14

Implementing Oracle Database Auditing

ORACLE

Objectives

After completing this lesson, you should be able to:

- Describe DBA responsibilities for security and auditing
- Enable unified auditing
- Create unified audit policies
- Maintain the audit trail

Database Security

A secure system ensures the confidentiality of the data that it contains. There are several aspects of security:

- Restricting access to data and services
- Authenticating users
- Monitoring for suspicious activity

Oracle Database provides the industry's best framework for a secure system. But for that framework to be effective, the database administrator must follow best practices and continually monitor database activity.

Restricting Access to Data and Services

All users must not have access to all data. Depending on what is stored in your database, restricted access can be mandated by business requirements, by customer expectations, and (increasingly) by legal restrictions. Credit card information, health-care data, identity information, and so on must be protected from unauthorized access. The Oracle Database server provides extremely fine-grained authorization controls to limit database access. Restricting access must include applying the principle of least privilege.

Authenticating Users

To enforce access controls on sensitive data, the system must first know who is trying to access the data. Compromised authentication can render all other security precautions useless. The most basic form of user authentication is challenging users to provide something that they know, such as a password. Ensuring that passwords follow simple rules can greatly increase the security of your system. Stronger authentication methods include requiring users to provide something that they have, such as a token or public key infrastructure (PKI) certificate. An even stronger form of authentication is to identify users through a unique biometric characteristic such as a fingerprint, an iris scan, bone structure patterns, and so on. The Oracle Database server supports advanced authentication techniques (such as token-, biometric-, and certificate-based identification) through the Oracle Advanced Security option. User accounts that are not in use must be locked to prevent attempts to compromise authentication.

Monitoring for Suspicious Activity

Even authorized and authenticated users can sometimes compromise your system. Identifying unusual database activity (such as an employee who suddenly begins querying large amounts of credit card information, research results, or other sensitive information) can be the first step to detecting information theft. The Oracle Database server provides a rich set of auditing tools to track user activity and identify suspicious trends.

Monitoring for Compliance

- Monitoring or auditing must be an integral part of your security procedures.
- Review the following:
  - Mandatory auditing
  - Standard database auditing
  - Value-based auditing
  - Fine-grained auditing (FGA)

Auditing, which means capturing and storing information about what is happening in the system, increases the amount of work the system must do. Auditing must be focused so that only events that are of interest are captured. Properly focused auditing has minimal impact on system performance. Improperly focused auditing can significantly affect performance.

- **Mandatory auditing:** All Oracle databases audit certain actions regardless of other audit options or parameters. The reason for mandatory audit logs is that the database needs to record some database activities, such as connections by privileged users.
- **Standard database auditing:** Select the objects and privileges that you want to audit and create the appropriate audit policies.
- **Value-based auditing:** Extends standard database auditing, capturing not only the audited event that occurred but also the actual values that were inserted, updated, or deleted. Value-based auditing is implemented through database triggers.
- **Fine-grained auditing (FGA):** Extends standard database auditing, capturing the actual SQL statement that was issued rather than only the fact that the event occurred.

Types of Activities to be Audited

You can audit the following types of activities:

- User accounts, roles, and privileges
- Object actions
- Application context values
- Oracle Data Pump
- Oracle Database Real Application Security
- Oracle Database Vault
- Oracle Label Security
- Oracle Recovery Manager
- Oracle SQL*Loader direct path events

Through the use of auditing policies, you can configure audit settings for the following activities:

- Logging on to the database and the use of privileges and roles
- Executing SQL statements against specific database objects
- Application context values
- Utilities and features:
  - Oracle Data Pump
  - Oracle Database Real Application Security
  - Oracle Database Vault
  - Oracle Label Security
  - Oracle Recovery Manager
  - Oracle SQL*Loader Direct Load

Mandatorily Audited Activities

The following activities are audited:

- CREATE/ALTER/DROP AUDIT POLICY
- AUDIT/NOAUDIT
- EXECUTE of:
  - DBMS_FGA
  - DBMS_AUDIT_MGMT
- ALTER TABLE against AUDSYS audit trail table
- Top-level statements by administrative users (SYS, SYSDBA, SYSOPER, SYSASM, SYSBACKUP, SYSDG, and SYSKM) until the database opens

The following audit-related activities are mandatorily audited:

- CREATE/ALTER/DROP AUDIT POLICY
- AUDIT/NOAUDIT
- EXECUTE of the DBMS_FGA PL/SQL package
- EXECUTE of the DBMS_AUDIT_MGMT PL/SQL package
- ALTER TABLE attempts on the AUDSYS audit trail table
- Top-level statements by SYS, SYSDBA, SYSOPER, SYSASM, SYSBACKUP, SYSDG, and SYSKM until the database opens. After the database opens, these users are audited based on the defined audit settings.

Understanding Auditing Implementation

- *Mixed mode auditing* is the default when a new Oracle Database 12*c* database is created.
- Mixed mode auditing enables the use of:
  - Pre–Oracle Database 12*c* auditing features
  - *Unified auditing* features of Oracle Database 12*c*
- The recommendation from Oracle is to migrate to unified auditing.
- Query `V$OPTION` to determine if the database has been migrated to unified auditing:

```
SELECT value FROM v$option
WHERE parameter = 'Unified Auditing'
```


Prior to Oracle Database 12*c*, audit records from various sources were stored in different locations. Oracle Database 12*c* supports *unified auditing*, in which all audit records are stored in a single audit table.

When you create a new Oracle Database 12*c* database, mixed mode auditing is enabled. This mode enables you to use the auditing features available before Oracle Database 12*c* and also the unified auditing features. Mixed mode auditing is enabled by default through the `ORA_SECURECONFIG` predefined auditing policy for newly created databases.

If you are upgrading a database to Oracle Database 12*c*, you must manually migrate to unified auditing to use the unified auditing features.

Oracle Corporation recommends that you migrate to unified auditing.

