31

Value-Based Auditing with Fine-Grained Audit Policies

Fine-grained auditing enables you to perform value-based auditing to audit access to certain rows based on values in specific columns.

- Overview of Fine-Grained Auditing
  Before you create fine-grained audit policies, you should understand the overall concepts how of fine-grained auditing works.
- Creating Fine-Grained Audit Policies
  The `DBMS_FGA.ADD_POLICY` procedure creates a fine-grained audit policy.
- Managing Fine-Grained Audit Policies
  After you create a fine-grained audit policy, you can alter or drop it.
- Tutorial: Adding an Email Alert to a Fine-Grained Audit Policy
  This tutorial demonstrates how to create a fine-grained audit policy that generates an email alert when users violate the policy.
- Fine-Grained Audit Policy Data Dictionary Views
  You can query data dictionary and dynamic views to find detailed auditing information about fine-grained audit policies.

Related Topics

- Value-Based Fine-Grained Audit Activities
  Use fine-grained auditing if you want to perform value-based auditing to audit access to certain rows based on values in specific columns or if you want to integrate with event handlers within the Oracle database.

31.1 Overview of Fine-Grained Auditing

Before you create fine-grained audit policies, you should understand the overall concepts how of fine-grained auditing works.

- About Fine-Grained Auditing
  Oracle Database enables you to create customized audit policies using fine-grained auditing (FGA), which is available in Oracle Database Enterprise Edition.
- Where Are Fine-Grained Audit Records Stored?
  Fine-grained auditing records are stored in the unified audit trail, which you can view by querying the `UNIFIED_AUDIT_TRAIL` data dictionary view.
- Who Can Perform Fine-Grained Auditing?
  Oracle provides roles for privileges needed to create fine-grained audit policies and to view and analyze fine-grained audit policy data.
- Fine-Grained Auditing on Tables or Views That Have Oracle VPD Policies
  The audit trail captures the VPD predicate for fine-grained audited tables or views that are included in an Oracle VPD policy.

- Fine-Grained Auditing in a Multitenant Environment
  You can create fine-grained audit policies in the CDB root, application root, CDB PDBs, and application PDBs.
- Fine-Grained Audit Policies with Editions
  You can create `DBMS_FGA` policies for use in an editions environment.

Related Topics

- Value-Based Fine-Grained Audit Activities
  Use fine-grained auditing if you want to perform value-based auditing to audit access to certain rows based on values in specific columns or if you want to integrate with event handlers within the Oracle database.

31.1.1 About Fine-Grained Auditing

Oracle Database enables you to create customized audit policies using fine-grained auditing (FGA), which is available in Oracle Database Enterprise Edition.

Use fine-grained auditing if you want to perform value-based auditing to audit access to certain rows based on values in specific columns or if you want to integrate with event handlers within the Oracle database.

Fine-grained auditing enables you to monitor data access based on content of the column values returned. For instance, with fine-grained auditing, you can audit access to a sensitive column such as `SALARY` in the `EMPLOYEES` table only when record values with `SALARY >1500` are retrieved by the query. Fine-grained audit policies also enable you to specify an event handler. Event handlers are PL/SQL functions that Oracle Database calls when an audit condition is triggered. When a SQL query satisfies the fine-grained audit policy conditions (that is, relevant columns and specific data values being accessed), Oracle Database invokes the event handler, which in turn can be configured to message to a database administrator or it can trigger a security alert in an external system. This speeds up the detection of a security violation and enables administrators to respond to the problem sooner.

Two key use-cases where you will want to consider fine-grained audit policies in addition to unified audit policies are:

- When you want to audit access to specific security-relevant columns, and their sensitive data values, such as salaries or Social Security numbers
- Raise alerts on possible security breaches

Fine-grained auditing enables you to monitor data access based on content. It provides granular auditing of queries, and `INSERT`, `UPDATE`, and `DELETE` operations. Some sample instances where you might consider fine-grained auditing includes the following:

- Accessing a table between 9 p.m. and 6 a.m. or on Saturday and Sunday
- Using an IP address from outside the corporate network
- Modifying a sensitive data value above an expected threshold

Fine-grained audit policies are based on simple, user-defined SQL predicates on table objects that act as conditions for selective auditing. The SQL statement is audited during fetching, whenever the policy conditions are met for a row.

Consider using fine-grained audit policies over unified audit policies if you have the following requirements:

- You need row value-based auditing. For instance, you want to audit updates to a salary column when the updated value is higher than a specified threshold, but not otherwise.

- You need to pro-actively notify administrators or other users of specific events in the Oracle database.
- You need to capture differing bind variable values in DML statement for bulk data processing operation using `BULK COLLECT` and `FORALL` in PL/SQL.

> **Note:**
>
> - Fine-grained auditing is supported only with cost-based optimization. For queries using rule-based optimization, fine-grained auditing checks before applying row filtering, which could result in an unnecessary audit event trigger.
> - Policies currently in force on an object involved in a flashback query are applied to the data returned from the specified flashback snapshot based on time or system change number (SCN).
> - If you want to use fine-grained auditing to audit data that is being directly loaded (for example, using Oracle Warehouse Builder to run DML statements), then Oracle Database transparently makes all direct loads that are performed in the database instance into conventional loads. If you want to preserve the direct loading of data, consider using unified audit policies instead.

