |
| AUDIT SYSTEM | CREATE USER | GRANT ANY ROLE |
| CREATE ANY JOB | DROP ANY PROCEDURE | |

Oracle Database 11g audits the following SQL shortcuts by default:

| | | |
|---|---|---|
| ROLE | SYSTEM AUDIT | PUBLIC SYNONYM |
| DATABASE LINK | PROFILE | SYSTEM GRANT |

In addition to Oracle 11 Default, I recommend the following AUDIT settings. These will capture most DDL and grants as well as failed SQL.

```
audit not exists;
audit select any table;
audit select any dictionary;
audit SELECT TABLE whenever not successful;
audit INSERT TABLE whenever not successful;
audit UPDATE TABLE whenever not successful;
audit DELETE TABLE whenever not successful;
audit table; (shortcut for create, drop, truncate)
audit alter table;
audit procedure;
audit trigger;
audit view;
audit index;
audit grant procedure;
audit grant table;
```

## Examples of Audit Trail Use for the DBA

There many times over the years when I have found audit data to be useful to me as a DBA. Here are some examples:

- Performance issues caused by excessive logon activity. Being able to quantify which users are connecting how often and for how long can be valuable when faced with performance issues caused by excessive logon activity. This is often the result of a mis-configured connection pool and can be easily fixed once identified.

- "Table Or View Does Not Exist" or "Insufficient Privileges"– which table? When an application cannot access a table because it doesn't exist or insufficient privileges, it will usually just return the Oracle error to the screen. You can easily identify the table(s) involved and the reason for the error by looking in the Audit Trail.

- Who dropped/altered this table/index/procedure? Even in tightly controlled environments, occasionally you have a change occur that no one will own up to.

- Unhandled Oracle errors. When an application doesn't gracefully handle Oracle errors, they will show up in the Audit Trail.

- Inappropriate system privileges. Someone was granted DBA in Production without proper approval – who is responsible?

- Diagnosing connection problems. For example, someone's account is repeatedly being locked without them knowing about it. Use the Audit Trail to determine which OS user and machine the failed login attempts are coming from and you have the culprit.

## Managing the Audit Trail

The first step to managing the audit trail is to move it out of SYSTEM into its own tablespace. In previous versions, this would be done manually

```
create tablespace "AUDIT"
   datafile '$HOME/data/aud01.dbf' size 500k
      default storage (initial 100k next 100k pctincrease 0)
/
create table audx tablespace "AUDIT"
   storage (initial 50k next 50k pctincrease 0)
      as select * from aud$ where 1 = 2
/
rename AUD$ to AUD$$
/
rename audx to aud$
/
create index i_aud2
  on aud$(sessionid, ses$tid)
    tablespace "AUDIT" storage(initial 50k next 50k pctincrease 0)
/
```

Be careful when doing any type of maintenance on the AUD$ or FGA_LOG$ tables that might cause them to get locked or otherwise unavailable if you are currently auditing. I have seen this happen on a database that is auditing logon/logoff activity and it quickly causes a bottleneck as no one will be able to login. You should turn off auditing with the NOAUDIT command before moving the table to another tablespace.

As of 11g, there is the DBMS_AUDIT_MGMT package that can be used to manage the audit data. Here is a 6-step process than can be used to manage your audit data. This will maintain 90 days worth of records in SYS.AUD$ as well as 90 days worth of OS audit files in audit_file_dest.

*Step 1: create a new tablespace*

```
create tablespace aud_data datafile '+DATA' size 2g
autoextend on;
```

*Step 2: Move SYS.AUD$ to dedicated tablespace.*
*Note: This should be done during a slow period or you should turn off session auditing first*

```
begin
DBMS_AUDIT_MGMT.SET_AUDIT_TRAIL_LOCATION(
AUDIT_TRAIL_TYPE => DBMS_AUDIT_MGMT.AUDIT_TRAIL_DB_STD,
AUDIT_TRAIL_LOCATION_VALUE => 'AUD_DATA');
end;
/
```

*Step 3: Initialize the Cleanup process and set the Cleanup interval*

```
begin
DBMS_AUDIT_MGMT.INIT_CLEANUP(
AUDIT_TRAIL_TYPE => DBMS_AUDIT_MGMT.AUDIT_TRAIL_ALL,
DEFAULT_CLEANUP_INTERVAL => 12 );
end;
/
```