Administering the Roles Required for Auditing

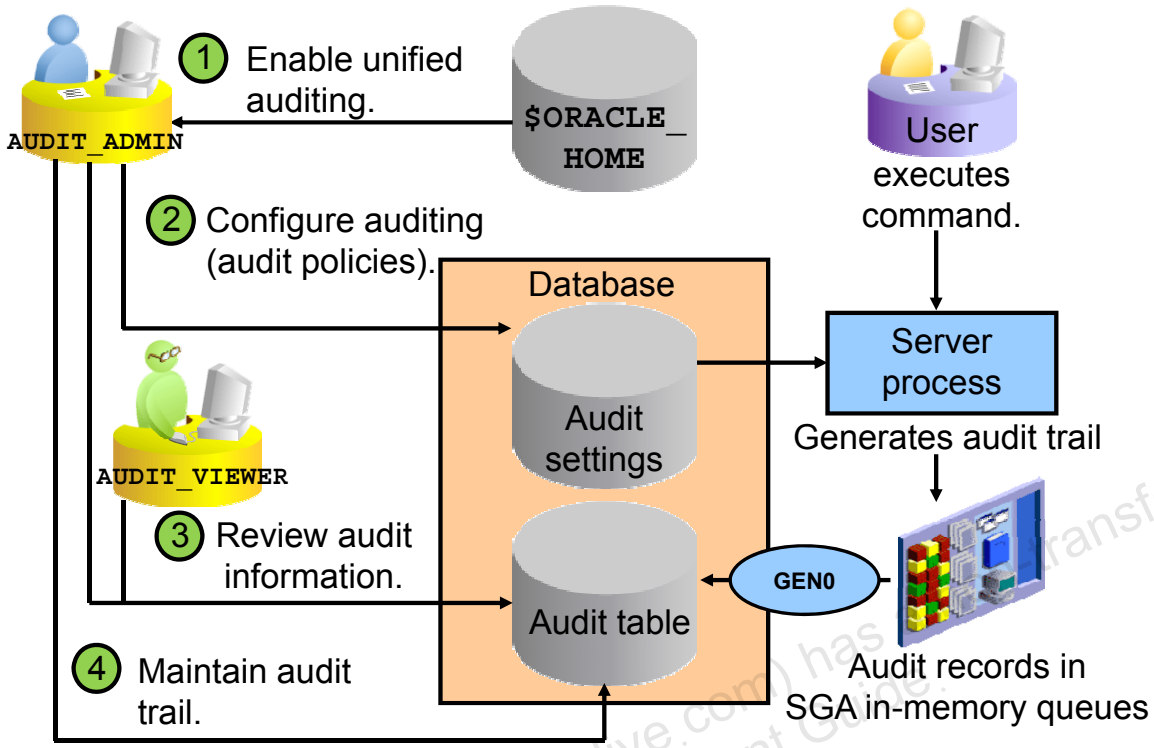
A user must be granted one of the following roles to perform auditing:

- `AUDIT_ADMIN` enables the user to:
  - Create unified and fine-grained audit policies
  - Execute the `AUDIT` and `NOAUDIT` SQL statements
  - View audit data
  - Manage the audit trail (table in the `AUDSYS` schema)
- `AUDIT_VIEWER` enables the user to:
  - View and analyze audit data

Users must be granted the appropriate privilege to configure auditing and view audit data. To support separation of duty, two default roles are provided:

- `AUDIT_ADMIN`: Enables the grantee to configure auditing settings, create and administer audit policies (unified and find-grained), and view and analyze audit data. This role is typically granted to a security administrator.
- `AUDIT_VIEWER`: Enables the grantee to view and analyze audit data. This role is typically granted to external auditors.

Database Auditing: Overview



To use the unified audit trail features, you must first enable unified auditing.

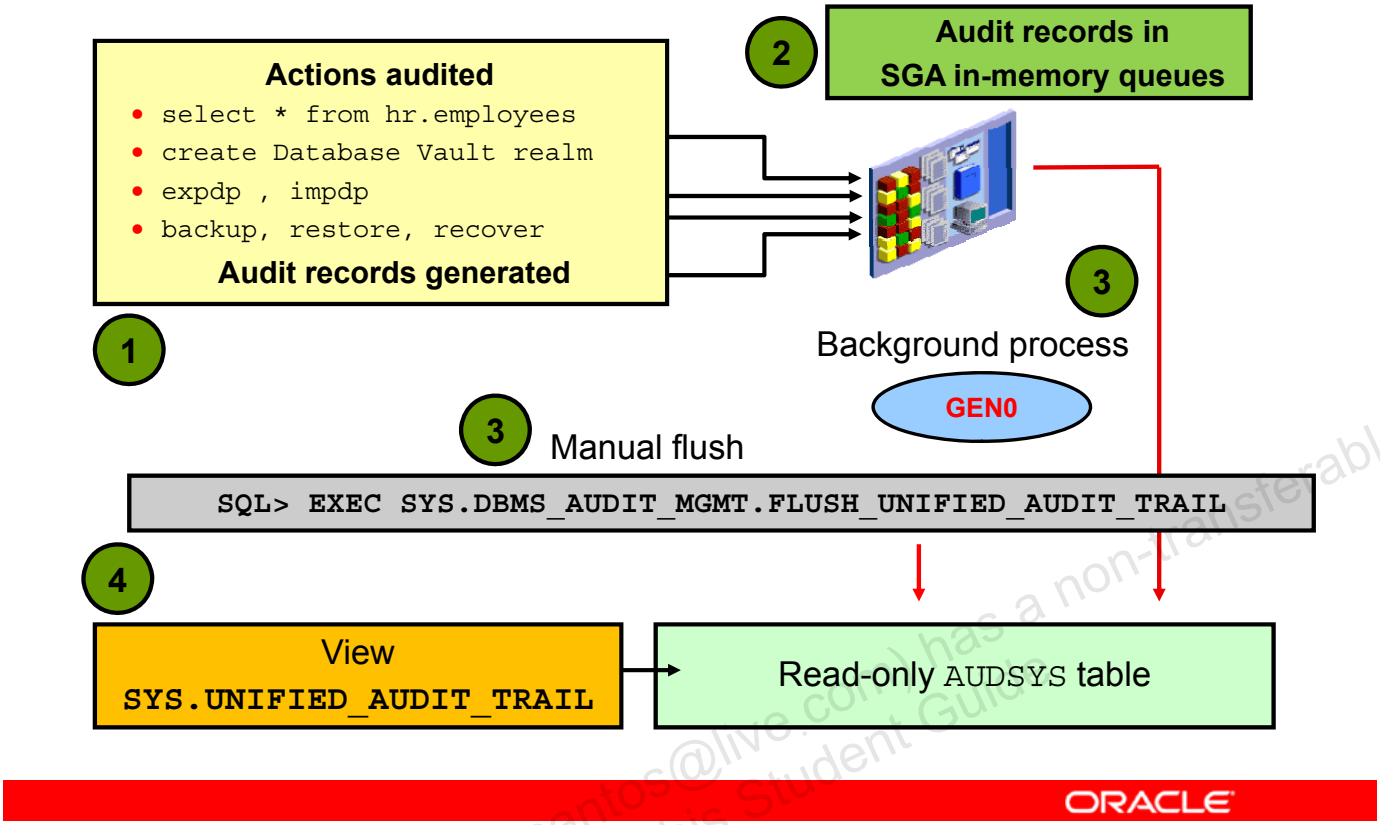
You must then create and enable unified audit policies to specify which activities should be audited.