31.1.2 Where Are Fine-Grained Audit Records Stored?

Fine-grained auditing records are stored in the unified audit trail, which you can view by querying the `UNIFIED_AUDIT_TRAIL` data dictionary view.

Administrators who have the `AUDIT_ADMIN` or `AUDIT_VIEWER` role can query `UNIFIED_AUDIT_TRAIL` data dictionary view.

The audit trail captures an audit record for each reference of a table or a view within a SQL statement. For example, if you run a `UNION` statement that references the `HR.EMPLOYEES` table twice, then an audit policy for statement generates two audit records, one for each access of the `HR.EMPLOYEES` table.

Related Topics

- Activities That Are Mandatorily Audited
  Certain security sensitive database activities are always audited and such audit configurations cannot be disabled.
- *Oracle Database PL/SQL Packages and Types Reference*

31.1.3 Who Can Perform Fine-Grained Auditing?

Oracle provides roles for privileges needed to create fine-grained audit policies and to view and analyze fine-grained audit policy data.

The fine-grained audit privileges are as follows:

- To create and administer fine-grained audit policies, you must be granted the `AUDIT_ADMIN` role or the `EXECUTE` privilege on the `DBMS_FGA` package. You must also be granted the `ADMINISTER FINE GRAINED AUDIT POLICY` system privilege to administer other schemas than your own schemas. (A user does not need this privilege to administer fine-grained audit policies in their own schema.) To grant the `ADMINISTER FINE GRAINED AUDIT POLICY` privilege:

  - Syntax of the `ADMINISTER FINE GRAINED AUDIT POLICY` privilege grant if the fine-grained audit policy is to apply to all non-`SYS` schemas across the database:

    ```
    GRANT ADMINISTER FINE GRAINED AUDIT POLICY TO grantee;
    ```

  - Syntax of the `ADMINISTER FINE GRAINED AUDIT POLICY` privilege grant if the fine-grained audit policy is to be restricted to a specific schema:

    ```
    GRANT ADMINISTER FINE GRAINED AUDIT POLICY ON SCHEMA schema TO grantee;
    ```

- To view and analyze fine-grained audit data, you must be granted the `AUDIT_VIEWER` role.

The PL/SQL package is already granted to `AUDIT_ADMIN` role. As with all privileges, an administrator must only grant these roles to trusted users only. You can find the roles that user have been granted by querying the `DBA_ROLE_PRIVS` data dictionary view.

31.1.4 Fine-Grained Auditing on Tables or Views That Have Oracle VPD Policies

The audit trail captures the VPD predicate for fine-grained audited tables or views that are included in an Oracle VPD policy.

This behavior is similar to how the unified audit trail captures the VPD predicate for unified audit policies.

The audit trail also captures internal predicates from Oracle Label Security and Oracle Real Application Security policies.

You do not need to create a special audit policy to capture the VPD predicate audit records. The predicate information is automatically stored in the `RLS_INFO` column of the `UNIFIED_AUDIT_TRAIL` data dictionary view.

Related Topics

- Auditing of Oracle Virtual Private Database Predicates
  The unified audit trail automatically captures the predicates that are used in Oracle Virtual Private Database (VPD) policies.
- *Oracle Database PL/SQL Packages and Types Reference*

31.1.5 Fine-Grained Auditing in a Multitenant Environment

You can create fine-grained audit policies in the CDB root, application root, CDB PDBs, and application PDBs.

Note the following general rules about fine-grained audit policies:

- You cannot create fine-grained audit policies on `SYS` objects.
- You cannot create fine-grained audit policies, either local or application common, for extended data link objects.
- When you create a fine-grained audit policy in the CDB root, the policy cannot be applied to all PDBs. It only applies to objects within the CDB root. (In other words, there is no such thing as a common fine-grained audit policy for the CDB root.) If you want to create a fine-grained audit policy to audit a common object's access in all the PDBs, then you must explicitly create that policy in each PDB and then enable it on the common objects that is accessible in the PDB.