*Step 4: Set the archive timestamp to sysdate-90. This tells the clean-up job to delete everything older than 90 days.*

```
begin
DBMS_AUDIT_MGMT.SET_LAST_ARCHIVE_TIMESTAMP(
AUDIT_TRAIL_TYPE => DBMS_AUDIT_MGMT.AUDIT_TRAIL_AUD_STD,
LAST_ARCHIVE_TIME => sysdate-90);
end;
/

begin
DBMS_AUDIT_MGMT.SET_LAST_ARCHIVE_TIMESTAMP(
AUDIT_TRAIL_TYPE => DBMS_AUDIT_MGMT.AUDIT_TRAIL_OS,
LAST_ARCHIVE_TIME => sysdate-90);
end;
/
```

*Step 5: Create the Purge job*

```
begin
DBMS_AUDIT_MGMT.CREATE_PURGE_JOB (
AUDIT_TRAIL_TYPE => DBMS_AUDIT_MGMT.AUDIT_TRAIL_ALL,
AUDIT_TRAIL_PURGE_INTERVAL => 12,
AUDIT_TRAIL_PURGE_NAME => 'AUDIT_TRAIL_PURGE',
USE_LAST_ARCH_TIMESTAMP => TRUE );
end;
/
```

### *Step 6: schedule a job to automatically advance the last_archive_timestamp*

```
create or replace procedure set_archive_retention
(retention in number default 90) as
begin
DBMS_AUDIT_MGMT.SET_LAST_ARCHIVE_TIMESTAMP(
audit_trail_type => DBMS_AUDIT_MGMT.AUDIT_TRAIL_AUD_STD,
last_archive_time => sysdate - retention);

DBMS_AUDIT_MGMT.SET_LAST_ARCHIVE_TIMESTAMP(
AUDIT_TRAIL_TYPE => DBMS_AUDIT_MGMT.AUDIT_TRAIL_os,
LAST_ARCHIVE_TIME => sysdate-retention);
end;
/

BEGIN
   DBMS_SCHEDULER.create_job (
   job_name => 'advance_archive_timestamp',
   job_type => 'STORED_PROCEDURE',
   job_action => 'SET_ARCHIVE_RETENTION',
   number_of_arguments => 1,
   start_date => SYSDATE,
   repeat_interval => 'freq=daily' ,
   enabled => false,
   auto_drop => FALSE);
   dbms_scheduler.set_job_argument_value
    (job_name =>'advance_archive_timestamp',
     argument_position =>1,
     -- one week, you can customize this line:
     argument_value => 90);
   DBMS_SCHEDULER.ENABLE('advance_archive_timestamp');
End;
/
```

## What's New in Oracle 12C?

The biggest change with respect to auditing in Oracle 12C is the Unified Audit Trail. Prior to 12C, there were several different locations where audit records could be found: AUD$, FGA_LOG$, audit_file_dest and Database Vault. Oracle 12C introduces Unified Auditing which captures all audit data in one single location: the unified_audit_trail view. The unified audit trail captures audit information from a variety of sources, such as:

SYSDBA activities
Unified Audit policies
Fine Grained Auditing records
Oracle Database Real Application Security audit records
Oracle Recovery Manager audit records
Oracle Database Vault audit records
Oracle Label Security audit records
Oracle Data Mining records
Oracle Data Pump
Oracle SQL*Loader Direct Load

The unified audit trail, which resides in a read-only table in the `AUDSYS` schema in the `SYSAUX` tablespace, makes this information available in a uniform format in the `UNIFIED_AUDIT_TRAIL` data dictionary view, and is available in both single-instance and Oracle Database Real Application Clusters environments. In addition to the user `SYS`, users who have been granted the `AUDIT_ADMIN` and `AUDIT_VIEWER` roles can query these views. If your users only need to query the views but not create audit policies, then grant them the `AUDIT_VIEWER` role.
When the database is writeable, audit records are written to the unified audit trail. If the database is not writable, then audit records are written to new format operating system files in the `$ORACLE_BASE/audit/$ORACLE_SID` directory.