When a user executes a command or performs an activity that is defined in an auditing policy, audit records are generated. The audit records are written to an audit table in the `AUDSYS` schema and can be viewed by querying the `UNIFIED_AUDIT_TRAIL` view.

Maintaining the audit trail is an important administrative task. Depending on the focus of the audit options, the audit trail can grow very large very quickly. If not properly maintained, the audit trail can create so many records that it affects the performance of the system. Audit overhead is directly related to the number of records that are produced.

Detailed information about all these steps is provided later in this lesson.

Understanding the Audit Architecture



The default mode of operation for unified auditing is *queued write mode*. Audit records are written to a queue in the System Global Area (SGA). The named queues in the SGA have a corresponding persistent storage in tables owned by AUDSYS. When the SGA queue overflows, a background process flushes the contents to the table(s).
Each client has two SGA queues so that a client can continue to write to the second queue while the first queue is being written to the table.

You can use the following procedure to explicitly flush the queues to disk so that you view the audit trail records immediately:

```
SQL> EXEC SYS.DBMS_AUDIT_MGMT.FLUSH_UNIFIED_AUDIT_TRAIL
```

Enabling Unified Auditing

1. In SQL*Plus, shut down the database instance:

```
SQL> SHUTDOWN IMMEDIATE
```

2. Shut down the listener:

```
$ lsnrctl stop
```

3. At the operating system prompt, enable the unified auditing executable:

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

4. Restart the listener:

```
$ lsnrctl start
```

5. In SQL*Plus, restart the database instance:

```
SQL> STARTUP
```


Before enabling the unified auditing executable, shut down the database instance and the listener.

Change to the `$ORACLE_HOME/rdbms/lib` directory and execute the following command:

```
make -f ins_rdbms.mk uniaud_on ioracle ORACLE_HOME=$ORACLE_HOME
```

The `make` command is used to relink the Oracle executable with a different set of libraries to enable unified auditing.

After the `make` command completes, restart the listener and the database instance.

You can log in to SQL*Plus and verify that unified auditing has been enabled as follows:

```
SQL> select value
  2  from v$option
  3  where parameter = 'Unified Auditing';

VALUE
----------------------------------------------------------------
TRUE
```

Configuring Auditing

Method	Description
Unified audit policies	Group audit settings into a policy
Default unified audit policies	Three default policies: ORA_SECURECONFIG ORA_DATABASE_PARAMETER_AUDIT ORA_ACCOUNT_MGMT_AUDIT
Fine-grained audit policies	Define specific conditions that must be met for auditing to take place

You can configure auditing by grouping audit settings into a unified audit policy.

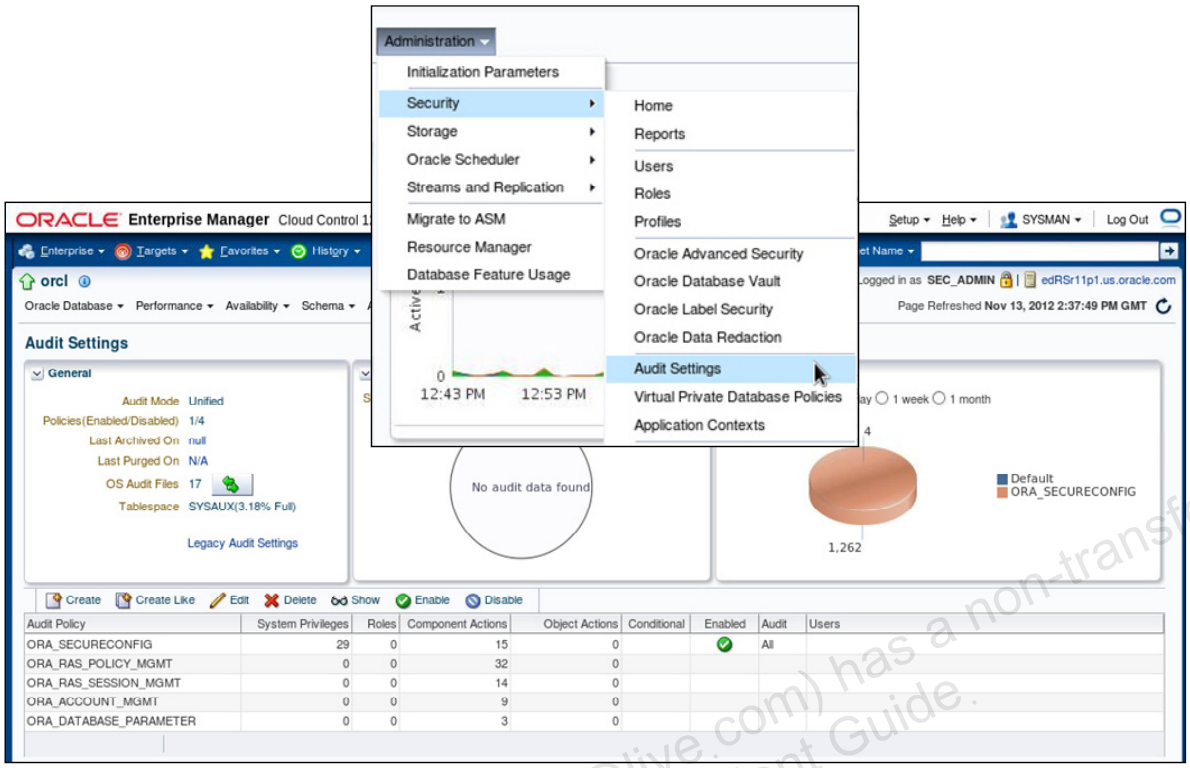
Oracle Database includes three predefined unified audit policies, which include commonly used security audit settings:

- ORA_SECURECONFIG: Contains the same default audit settings as Oracle Database 11*g*. This policy is enabled by default for unified auditing and mixed-mode auditing environments. Refer to the *Oracle Database Security Guide 12c* for a complete list of settings.
- ORA_DATABASE_PARAMETER_AUDIT: Contains audit settings for commonly used Oracle Database commands that set parameters (ALTER DATABASE, ALTER SYSTEM, and CREATE SPFILE)
- ORA_ACCOUNT_MGMT_AUDIT: Contains audit settings for commonly used user account and privilege commands (CREATE USER, ALTER USER, DROP USER, CREATE ROLE, DROP ROLE, ALTER ROLE, SET ROLE, GRANT, and REVOKE)

Fine-grained audit policies enable you to define specific conditions that generate an audit record.

Additional information about these methods is provided later in the lesson.

Using Enterprise Manager Cloud Control



You can use Enterprise Manager Cloud Control to view and manage audit settings.