- When you create a fine-grained audit policy in a PDB, it applies only to objects within the PDB. You cannot create one policy for the entire multitenant environment. The policy must be specific to objects within a PDB.
- You can create application common fine-grained audit policies only if you are connected to the application root and only within the `BEGIN`/`END` block. If you are connected to the application root and create the fine-grained audit policy outside the `BEGIN`/`END` block, then the fine-grained audit policy is created in the application root.
- You cannot create application common fine-grained audit policies on local PDB objects.
- If the application common fine-grained audit policy has a handler, then this handler must be owned by either an application common user or a CDB common user.
- You can create an application fine-grained audit policy on local (PDB) objects and CDB common objects. Because the policy is local to its container, the object on which the policy is defined is audited only in the particular container where the policy is defined. For example, if you create a fine-grained audit policy in the `hr_pdb` PDB, the object for which you create this policy must exist in the `hr_pdb` PDB.
- You cannot create local fine-grained audit policies in an application PDB on object linked and extended data link objects. On metadata-linked objects are allowed in the fine-grained audit policy.
- Application root local policies are allowed for all application common objects.
- When you create a fine-grained audit policy as a common audit policy in an application root, it will be effective in each PDB that belongs to this application root. Therefore, any access to the application common object and CDB common object (on which the application common fine-grained audit policy is defined) from the application PDB is audited in the fine-grained audit trail in that application PDB.
- When you create scripts for application install, upgrade, patch, or uninstall operations, you can include SQL statements within the `ALTER PLUGGABLE DATABASE` *`app_name`* `BEGIN INSTALL` and `ALTER PLUGGABLE DATABASE` *`app_name`* `END INSTALL` blocks to perform various operations. You can include fine-grained audit policy statements only within these blocks.
- You can only enable, disable, or drop application common fine-grained audit policies from the application root, and from within a `ALTER PLUGGABLE DATABASE` *`app_name`* `BEGIN INSTALL` and `ALTER PLUGGABLE DATABASE` *`app_name`* `END INSTALL` block in a script.

31.1.6 Fine-Grained Audit Policies with Editions

You can create `DBMS_FGA` policies for use in an editions environment.

Note the following:

- You can prepare an application for edition-based redefinition, and cover each table that the application uses with an editioning view. If you do this, then you must move the fine-grained audit polices that protect these tables to the editioning view. You can find information about the currently configured editions by querying the `DBA_EDITIONS` data dictionary view. To find information about fine-grained audit policies, query `DBA_AUDIT_POLICIES`.
- If you plan to use the `DBMS_FGA` package policy across different editions, then you can control the results of the policy: whether the results are uniform across all editions, or specific to the edition in which the policy is used.

Related Topics

- How Editions Affects the Results of a Global Application Context PL/SQL Package
  Global application context packages, Oracle Virtual Private Database packages, and fine-grained audit policies can be used across multiple editions.

31.2 Creating Fine-Grained Audit Policies

The `DBMS_FGA.ADD_POLICY` procedure creates a fine-grained audit policy.

- About Creating a Fine-Grained Audit Policy
  You can create and manage fine-grained audit policies by using the `DBMS_FGA` PL/SQL package.
- Syntax for Creating a Fine-Grained Audit Policy
  The `DBMS_FGA.ADD_POLICY` procedure includes many settings, such as the ability to use a handler for complex auditing.
- Example: Using DBMS_FGA.ADD_POLICY to Create a Fine-Grained Audit Policy
  The `DBMS_FGA.ADD_POLICY` procedure can create a fine-grained audit policy using multiple statement types.
- Audits of Specific Columns and Rows
  You can do value-based auditing to audit access to certain rows based on values in specific columns.

31.2.1 About Creating a Fine-Grained Audit Policy

You can create and manage fine-grained audit policies by using the `DBMS_FGA` PL/SQL package.

Consider the following when you create fine-grained audit policies:

- The `DBMS_FGA` PL/SQL package enables you to add all combinations of the following statements into one policy:
  - `SELECT`
  - `INSERT`
  - `UPDATE`
  - `DELETE`
- For `MERGE` statements:
  - You can audit `MERGE` statements by configuring fine-grained access on the underlying actions of `INSERT` and `UPDATE`.
  - Only one record is generated for each policy for successful `MERGE` operations.

If you plan to create a materialized view on the base table on which you want to create a fine-grained audit policy, then you must create the fine-grained audit policy on the base table *before* you create the materialized view on the same table. Otherwise, any refresh operations on the materialized view will fail with an `ORA-12008: error in materialized view refresh path` error.

When you create a fine-grained audit policy, be aware that sensitive data, such as credit card information, can be recorded in clear text.

To administer fine-grained audit policies, you must have be granted the `AUDIT_ADMIN` role. Note also that the `EXECUTE` privilege for the `DBMS_FGA` package is mandatorily audited.

The audit policy is bound to the table for which you created it. This simplifies the management of audit policies because the policy only needs to be changed once in the database, not in each application. In addition, no matter how a user connects to the database—from an application, a Web interface, or through SQL*Plus or Oracle SQL Developer—Oracle Database records any actions that affect the policy.

If any rows returned from a query match the audit condition that you define, then Oracle Database inserts an audit entry into the unified audit trail. This entry excludes all the information that is reported in the regular audit trail. In other words, only one row of audit information is inserted into the audit trail for every fine-grained audit policy that evaluates to true.

The `DBMS_FGA.ADD_POLICY` procedure creates an audit policy using the supplied predicate as the audit condition.

By default, Oracle Database runs the policy predicate with the privileges of the user who owns 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. The fine grained policy is only created in the local PDB.

You cannot modify a fine-grained audit policy after you have created it. If you must modify the policy, then drop and recreate it.

You can find information about a fine-grained audit policy by querying the `ALL_AUDIT_POLICIES`, `DBA_AUDIT_POLICIES`, and `USER_AUDIT_POLICIES` views. The `UNIFIED_AUDIT_TRAIL` view contains a column entitled `FGA_POLICY_NAME`, which you can use to filter out rows that were generated using a specific fine-grained audit policy.