For newly created Oracle 12C databases, the default mode of auditing is referred to as mixed-mode. This means that both traditional auditing functionality and the new unified auditing both work. You can convert the database to pure unified auditing by shutting down the database and then:

On Unix: relink with uniaud_on option

```
cd $ORACLE_HOME/rdbms/lib
make -f ins_rdbms.mk uniaud_on ioracle
```

On Windows:

```
cd %ORACLE_HOME%\bin
mv orauniadu12.dll.dbl orauniaud12.dll
```

Restart the database (and service on Windows)
One of the advantages of Unified Auditing is improved performance because it can operate in Queued Write mode. This means that audit records are initially written to the SGA and then periodically flushed to disk. There is an init parameter that controls the size of the memory used for this: `UNIFIED_AUDIT_SGA_QUEUE_SIZE` which can

have a value between 1 and 30 MB with a default of 1. In order to control the write mode, use the DBMS_AUDIT_MGMT program.
To use immediate-write mode, run the following procedure:

```
BEGIN
 DBMS_AUDIT_MGMT.SET_AUDIT_TRAIL_PROPERTY(
  DBMS_AUDIT_MGMT.AUDIT_TRAIL_UNIFIED,
  DBMS_AUDIT_MGMT.AUDIT_TRAIL_WRITE_MODE,
  DBMS_AUDIT_MGMT.AUDIT_TRAIL_IMMEDIATE_WRITE);
END;
/
```

To use queued-write mode, run the following procedure:

```
BEGIN
 DBMS_AUDIT_MGMT.SET_AUDIT_TRAIL_PROPERTY(
  DBMS_AUDIT_MGMT.AUDIT_TRAIL_UNIFIED,
  DBMS_AUDIT_MGMT.AUDIT_TRAIL_WRITE_MODE,
  DBMS_AUDIT_MGMT.AUDIT_TRAIL_QUEUED_WRITE);
END;
/
```

One thing to be aware of with Queued Write mode is that if the instance crashes, you can possibly lose the audit records that haven't yet been written to disk.
The syntax for Unified Auditing is different than traditional auditing. You create audit policies that contain all of the directives for what to audit and then enable the policies. For example, if you want to audit all failed SELECT statements on two tables, you can do this:

**Traditional:**

```
audit select on mdean.table1 whenever not successful;
audit select on mdean.table2 whenever not successful;
```

**Unified Auditing:**

```
create audit policy mdean_table1 actions
select on mdean.table1,
select on mdean.table2;
audit policy mdean_table1 whenever not successful;
```

Notice that with Unified Auditing, you can combine multiple audit commands into a single command.
Oracle provides several pre-defined audit policies that can be implemented easily. Refer to the Oracle 12C Security Guide for details on each of these:

```
ORA_LOGON_FAILURES
ORA_SECURECONFIG   <- enabled by default
ORA_DATABASE_PARAMETER
ORA_ACCOUNT_MGMT
ORA_CIS_RECOMMENDATIONS
ORA_RAS_POLICY_MGMT
ORA_RAS_SESSION_MGMT
ORA_DV_AUDPOL
```

## Conditional Auditing

Unified Auditing has the ability to create conditional policies using the SYS_CONTEXT namespace to specify conditions such as a specific OS username or host. This is similar to Fine Grained Auditing with the exception of being able to audit specific columns and to specify event handlers. To create a conditional audit policy, include the WHEN clause in the CREATE AUDIT POLICY statement. The syntax is

```
CREATE AUDIT POLICY policy_name
action_privilege_role_audit_option
[WHEN function_operation_value_list_1 [[AND | OR]
function_operation_value_list_n]
EVALUATE PER STATEMENT | SESSION | INSTANCE];
```

EVALUATE PER refers to the following options:

- STATEMENT evaluates the condition for each relevant auditable statement that occurs.
- SESSION evaluates the condition only once during the session, and then caches and re-uses the result during the remainder of the session. Oracle Database evaluates the condition the first time the policy is used, and then stores the result in UGA memory afterward.
- INSTANCE evaluates the condition only once during the database instance lifetime. After Oracle Database evaluates the condition, it caches and re-uses the result for the remainder of the instance lifetime. As with the SESSION evaluation, the evaluation takes place the first time it is needed, and then the results are stored in UGA memory afterward.