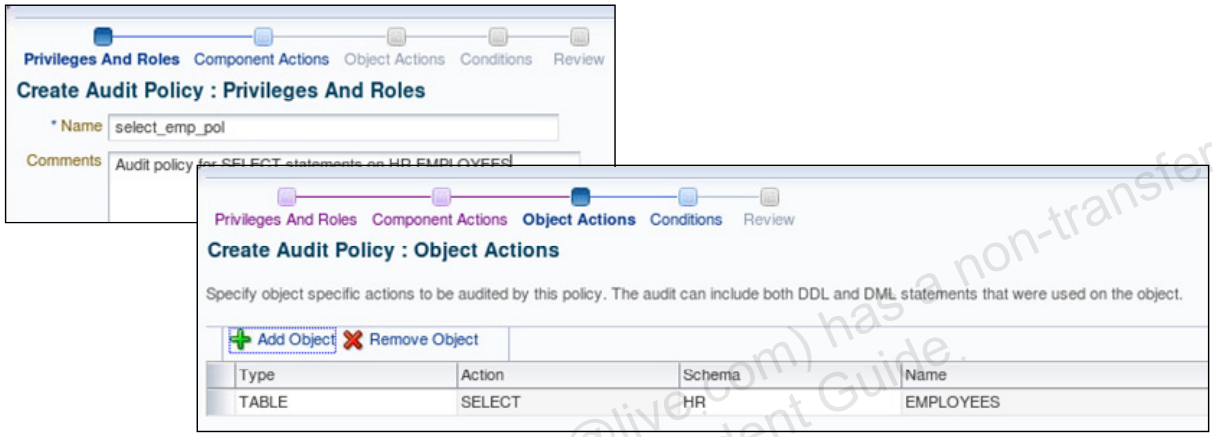
Access the Audit Settings page by expanding the Administration menu on your database home page. Expand Security and select Audit Settings.

Creating a Unified Audit Policy

- Use the `CREATE AUDIT POLICY` statement:

```
CREATE AUDIT POLICY select_emp_pol
ACTIONS select on hr.employees
```

- Use Enterprise Manager Cloud Control:



Use the `CREATE AUDIT POLICY` statement to create a unified audit policy. The example in the slide shows how to create an audit policy named `SELECT_EMP_POL`. This policy specifies that `SELECT` statements against the `HR.EMPLOYEES` table will be audited.

You can also create audit policies by using Enterprise Manager Cloud Control. On the Audit Settings page, click Create to invoke the wizard.

Creating an Audit Policy: System-Wide Audit Options

- System privileges:

```
CREATE AUDIT POLICY audit_syspriv_pol1
PRIVILEGES SELECT ANY TABLE, CREATE LIBRARY
```

- Actions:

```
CREATE AUDIT POLICY audit_actions_pol2
ACTIONS AUDIT, ALTER TRIGGER
```

- Roles:

```
CREATE AUDIT POLICY audit_role_pol3
ROLES mgr_role
```

- System privileges, actions, and roles:

```
CREATE AUDIT POLICY audit_mixed_pol4
PRIVILEGES DROP ANY TABLE
ACTIONS    CREATE TABLE, DROP TABLE, TRUNCATE TABLE
ROLES      emp_role
```

ORACLE

You can create an audit policy with system-wide or object-specific audit options.

The system-wide options can be of three types:

- The privilege audit option audits all events that exercise the specified system privilege, as in the first example in the slide.
- The action audit option indicates which RDBMS action should be audited, such as the `ALTER TRIGGER` action in the second example.
- The role audit option audits the use of all system or object privileges granted directly to the `MGR_ROLE` role, as in the third example.

You can configure privilege, action, and role audit options together in the same audit policy, as shown in the fourth example.

You can find a list of auditable system-wide options in the `SYS.AUDITABLE_SYSTEM_ACTIONS` table.

Creating an Audit Policy: Object-Specific Actions

Create audit policies based on object-specific options.

```
CREATE AUDIT POLICY audit_objpriv_pol5
ACTIONS SELECT, UPDATE, LOCK ON hr.employees
```

```
CREATE AUDIT POLICY audit_objpriv_pol6
ACTIONS ALL
```

```
CREATE AUDIT POLICY audit_objpriv_pol7
ACTIONS EXECUTE, GRANT ON hr.raise_salary_proc
```


Object-specific options are the second type of audit options. These options are actions that are specific to objects in the database. The object-level action audit options are listed in the `SYS.AUDITABLE_OBJECT_ACTIONS` table.

The first example in the slide creates an audit policy to audit any select and update action on any object, and any lock on the `HR.EMPLOYEES` table.

The second example creates an audit policy to audit any object-specific action on any object.

The third example creates an audit policy to audit any execute action on any procedural object, and any grant on the `HR.RAISE_SALARY_PROC` procedure.

The object-level audit options are dynamic. That is, changes in these options become applicable for current and subsequent user sessions.

Creating an Audit Policy: Specifying Conditions

- Condition and evaluation **PER SESSION**

```
CREATE AUDIT POLICY audit_mixed_pol5
ACTIONS RENAME ON hr.employees,ALTER  ON hr.jobs,
WHEN 'SYS_CONTEXT (''USERENV'', ''SESSION_USER'')=''JIM'''
EVALUATE PER SESSION
```

- Condition and evaluation **PER STATEMENT**

```
CREATE AUDIT POLICY audit_objpriv_pol6
ACTIONS ALTER ON OE.ORDERS
WHEN 'SYS_CONTEXT(''USERENV'',''CLIENT_IDENTIFIER'')=''OE'''
EVALUATE PER STATEMENT
```

- Condition and evaluation **PER INSTANCE**

```
CREATE AUDIT POLICY audit_objpriv_pol7
ROLES dba
WHEN SYS_CONTEXT(''USERENV'',''INSTANCE_NAME'')=''sales'''
EVALUATE PER INSTANCE
```


Audit policies can evaluate the condition per statement, once per session or once per instance.

The first example in the slide creates an audit policy to audit any rename action on the `HR.EMPLOYEES` table, and any alter action on the `HR.JOBS` table, provided that the user executing the audited statement is JIM. The condition is evaluated only once in the session.

The second example creates an audit policy to audit any alter action on the `OE.ORDERS` table, provided that the user executing the audited statement is the schema owner `OE`. The condition is evaluated each time an `ALTER` statement is executed on the `OE.ORDERS` table.

The third example creates an audit policy to audit any privilege granted to the DBA role, provided that the instance name is “sales”. The condition is evaluated only once during the database instance lifetime. After Oracle Database evaluates the condition, it caches and reuses the result for the remainder of the instance lifetime.