Related Topics

- *Oracle Database PL/SQL Packages and Types Reference*

31.2.2 Syntax for Creating a Fine-Grained Audit Policy

The `DBMS_FGA.ADD_POLICY` procedure includes many settings, such as the ability to use a handler for complex auditing.

The `DBMS_FGA.ADD_POLICY` procedure syntax is as follows:

```
DBMS_FGA.ADD_POLICY(
   object_schema      IN  VARCHAR2 DEFAULT NULL
   object_name        IN  VARCHAR2,
   policy_name        IN  VARCHAR2,
   audit_condition    IN  VARCHAR2 DEFAULT NULL,
   audit_column       IN  VARCHAR2 DEFAULT NULL
   handler_schema     IN  VARCHAR2 DEFAULT NULL,
   handler_module     IN  VARCHAR2 DEFAULT NULL,
   enable             IN  BOOLEAN DEFAULT TRUE,
   statement_types    IN  VARCHAR2 DEFAULT SELECT,
   audit_trail        IN  BINARY_INTEGER DEFAULT NULL,
   audit_column_opts  IN  BINARY_INTEGER DEFAULT ANY_COLUMNS,
   policy_owner       IN  VARCHAR2 DEFAULT NULL);
```

In this specification:

- `object_schema` specifies the schema of the object to be audited. (If `NULL`, the current log-on user schema is assumed.)
- `object_name` specifies the name of the object to be audited.
- `policy_name` specifies the name of the policy to be created. Ensure that this name is unique.
- `audit_condition` specifies a Boolean condition in a row. `NULL` is allowed and acts as `TRUE`. If you specify `NULL` or no audit condition, then any action on a table with that policy creates an audit record, whether or not rows are returned.

  Follow these guidelines:

  – Do not include functions, which run the auditable statement on the same base table, in the `audit_condition` setting. For example, suppose you create a function that runs an `INSERT` statement on the `HR.EMPLOYEES` table. The policy's `audit_condition` contains this function and it is for `INSERT` statements (as set by `statement_types`). When the policy is used, the function runs recursively until the system has run out of memory. This can raise the error `ORA-1000: maximum open cursors exceeded` or `ORA-00036: maximum number of recursive SQL levels (50) exceeded`.

  – Do not issue the `DBMS_FGA.ENABLE_POLICY` or `DBMS_FGA.DISABLE_POLICY` statement from a function in a policy's condition.
- `audit_column` specifies one or more columns to audit, including hidden columns. If set to `NULL` or omitted, all columns are audited. These can include Oracle Label Security hidden columns or object type columns. The default, `NULL`, causes audit if any column is accessed or affected.
- `handler_schema`: If an alert is used to trigger a response when the policy is violated, specifies the name of the schema that contains the event handler. The default, `NULL`, uses the current schema.
- `handler_module` specifies the name of the event handler. Include the package the event handler is in. This function is invoked only after the first row that matches the audit condition in the query is processed.

  Follow these guidelines:

  – Do not create recursive fine-grained audit handlers. For example, suppose you create a handler that runs an `INSERT` statement on the `HR.EMPLOYEES` table. The policy that is associated with this handler is for `INSERT` statements (as set by the `statement_types` parameter). When the policy is used, the handler runs recursively until the system has run out of memory. This can raise the error `ORA-1000: maximum open cursors exceeded` or `ORA-00036: maximum number of recursive SQL levels (50) exceeded`.

  – Do not issue the `DBMS_FGA.ENABLE_POLICY` or `DBMS_FGA.DISABLE_POLICY` statement from a policy handler. Doing so can raise the `ORA-28144: Failed to execute fine-grained audit handler` error.
- `enable` enables or disables the policy using true or false. If omitted, the policy is enabled. The default is `TRUE`.
- `statement_types`: Specifies the SQL statements to be audited: `INSERT`, `UPDATE`, `DELETE`, or `SELECT` only. If you want to audit a `MERGE` operation, then set `statement_types` to `'INSERT,UPDATE'`. The default is `SELECT`.
- `audit_trail`: If you have migrated to unified auditing, then Oracle Database ignores this parameter and writes the audit records immediately to the unified audit trail. Starting in Oracle Database 23ai, traditional auditing is desupported, so the `audit_trail` is ignored.

- `audit_column_opts`: If you specify more than one column in the `audit_column` parameter, then this parameter determines whether to audit all or specific columns.
- `policy_owner` is the user who owns the fine-grained auditing policy. However, this setting is not a user-supplied argument. The Oracle Data Pump client uses this setting internally to recreate the fine-grained audit policies appropriately.

Related Topics

- Audits of Specific Columns and Rows
  You can do value-based auditing to audit access to certain rows based on values in specific columns.
- *Oracle Database PL/SQL Packages and Types Reference*

31.2.3 Example: Using DBMS_FGA.ADD_POLICY to Create a Fine-Grained Audit Policy

The `DBMS_FGA.ADD_POLICY` procedure can create a fine-grained audit policy using multiple statement types.

Example 31-1 shows how to audit statements `INSERT`, `UPDATE`, `DELETE`, and `SELECT` on table `HR.EMPLOYEES`.