For example, you can audit updates and deletes unless coming from a particular host

```
CREATE AUDIT POLICY orders_table_audit
ACTIONS UPDATE ON OE.ORDERS, DELETE ON SALES.ORDERS
WHEN 'SYS_CONTEXT (''USERENV'', ''HOST'') NOT IN
(''sales_24'',''sales_12'')'
EVALUATE PER STATEMENT;
```

Note there is a 4000 byte maximum for WHEN clause

## What is the performance impact of Auditing?

In my experience, Auditing does not impose a significant burden of overhead and its benefits far outweigh the costs. The possible exception to this would be with auditing logon/logoff activity. Some applications generate hundreds or even thousands of connections per second and any delay could quickly become a bottleneck. In this case, the ideal solution would be to modify the application to have less connections but of course that isn't always possible. Prior to the release of 12C, Oracle published a really

good white paper discussing this subject called Oracle Database Auditing: Performance Guidelines and includes the following charts. It makes sense that the most resource intensive type of auditing is DB,Extended as it records the entire SQL statement and bind values. Keep in mind that these tests were generating 200-250 audit records per second which is likely a lot more than most systems would actually generate.

**Overhead of Standard Auditing during TPC-C OLTP benchmark (based on Oracle 11)**

| Audit Trail Setting | Additional Throughput Time | Additional CPU Usage |
|---|---|---|
| OS | 1.39% | 1.75% |
| XML | 1.70% | 3.51% |
| XML, Extended | 3.70% | 5.26% |
| DB | 4.57% | 8.77% |
| DB, Extended | 14.09% | 15.79% |

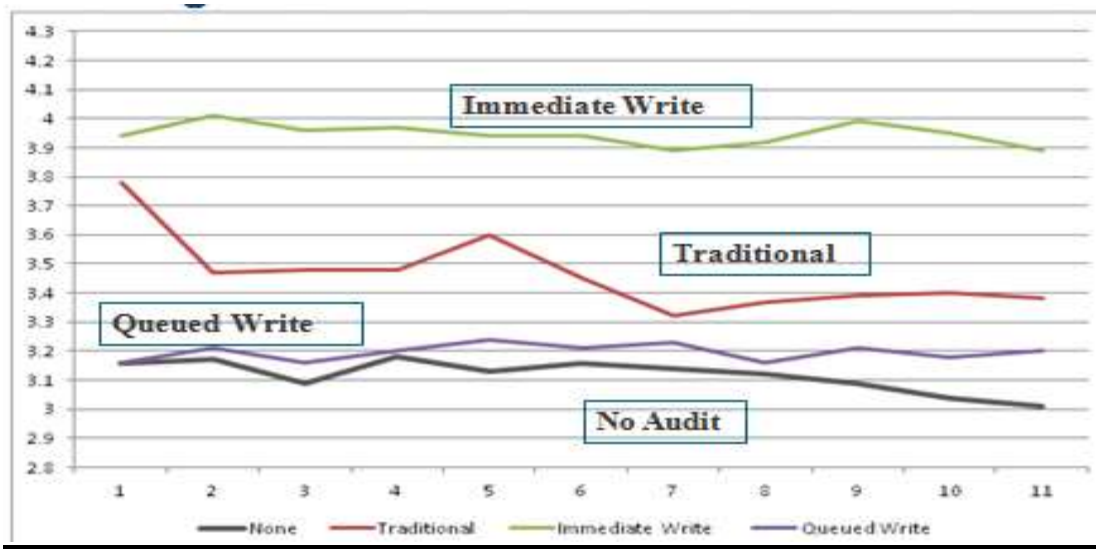
**Overhead of Fine Grained Auditing during TPC-C OLTP benchmark**

| Audit Trail Setting | Additional Throughput Time | Additional CPU Usage |
|---|---|---|
| XML | 3.66% | 4.35% |
| XML, Extended | 4.62% | 9.09% |
| DB | 6.60% | 11.11% |
| DB, Extended | 9.61% | 20% |

Source: **Oracle Database Auditing Performance (Doc ID 1509723.1)**

With Oracle 12C, I haven't found any specific benchmarks from Oracle but they do say that Unified Auditing is "much faster" than traditional auditing. I performed some rather simple but effective tests in order to validate this statement. For each timed test, I did 1000 select statements from a small table and audited each select, generating 1000 audit records per test. I tested Unified Auditing in Immediate Write mode and Queued Write mode along with Traditional auditing as well as a control group with no auditing. As you

can see from the graph, Oracle's "much faster" statement seems to be true as long as you are using Queued Write mode. There is very little overhead. Please keep in mind that this was a very basic benchmark test and you would need to judge the impact of auditing in your own environment.