Enabling and Disabling Audit Policies

Enable audit policies:

- Apply to all users.

```
SQL> AUDIT POLICY audit_syspriv_pol1;
```

- Apply only to some users.

```
SQL> AUDIT POLICY audit_pol2 BY scott, oe;
SQL> AUDIT POLICY audit_pol3 BY sys;
```

- Exclude some users.

```
SQL> AUDIT POLICY audit_pol4 EXCEPT jim, george;
```

- Audit the recording based on failed or succeeded actions.

```
SQL> AUDIT POLICY audit_syspriv_pol1 WHENEVER SUCCESSFUL ;
SQL> AUDIT POLICY audit_objpriv_pol2 WHENEVER NOT SUCCESSFUL ;
```

```
SQL> AUDIT POLICY auditpol5 BY joe WHENEVER SUCCESSFUL ;
```

Disable audit policies by using the `NOAUDIT` command.

After creating the audit policy, you must enable it by using the `AUDIT` statement.

All users are audited by default, except if you define a list of those audited. Apply the audit policy to one or more users by using the `BY` clause. The audit administrator audits `SYS`-user-specific actions the same way as other user actions. Exclude some users by using the `EXCEPT` clause.
You cannot have a `BY` list and an `EXCEPT` list in the same policy enablement statement

Audit records are generated whether the user's actions failed or succeeded. If you want to audit actions only when the user's actions failed, use the `WHENEVER NOT SUCCESSFUL` clause. If you want to audit actions only when the user's actions succeeded, use the `WHENEVER SUCCESSFUL` clause. When you omit the `WHENEVER` clause, the statement is audited whether the action is successful or not, and the `RETURN_CODE` column displays whether the action succeeded or not.

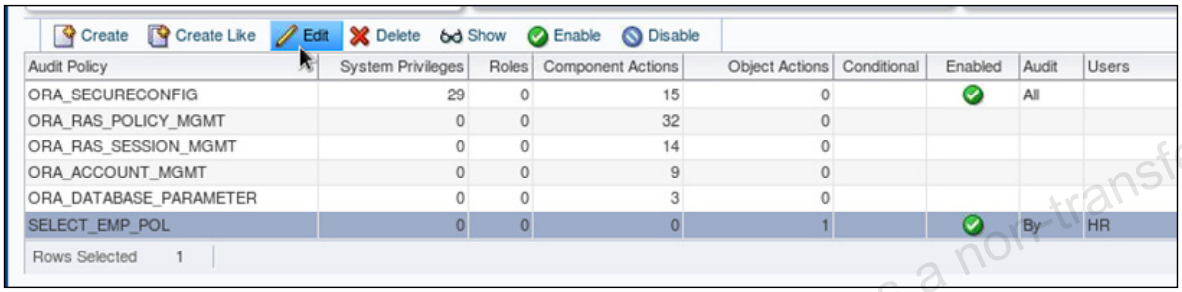
To disable an audit policy, use the `NOAUDIT` command.

Altering a Unified Audit Policy

- Use the `ALTER AUDIT POLICY` statement:

```
ALTER AUDIT POLICY select_emp_pol
ADD ACTIONS select on hr.job_history
```

- Use Enterprise Manager Cloud Control:



You can change most properties in a unified audit policy, except a CONTAINER setting. You can make changes to enabled and disabled audit policies. You do not need to re-enable an audit policy after altering it.

For an object unified audit policy, the new audit settings take place immediately after it has been altered, for both the active and subsequent user sessions. If you alter system audit options, or audit conditions of the policy, then they are activated for new user sessions, but not the current user session.

To alter an audit policy by using Enterprise Manager Cloud Control, select the policy on the Audit Settings page and click Edit.

Viewing Audit Policy Information

```
SQL> SELECT policy_name, audit_option, condition_eval_opt
  2  FROM   audit_unified_policies;

POLICY_NAME          AUDIT_OPTION     CONDITION_EVAL_OPT
-------------------- ---------------- ----------------
POL1                 DELETE           INSTANCE
POL2                 TRUNCATE TABLE   NONE
POL3                 RENAME           SESSION
POL4                 ALL ACTIONS      STATEMENT
```

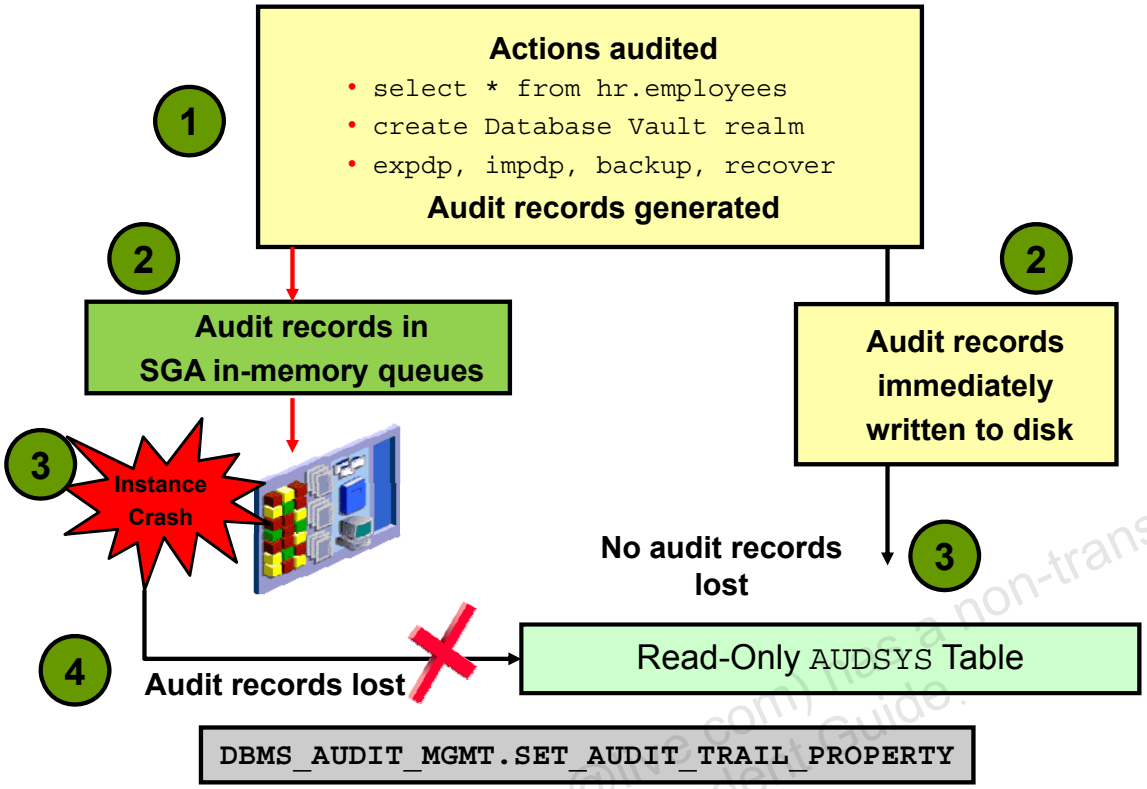
```
SQL> SELECT policy_name, enabled_opt, user_name, success, failure
  2  FROM   audit_unified_enabled_policies;

POLICY_NAME          ENABLED_ USER_NAME  SUC FAI
-------------------- -------- ---------- --- ---
POL3                 BY       PM         NO  YES
POL2                 EXCEPT   SYSTEM     NO  YES
POL4                 BY       SYS        YES YES
POL6                 BY       ALL USERS  YES NO
```

Query `AUDIT_UNIFIED_POLICIES` to view a list of the existing audit policies. The `CONDITION_EVAL_OPT` column defines when the condition, if any, is evaluated.