Note that this example omits the `audit_column_opts` parameter, because it is not a mandatory parameter.

Example 31-1 Using DBMS_FGA.ADD_POLICY to Create a Fine-Grained Audit Policy

```
BEGIN
  DBMS_FGA.ADD_POLICY(
   object_schema      => 'HR',
   object_name        => 'EMPLOYEES',
   policy_name        => 'chk_hr_employees',
   audit_column       => 'SALARY',
   enable             =>  TRUE,
   statement_types    => 'INSERT, UPDATE, SELECT, DELETE');
END;
/
```

After you create the policy, if you query the `DBA_AUDIT_POLICIES` view, you will find the new policy listed:

```
SELECT POLICY_NAME FROM DBA_AUDIT_POLICIES;

POLICY_NAME
------------------------------
CHK_HR_EMPLOYEES
```

Afterwards, any of the following SQL statements log an audit event record.

```
SELECT COUNT(*) FROM HR.EMPLOYEES WHERE COMMISSION_PCT = 20 AND SALARY > 4500;

SELECT SALARY FROM HR.EMPLOYEES WHERE DEPARTMENT_ID = 50;

DELETE FROM HR.EMPLOYEES WHERE SALARY > 1000000;
```

31.2.4 Audits of Specific Columns and Rows

You can do value-based auditing to audit access to certain rows based on values in specific columns.

To accomplish this, use the `audit_column` parameter of the `DBMS_FGA.ADD_POLICY` procedure to specify one or more sensitive columns. Use the `audit_condition` boolean parameter to audit data in specific rows. Consider using unified audit policy if you do not have a need to do value-based auditing.

The following settings enable you to perform an audit if anyone in Department 50 (`DEPARTMENT_ID = 50`) tries to access the `SALARY` and `COMMISSION_PCT` columns.

```
audit_condition    => 'DEPARTMENT_ID = 50',
audit_column       => 'SALARY,COMMISSION_PCT,'
```

As you can see, this feature is enormously beneficial. It not only enables you to pinpoint particularly important types of data to audit, but it provides increased protection for columns that contain sensitive data, such as Social Security numbers, salaries, patient diagnoses, and so on.

If the `audit_column` lists more than one column, then you can use the `audit_column_opts` parameter to specify whether a statement is audited when the query references *any* column specified in the `audit_column` parameter or only when *all* columns are referenced. For example:

```
audit_column_opts   => DBMS_FGA.ANY_COLUMNS,

audit_column_opts   => DBMS_FGA.ALL_COLUMNS,
```

If you do not specify a relevant column, then auditing applies to all columns.

Related Topics

- Unified Auditing with Configurable Conditions
  You can use the `CREATE AUDIT POLICY` statement to create conditions for a unified audit policy.
- *Oracle Database PL/SQL Packages and Types Reference*

31.3 Managing Fine-Grained Audit Policies

After you create a fine-grained audit policy, you can alter or drop it.

- Enabling a Fine-Grained Audit Policy
  The `DBMS_FGA.ENABLE_POLICY` procedure enables a fine-grained audit policy.
- Disabling a Fine-Grained Audit Policy
  The `DBMS_FGA.DISABLE_POLICY` procedure disables a fine-grained audit policy.
- Dropping a Fine-Grained Audit Policy
  The `DBMS_FGA.DROP_POLICY` procedure drops a fine-grained audit policy.

31.3.1 Enabling a Fine-Grained Audit Policy

The `DBMS_FGA.ENABLE_POLICY` procedure enables a fine-grained audit policy.

- Use the following syntax to enable a fine-grained audit policy:

```
DBMS_FGA.ENABLE_POLICY(
   object_schema  VARCHAR2,
   object_name    VARCHAR2,
   policy_name    VARCHAR2,
   enable         BOOLEAN);
```

For example, to reenable the `chk_hr_emp` policy by using the `DBMS_FGA.ENABLE_POLICY` procedure

```
BEGIN
 DBMS_FGA.ENABLE_POLICY(
  object_schema        => 'HR',
  object_name          => 'EMPLOYEES',
  policy_name          => 'chk_hr_employees',
  enable               => TRUE);
END;
/
```

Related Topics

- *Oracle Database PL/SQL Packages and Types Reference*

31.3.2 Disabling a Fine-Grained Audit Policy

The `DBMS_FGA.DISABLE_POLICY` procedure disables a fine-grained audit policy.

- Use the following syntax to disable a fine-grained audit policy:

```
DBMS_FGA.DISABLE_POLICY(
   object_schema  VARCHAR2,
   object_name    VARCHAR2,
   policy_name    VARCHAR2);
```

For example:

```
BEGIN
 DBMS_FGA.DISABLE_POLICY(
  object_schema        => 'HR',
  object_name          => 'EMPLOYEES',
  policy_name          => 'chk_hr_employees');
END;
/
```

Related Topics

- *Oracle Database PL/SQL Packages and Types Reference*

31.3.3 Dropping a Fine-Grained Audit Policy

The `DBMS_FGA.DROP_POLICY` procedure drops a fine-grained audit policy.