Along with using Queued Write mode to improve auditing performance, there are some other things to keep in mind as well. You should limit the number of policies created, the new syntax allows for multiple audit settings to be configured in one policy. When using Queued Write mode, you can adjust the UNIFIED_AUDIT_SGA_QUEUE_SIZE parameter to allow more in-memory storage. Also, do not mix traditional auditing with Unified Auditing - this will just increase the amount of overhead. And of course, try to minimize the amount of audit data being written.

## Auditing in a Multi-Tenant Environment

With Oracle 12C came the new Multi-Tenant architecture, consisting of Container Databases (CDB) and Pluggable Databases (PDB). Auditing, both traditional and Unified Audit policies, can be done at either the CDB or PDB level. A local audit policy exists at either the CDB or PDB level. If at the PDB level, then it can only apply to local users and objects. If it is at the CDB level, then it can apply to either common or local objects/users. A common audit policy exists in the CDB only and applies only to common objects and users. To specify if it is a local or common policy, use the CONTAINER keyword

This is a common policy and can only exist in the CDB and applies only to common objects/users:

```
CREATE AUDIT POLICY dict_updates
ACTIONS UPDATE ON SYS.USER$,
DELETE ON SYS.USER$,
```

```
UPDATE ON SYS.LINK$,
DELETE ON SYS.LINK$
CONTAINER = ALL;
```

This is local policy and can exist in the CDB or PDB but only applies to local objects/users:

```
CREATE AUDIT POLICY pdb_security
ACTIONS UPDATE ON sales_owner.app_security
CONTAINER = CURRENT;
```

## Conclusion

Oracle provides a wealth of functionality that can be used to audit various aspects of your database. When properly implemented and reviewed on a regular basis, auditing is an important and useful tool for securing your database and maintaining compliance with your organizational security requirements. It can also be useful to the DBA for troubleshooting purposes. With Oracle 12C, the auditing functionality has been improved and expanded and becomes even more valuable as a security and troubleshooting tool.

## Appendix

Use these commands to implement my recommended set of AUDIT settings: (traditional auditing prior to 12C)

```
AUDIT ALTER ANY PROCEDURE;
AUDIT CREATE ANY LIBRARY;
AUDIT DROP ANY TABLE;
AUDIT ALTER ANY TABLE;
AUDIT CREATE ANY PROCEDURE;
AUDIT DROP PROFILE;
AUDIT ALTER DATABASE;
AUDIT CREATE ANY TABLE;
AUDIT DROP USER;
AUDIT ALTER PROFILE;
AUDIT CREATE EXTERNAL JOB;
AUDIT EXEMPT ACCESS POLICY;
AUDIT ALTER SYSTEM;
AUDIT CREATE PUBLIC DATABASE LINK;
AUDIT GRANT ANY OBJECT PRIVILEGE;
AUDIT ALTER USER;
AUDIT CREATE SESSION;
AUDIT GRANT ANY PRIVILEGE;
AUDIT AUDIT SYSTEM;
AUDIT CREATE USER;
AUDIT GRANT ANY ROLE;
```

```
AUDIT CREATE ANY JOB;
AUDIT DROP ANY PROCEDURE;
AUDIT ROLE;
AUDIT SYSTEM AUDIT;
AUDIT PUBLIC SYNONYM;
AUDIT DATABASE LINK;
AUDIT PROFILE;
AUDIT SYSTEM GRANT;
audit not exists;
audit select any table;
audit select any dictionary;
audit SELECT TABLE whenever not successful;
audit INSERT TABLE whenever not successful;
audit UPDATE TABLE whenever not successful;
audit DELETE TABLE whenever not successful;
audit table;
audit alter table;
audit procedure;
audit trigger;
audit view;
audit index;
audit grant procedure;
audit grant table;
```

Use these commands to turn off all of the above AUDIT settings:

```
noaudit all;
noaudit all privileges;
noaudit exempt access policy;
noaudit select any dictionary;
noaudit SELECT TABLE ;
noaudit INSERT TABLE ;
noaudit UPDATE TABLE ;
noaudit DELETE TABLE ;
noaudit grant procedure;
noaudit grant table;
noaudit alter table;
```