Query `AUDIT_UNIFIED_ENABLED_POLICIES` to view a list of enabled audit policies. The `ENABLED_OPT` column indicates whether a list of audited users is defined with the `BY` value or an exception list of excluded users is defined with the `EXCEPT` value. The audited or excluded users are listed in the `USER_NAME` column. The `SUCCESS` and `FAILURE` columns indicate whether the policy generates audit records only when the user's actions succeed or fail.

Setting the Write Mode for Audit Trail Records



The in-memory queuing infrastructure means that the audit records are not written immediately to the AUDSYS table, but are queued in the SGA. In cases such as an instance crash, *queued* audit records can be lost. Hence, the audit trail must be configurable to indicate the tolerance level. The following two modes are based on different levels of loss tolerance:

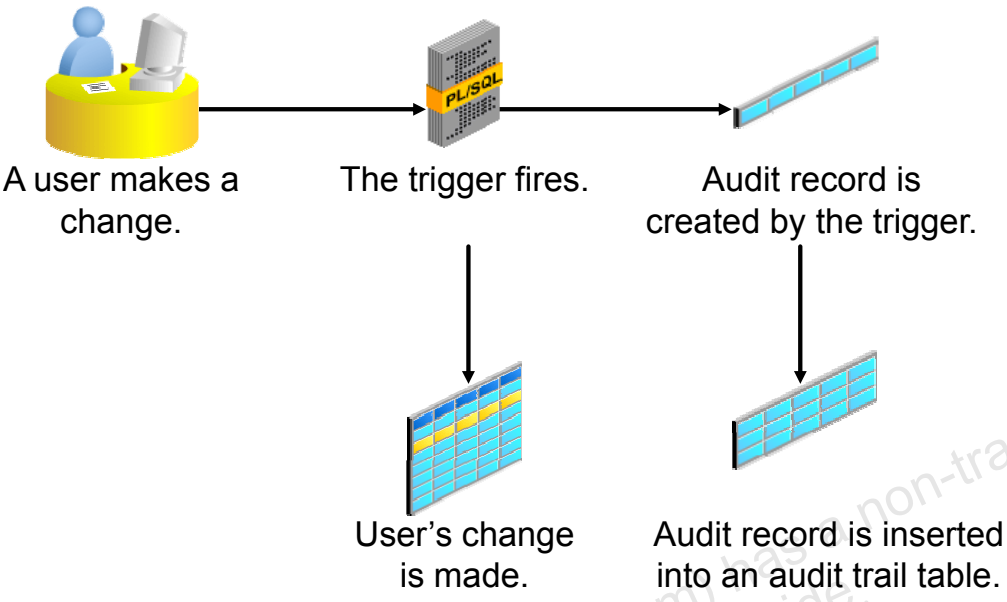
- Immediate-Write mode: The audit records are written immediately.

```
SQL> EXEC DBMS_AUDIT_MGMT.SET_AUDIT_TRAIL_PROPERTY(-
  2       DBMS_AUDIT_MGMT.AUDIT_TRAIL_UNIFIED, -
  3       DBMS_AUDIT_MGMT.AUDIT_TRAIL_WRITE_MODE, -
  4       DBMS_AUDIT_MGMT.AUDIT_TRAIL_IMMEDIATE_WRITE);
```

- Queued-Write mode: The content of the SGA queues is periodically written to disk. This is the default mode.

```
SQL> EXEC DBMS_AUDIT_MGMT.SET_AUDIT_TRAIL_PROPERTY(-
  2       DBMS_AUDIT_MGMT.AUDIT_TRAIL_UNIFIED, -
  3       DBMS_AUDIT_MGMT.AUDIT_TRAIL_WRITE_MODE, -
  4       DBMS_AUDIT_MGMT.AUDIT_TRAIL_QUEUED_WRITE);
```

Value-Based Auditing



Value-based auditing leverages database triggers (event-driven PL/SQL constructs) to capture changed values in objects.

When a user inserts, updates, or deletes data from a table with the appropriate trigger attached, the trigger works in the background to copy audit information to a table that is designed to contain the audit information. Value-based auditing tends to degrade performance more than standard database auditing because the audit trigger code must be executed each time the insert, update, or delete operation occurs. The degree of degradation depends on the efficiency of the trigger code. Value-based auditing must be used only in situations in which the information captured by standard database auditing is insufficient.

Value-based auditing is implemented by user or third-party code. The Oracle database provides the PL/SQL constructs to allow value-based audit systems to be built.

The key to value-based auditing is the audit trigger, which is simply a PL/SQL trigger that is constructed to capture audit information.

Example of a typical audit trigger:

```
CREATE OR REPLACE TRIGGER system.hrsalary_audit
      AFTER UPDATE OF salary
      ON hr.employees
      REFERENCING NEW AS NEW OLD AS OLD
      FOR EACH ROW
  BEGIN
   IF :old.salary != :new.salary THEN
       INSERT INTO system.audit_employees
       VALUES (sys_context('userenv','os_user'), sysdate,
       sys_context('userenv','ip_address'),
       :new.employee_id ||
       ' salary changed from '||:old.salary||
       ' to '||:new.salary);
  END IF;
  END;
/
```

This trigger focuses auditing to capture changes to the salary column of the `hr.employees` table. When a row is updated, the trigger checks the salary column. If the old salary is not equal to the new salary, the trigger inserts an audit record into the `audit_employees` table (created via a separate operation in the `SYSTEM` schema). The audit record includes the username, the IP address from which the change is made, the primary key identifying which record is changed, and the actual salary values that are changed.