Oracle Database automatically drops the audit policy if you remove the object specified in the `object_name` parameter of the `DBMS_FGA.ADD_POLICY` procedure, or if you drop the user who created the audit policy.

- Use the following syntax to drop a fine-grained audit policy:

```
DBMS_FGA.DROP_POLICY(
   object_schema  VARCHAR2,
   object_name    VARCHAR2,
   policy_name    IVARCHAR2);
```

For example:

```
BEGIN
 DBMS_FGA.DROP_POLICY(
  object_schema      => 'HR',
  object_name        => 'EMPLOYEES',
  policy_name        => 'chk_hr_employees');
END;
/
```

Related Topics

- *Oracle Database PL/SQL Packages and Types Reference*

31.4 Tutorial: Adding an Email Alert to a Fine-Grained Audit Policy

This tutorial demonstrates how to create a fine-grained audit policy that generates an email alert when users violate the policy.

- About This Tutorial
  This tutorial shows how you can add an email alert to a fine-grained audit policy that goes into effect when a user (or an intruder) violates the policy.
- Step 1: Install and Configure the UTL_MAIL PL/SQL Package
  The `UTL_MAIL` PL/SQL manages email that includes commonly used email features, such as attachments, CC, and BCC.
- Step 2: Create User Accounts
  You must create an administrative account and an auditor user.
- Step 3: Configure an Access Control List File for Network Services
  An access control list (ACL) file can be used to enable fine-grained access to external network services.
- Step 4: Create the Email Security Alert PL/SQL Procedure
  The email security alert PL/SQL procedure generates a message describing the violation and then sends this message to the appropriate users.
- Step 5: Create and Test the Fine-Grained Audit Policy Settings
  The fine-grained audit policy will trigger the alert when the policy is violated.
- Step 6: Test the Alert
  With the components in place, you are ready to test the alert.
- Step 7: Remove the Components of This Tutorial
  If you no longer need the components of this tutorial, then you can remove them.

31.4.1 About This Tutorial

This tutorial shows how you can add an email alert to a fine-grained audit policy that goes into effect when a user (or an intruder) violates the policy.

> **Note:**
>
> - To complete this tutorial, you must use a database that has an SMTP server.
> - This tutorial applies to the current PDB only.

To add an email alert to a fine-grained audit policy, you first must create a procedure that generates the alert, and then use the following `DBMS_FGA.ADD_POLICY` parameters to call this function when someone violates this policy:

- `handler_schema`: The schema in which the handler event is stored
- `handler_module`: The name of the event handler

The alert can come in any form that best suits your environment: an email or pager notification, updates to a particular file or table, and so on. Creating alerts also helps to meet certain compliance regulations, such as California Senate Bill 1386. In this tutorial, you will create an email alert.

In this tutorial, you create an email alert that notifies a security administrator that a Human Resources representative is trying to select or modify salary information in the `HR.EMPLOYEES` table. The representative is permitted to make changes to this table, but to meet compliance regulations, we want to create a record of all salary selections and modifications to the table.

31.4.2 Step 1: Install and Configure the UTL_MAIL PL/SQL Package

The `UTL_MAIL` PL/SQL manages email that includes commonly used email features, such as attachments, CC, and BCC.

You must install and configure this package before you can use it. It is not installed and configured by default.

1. Log in to a PDB as user `SYS` with the `SYSDBA` administrative privilege.

   ```
   sqlplus sys@pdb_name as sysdba
   Enter password: password
   ```

   To find the available PDBs in a CDB, log in to the CDB root container and then query the `PDB_NAME` column of the `DBA_PDBS` data dictionary view. To check the current container, run the `show con_name` command.

2. Install the `UTL_MAIL` package.

   ```
   @$ORACLE_HOME/rdbms/admin/utlmail.sql
   @$ORACLE_HOME/rdbms/admin/prvtmail.plb
   ```

   The `UTL_MAIL` package enables you to manage email.

   Be aware that currently, the `UTL_MAIL` PL/SQL package does not support SSL servers.

3. Check the current value of the `SMTP_OUT_SERVER` initialization parameter, and make a note of this value so that you can restore it when you complete this tutorial.

   For example:

   ```
   SHOW PARAMETER SMTP_OUT_SERVER
   ```

   If the `SMTP_OUT_SERVER` parameter has already been set, then output similar to the following appears:

```
NAME                      TYPE             VALUE
------------------------- ---------------- --------------------------------
SMTP_OUT_SERVER           string           some_imap_server.example.com
```

4. Issue the following `ALTER SYSTEM` statement:

   ```
   ALTER SYSTEM SET SMTP_OUT_SERVER="imap_mail_server.example.com";
   ```

   Replace *imap_mail_server.example.com* with the name of your SMTP server, which you can find in the account settings in your email tool. Enclose these settings in quotation marks. For example:

   ```
   ALTER SYSTEM SET SMTP_OUT_SERVER="my_imap_server.example.com";
   ```

5. Connect as `SYS` using the `SYSOPER` privilege and then restart the database.

   ```
   CONNECT SYS@pdb_name AS SYSOPER
   Enter password: password

   SHUTDOWN IMMEDIATE
   STARTUP
   ```

6. Ensure that the `SMTP_OUT_SERVER` parameter setting is correct.

   ```
   CONNECT SYS@pdb_name AS SYSDBA
   Enter password: password

   SHOW PARAMETER SMTP_OUT_SERVER
   ```

   Output similar to the following appears:

   ```
   NAME                      TYPE             VALUE
   ------------------------- ---------------- --------------------------------
   SMTP_OUT_SERVER           string           my_imap_server.example.com
   ```

Related Topics

- *Oracle Database PL/SQL Packages and Types Reference*

31.4.3 Step 2: Create User Accounts

You must create an administrative account and an auditor user.