Database triggers can also be used to capture information about user connections in cases where standard database auditing does not gather sufficient data. With login triggers, the administrator can capture data that identifies the user who is connecting to the database. Examples include the following:

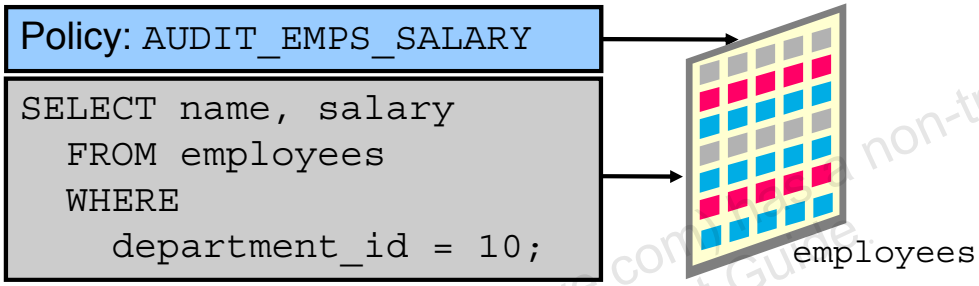
- IP address of the person logging in
- First 48 characters of the program name that is used to connect to the instance
- Terminal name that is used to connect to the instance

For a complete list of user parameters, see the section titled "`SYS_CONTEXT`" in the *Oracle Database SQL Reference*.

Value-based triggers have been superseded in many cases by the fine-grained auditing (FGA) feature.

Fine-Grained Auditing

- Monitors data access on the basis of content
- Audits SELECT, INSERT, UPDATE, DELETE, and MERGE
- Can be linked to one or more columns in a table or view
- May execute a procedure
- Is administered with the DBMS_FGA package



Database auditing records the fact that an operation has occurred but does not capture information about the statement that caused the operation. Fine-grained auditing (FGA) extends that capability to enable the capture of actual SQL statements that query or manipulate data.

FGA also allows auditing to be more narrowly focused than standard or value-based database auditing.

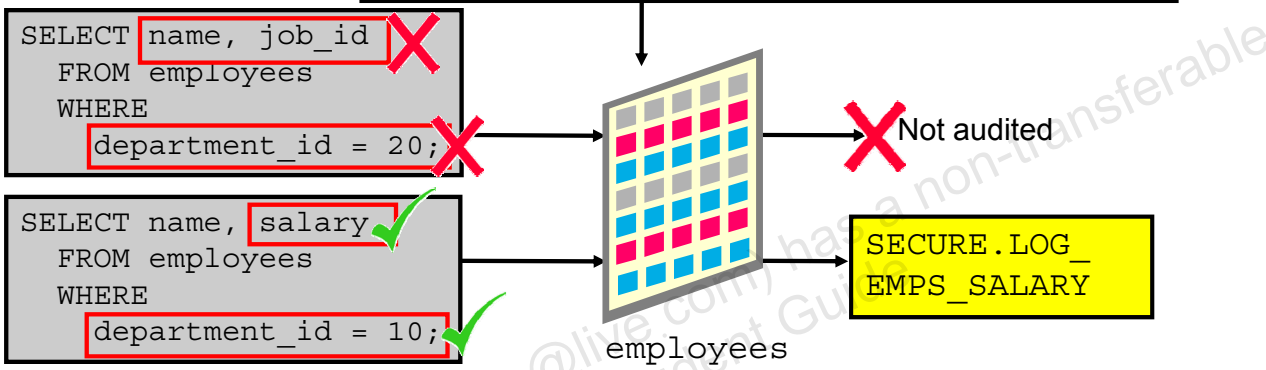
FGA options can be focused by individual columns in a table or view, and can even be conditional so that audits are captured only if certain administrator-defined specifications are met. More than one relevant column is supported for an FGA policy. By default, if any one of these columns is present in the SQL statement, it is audited. DBMS_FGA.ALL_COLUMNS and DBMS_FGA.ANY_COLUMNS are provided to audit on the basis of whether any or all of the relevant columns are used in the statement.

Use the DBMS_FGA PL/SQL package to create an audit policy on the target table or view. If any of the rows returned from a query block match the audited column and the specified audit condition, an audit event causes an audit record to be created and stored in the audit trail. As an option, the audit event can also execute a procedure. FGA automatically focuses auditing at the statement level. A SELECT statement that returns thousands of rows thus generates only one audit record.

FGA Policy

- Defines:
  - Audit criteria
  - Audit action
- Is created with DBMS_FGA .ADD_POLICY

```
dbms_fga.add_policy (
 object_schema   => 'HR',
 object_name     => 'EMPLOYEES',
 policy_name => 'audit_emps_salary',
 audit_condition=> 'department_id=10',
 audit_column    => 'SALARY,COMMISSION_PCT',
 handler_schema  => 'secure',
 handler_module  => 'log_emps_salary',
 enable          => TRUE,
 statement_types => 'SELECT,UPDATE');
```



ORACLE

The example in the slide shows the creation of a fine-grained auditing policy with the DBMS_FGA.ADD_POLICY procedure, which accepts the following arguments.

Policy Name

You assign each FGA policy a name when you create it. The example in the slide names the policy AUDIT_EMPS_SALARY by using the following argument:

```
policy_name => 'audit_emps_salary'
```

Audit Condition

The audit condition is a SQL predicate that defines when the audit event must fire. In the slide example, all rows in department 10 are audited by using the following condition argument:

```
audit_condition => 'department_id = 10'
```

Note: Fine-grained auditing looks at the result set of the query, so with the FGA policy shown in the slide, queries that returns rows matching the policy specifications will cause an audit record to be created. For example, in the query "`select * from employees`", all rows including those having “10” in DEPARTMENT_ID may be returned, so an audit row is created.

Audit Column

The audit column defines the data that is being audited. An audit event occurs if this column is included in the `SELECT` statement or if the audit condition allows the selection. The example in the slide audits two columns by using the following argument:

```
audit_column => 'SALARY,COMMISION_PCT'
```

This argument is optional. If it is not specified, only the `AUDIT_CONDITION` argument determines whether an audit event must occur.

Object

The object is the table or view that is being audited. It is passed as two arguments:

- The schema that contains the object
- The name of the object

The example in the slide audits the `hr.employees` table by using the following arguments:

```
object_schema => 'hr'
object_name => 'employees'
```

Handler

An optional event handler is a PL/SQL procedure that defines additional actions that must be taken during auditing. For example, the event handler can send an alert page to the administrator. If it is not defined, an audit event entry is inserted into the audit trail. If an audit event handler is defined, the audit entry is inserted into the audit trail and the audit event handler is executed.

The audit event entry includes the FGA policy that caused the event, the user executing the SQL statement, and the SQL statement and its bind variables.

The event handler is passed as two arguments:

- The schema that contains the PL/SQL program unit
- The name of the PL/SQL program unit

The example in the slide executes the `SECURE.LOG_EMPS_SALARY` procedure by using the following arguments:

```
handler_schema => 'secure'
handler_module => 'log_emps_salary'
```

By default, the audit trail always writes the SQL text and SQL bind information to LOBs. The default can be changed (for example, if the system would suffer performance degradation).