1. Ensure that you are connected as `SYS` using the `SYSDBA` administrative privilege, and then create the `fga_admin` user, who will create the fine-grained audit policy.

   For example:

   ```
   CONNECT SYS@pdb_name AS SYSDBA
   Enter password: password

   CREATE USER fga_admin IDENTIFIED BY password;
   GRANT CREATE SESSION, CREATE PROCEDURE, AUDIT_ADMIN TO fga_admin;
   GRANT ADMINISTER FINE GRAINED AUDIT POLICY TO fga_admin;
   GRANT EXECUTE ON UTL_TCP TO fga_admin;
   GRANT EXECUTE ON UTL_SMTP TO fga_admin;
   GRANT EXECUTE ON UTL_MAIL TO fga_admin;
   GRANT EXECUTE ON DBMS_NETWORK_ACL_ADMIN TO fga_admin;
   ```

   Replace *password* with a password that is secure.

   The `UTL_TCP`, `UTL_SMTP`, `UTL_MAIL`, and `DBMS_NETWORK_ACL_ADMIN` PL/SQL packages are used by the email security alert that you create.

2. Create the auditor user, who will check the audit trail for this policy.

```
GRANT CREATE SESSION TO fga_auditor IDENTIFIED BY password;
GRANT AUDIT_VIEWER TO fga_auditor;
```

3. Connect as user `SYSTEM`.

```
CONNECT SYSTEM@pdb_name
Enter password: password
```

4. Ensure that the `HR` schema account is unlocked and has a password. If necessary, unlock `HR` and grant this user a password.

```
SELECT USERNAME, ACCOUNT_STATUS FROM DBA_USERS WHERE USERNAME = 'HR';
```

The account status should be `OPEN`. If the `DBA_USERS` view lists user `HR` as locked and expired, then enter the following statement to unlock the `HR` account and create a new password:

```
ALTER USER HR ACCOUNT UNLOCK IDENTIFIED BY password;
```

Create a password that is secure. For greater security, do **not** give the `HR` account the same password from previous releases of Oracle Database.

5. Create a user account for Susan Mavris, who is an HR representative whose actions you will audit, and then grant this user access to the `HR.EMPLOYEES` table.

```
GRANT CREATE SESSION TO smavris IDENTIFIED BY password;
GRANT SELECT, INSERT, UPDATE, DELETE ON HR.EMPLOYEES TO SMAVRIS;
```

Related Topics

- Guidelines for Securing Passwords
  Oracle provides guidelines for securing passwords in a variety of situations.

31.4.4 Step 3: Configure an Access Control List File for Network Services

An access control list (ACL) file can be used to enable fine-grained access to external network services.

Before you can use PL/SQL network utility packages such as `UTL_MAIL`, you must configure this type of access control list (ACL) file.

1. Connect to the PDB as user `fga_admin`.

```
CONNECT fga_admin@pdb_name
Enter password: password
```

2. Configure the following access control setting and its privilege definitions.

```
BEGIN
 DBMS_NETWORK_ACL_ADMIN.APPEND_HOST_ACE(
  host       => 'SMTP_OUT_SERVER_setting',
  lower_port => 25,
  ace        =>  xs$ace_type(privilege_list => xs$name_list('smtp'),
                             principal_name => 'FGA_ADMIN',
                             principal_type => xs_acl.ptype_db));
END;
/
```

In this example:

- *SMTP_OUT_SERVER_setting*: Enter the SMTP_OUT_SERVER setting that you set for the SMTP_OUT_SERVER parameter when you installed and configured the UTL_MAIL PL/SQL package. This setting should match exactly the setting that your email tool specifies for its outgoing server.
- *lower_port*: Enter the port number that your email tool specifies for its outgoing server. Typically, this setting is 25. Enter this value for the lower_port setting. (Currently, the UTL_MAIL package does not support SSL. If your email server is an SSL server, then enter 25 for the port number, even if the email server uses a different port number.)
- ace: Define the privileges here.

Related Topics

- Step 1: Install and Configure the UTL_MAIL PL/SQL Package
  The UTL_MAIL PL/SQL manages email that includes commonly used email features, such as attachments, CC, and BCC.
- Managing Fine-Grained Access in PL/SQL Packages and Types
  Oracle Database provides PL/SQL packages and types for fine-grained access to control access to external network services and wallets.

31.4.5 Step 4: Create the Email Security Alert PL/SQL Procedure

The email security alert PL/SQL procedure generates a message describing the violation and then sends this message to the appropriate users.