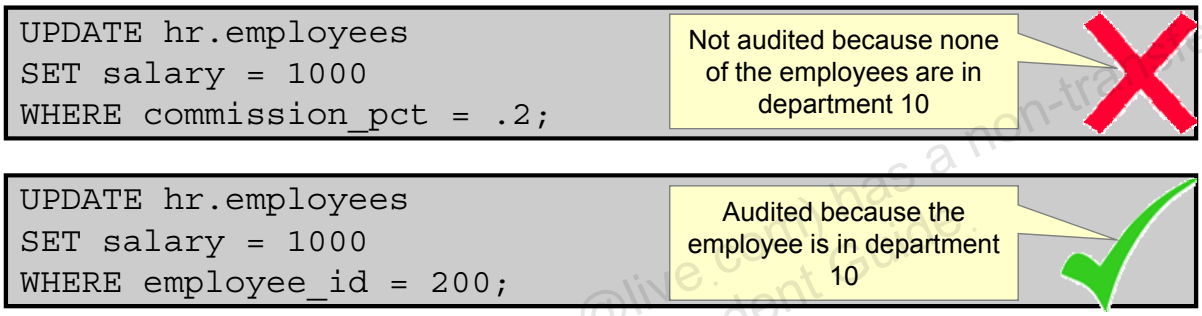
Status

The status indicates whether the FGA policy is enabled. In the slide example, the following argument enables the policy:

```
enable => TRUE
```

Audited DML Statement: Considerations

- Records are audited if the FGA predicate is satisfied and the relevant columns are referenced.
- `DELETE` statements are audited regardless of columns specified.
- `MERGE` statements are audited with the underlying `INSERT`, `UPDATE`, and `DELETE` generated statements.



With an FGA policy defined for DML statements, a DML statement is audited if the data rows (both new and old) that are being manipulated meet the policy predicate criteria.

However, if relevant columns are also specified in the policy definition, the statement is audited when the data meets the FGA policy predicate and the statement references the relevant columns defined.

For `DELETE` statements, specifying relevant columns during policy definition is not useful because all columns in a table are touched by a `DELETE` statement. Therefore, a `DELETE` statement is always audited regardless of the relevant columns.

`MERGE` statements are supported by FGA. The underlying `INSERT`, `UPDATE`, and `DELETE` statements are audited if they meet the defined `INSERT`, `UPDATE`, or `DELETE` FGA policies.

Using the previously defined FGA policy, the first statement is not audited whereas the second one is. None of the employees in department 10 receive a commission, but `employee_id=200` specifies an employee in department 10.

FGA Guidelines

- To audit all rows, use a `null` audit condition.
- To audit all columns, use a `null` audit column.
- Policy names must be unique.
- The audited table or view must already exist when you create the policy.
- If the audit condition syntax is invalid, an `ORA-28112` error is raised when the audited object is accessed.
- If the audited column does not exist in the table, no rows are audited.
- If the event handler does not exist, no error is returned and the audit record is still created.

For the `SELECT` statements, FGA captures the statement itself and not the actual rows. However, when FGA is combined with Flashback Query, the rows can be reconstructed as they existed at that point in time.

For more details about the `DBMS_FGA` package, see the *Oracle Database PL/SQL Packages and Types Reference*.

Archiving and Purging the Audit Trail

- Periodically archive and purge the audit trail to prevent it from growing too large.
- Create an archive by:
  - Copying audit trail records to a database table
  - Using Oracle Audit Vault
- Purge the audit trail by:
  - Creating and scheduling a purge job to run at a specified time by using the `DBMS_AUDIT_MGMT.CREATE_PURGE_JOB` PL/SQL procedure
  - Manually by using the `DBMS_AUDIT_MGMT.CLEAN_AUDIT_TRAIL` PL/SQL procedure

Basic maintenance of the audit trail includes reviewing the audit records and removing older records. The audit trail can grow to fill the available storage if not reviewed periodically and archived and purged. If the database audit trail fills the tablespace, audited actions do not complete.

Purging Audit Trail Records

- Schedule an automatic purge job:

```
DBMS_AUDIT_MGMT.CREATE_PURGE_JOB
(AUDIT_TRAIL_TYPE=> DBMS_AUDIT_MGMT.AUDIT_TRAIL_UNIFIED,
 AUDIT_TRAIL_PURGE_INTERVAL => 12,
 AUDIT_TRAIL_PURGE_NAME => 'Audit_Trail_PJ',
 USE_LAST_ARCH_TIMESTAMP => TRUE,
 CONTAINER => DBMS_AUDIT_MGMT.CONTAINER_CURRENT);
```

- Manually purge the audit records:

```
DBMS_AUDIT_MGMT.CLEAN_AUDIT_TRAIL(
   AUDIT_TRAIL_TYPE  => DBMS_AUDIT_MGMT.AUDIT_TRAIL_UNIFIED)
```


To perform the audit trail purge tasks, you use the DBMS_AUDIT_MGMT package. You must have the AUDIT_ADMIN role to use the package. By mandate, Oracle Database audits all executions of the DBMS_AUDIT_MGMT package procedures.

The DBMS_AUDIT_MGMT.CREATE_PURGE_JOB procedure includes the CONTAINER attribute. Setting CONTAINER to CONTAINER_CURRENT deletes audit trail records for the current pluggable database. Setting CONTAINER to CONTAINER_ALL deletes audit trail records for all pluggable databases, creating a job in the root, and the invocation of this job will invoke cleanup in all the PDBs.

USE_LAST_ARCH_TIMESTAMP specifies whether the last archived time stamp should be used for determining the records that should be deleted. Setting USE_LAST_ARCH_TIMESTAMP to TRUE indicates that only audit records created before the last archive time stamp should be deleted. A value of FALSE indicates that all audit records should be deleted. The default value is TRUE. Oracle recommends using this value because this helps guard against inadvertent deletion of records.

You can automate the cleanup process by creating and scheduling a cleanup purge job, or you can manually run a cleanup purge job. If you manually run cleanup purge jobs, use the DBMS_AUDIT_MGMT.CLEAN_AUDIT_TRAIL procedure with a new type value of DBMS_AUDIT_MGMT.AUDIT_TRAIL_UNIFIED.

Quiz

Top-level statements performed before the database opens by administrative users such as `SYS` and `SYSDBA` are mandatorily audited.

a. True
b. False

Answer: a

Summary

In this lesson, you should have learned how to:

- Describe DBA responsibilities for security and auditing
- Enable unified auditing
- Create unified audit policies
- Maintain the audit trail

Practice 14

- 14-1: Enabling Unified Auditing
- 14-2: Creating Audit Users
- 14-3: Creating an Audit Policy