- As user fga_admin, create the following procedure.

```
CREATE OR REPLACE PROCEDURE email_alert (sch varchar2, tab varchar2, pol
varchar2)
AS
msg varchar2(20000) := 'HR.EMPLOYEES table violation. The time is: ';
BEGIN
  msg := msg||TO_CHAR(SYSDATE, 'Day DD MON, YYYY HH24:MI:SS');
UTL_MAIL.SEND (
    sender      => 'youremail@example.com',
    recipients  => 'recipientemail@example.com',
    subject     => 'Table modification on HR.EMPLOYEES',
    message     => msg);
END email_alert;
/
```

In this example:

- CREATE OR REPLACE PROCEDURE ...AS: You must include a signature that describes the schema name (sch), table name (tab), and the name of the audit procedure (pol) that you will define in audit policy in the next step.
- sender and recipients: Replace *youremail@example.com* with your email address, and *recipientemail@example.com* with the email address of the person you want to receive the notification.

31.4.6 Step 5: Create and Test the Fine-Grained Audit Policy Settings

The fine-grained audit policy will trigger the alert when the policy is violated.

1. As user fga_admin, create the chk_hr_emp policy fine-grained audit policy as follows.

```
BEGIN
 DBMS_FGA.ADD_POLICY (
  object_schema      =>  'HR',
  object_name        =>  'EMPLOYEES',
  policy_name        =>  'CHK_HR_EMP',
  audit_column       =>  'SALARY',
  handler_schema     =>  'FGA_ADMIN',
  handler_module     =>  'EMAIL_ALERT',
  enable             =>   TRUE,
  statement_types    =>  'SELECT, UPDATE');
END;
/
```

2. Commit the changes you have made to the database.

```
COMMIT;
```

3. Test the settings that you have created so far.

```
EXEC email_alert ('hr', 'employees', 'chk_hr_emp');
```

SQL*Plus should display a `PL/SQL procedure successfully completed` message, and in a moment, depending on the speed of your email server, you should receive the email alert.

If you receive an `ORA-24247: network access denied by access control list (ACL)` error followed by `ORA-06512: at` *`string`*`line` *`string`* errors, then check the settings in the access control list file.

31.4.7 Step 6: Test the Alert

With the components in place, you are ready to test the alert.

1. Connect to the PDB as user `smavris`, check your salary, and give yourself a nice raise.

```
CONNECT smavris@pdb_name
Enter password: password

SELECT SALARY FROM HR.EMPLOYEES WHERE LAST_NAME = 'Mavris';

SALARY
-----------
6500

UPDATE HR.EMPLOYEES SET SALARY = 38000 WHERE LAST_NAME = 'Mavris';
```

By now, depending on the speed of your email server, you (or your recipient) should have received an email with the subject header `Table modification on HR.EMPLOYEES` notifying you of the tampering of the `HR.EMPLOYEES` table. Now all you need to do is to query the `UNIFIED_AUDIT_TRAIL` data dictionary view to find who the violator is.

2. As user `fga_auditor`, query the `UNIFIED_AUDIT_TRAIL` data dictionary view as follows:

```
CONNECT fga_auditor@pdb_name
Enter password: password

col dbusername format a20
col sql_text format a66
col audit_type format a17

SELECT DBUSERNAME, SQL_TEXT, AUDIT_TYPE
FROM UNIFIED_AUDIT_TRAIL
WHERE OBJECT_SCHEMA = 'HR' AND OBJECT_NAME = 'EMPLOYEES';
```

Output similar to the following appears:

```
DBUSERNAME  SQL_TEXT                                                        AUDIT_TYPE
----------  ------------------------------------------------------------- ----------------
SMAVRIS     UPDATE HR.EMPLOYEES SET SALARY = 38000 WHERE LAST_NAME = 'Mavris' FineGrainedAudit
```

The audit trail captures the SQL statement that Susan Mavris ran that affected the `SALARY` column in the `HR.EMPLOYEES` table. The first statement that Susan ran, in which she asked about her current salary, was not recorded because it was not affected by the audit policy. This is because Oracle Database runs the audit function as an autonomous transaction, committing only the actions of the `handler_module` setting and not any user transaction. The function has no effect on any user SQL transaction.

31.4.8 Step 7: Remove the Components of This Tutorial

If you no longer need the components of this tutorial, then you can remove them.

1. Connect to SQL*Plus as user `SYSTEM` privilege, and then drop users `fga_admin` (including the objects in the `fga_admin` schema), `fga_auditor`, and `smavris`.

   ```
   CONNECT SYSTEM@pdb_name
   Enter password: password

   DROP USER fga_admin CASCADE;
   DROP USER fga_auditor;
   DROP USER smavris;
   ```

2. Connect as user `HR` and remove the loftiness of Susan Mavris's salary.

   ```
   CONNECT HR@pdb_name
   Enter password: password

   UPDATE HR.EMPLOYEES SET SALARY = 6500 WHERE LAST_NAME = 'Mavris';
   ```

3. If you want, lock and expire `HR`, unless other users want to use this account:

   ```
   ALTER USER HR PASSWORD EXPIRE ACCOUNT LOCK;
   ```

4. Issue the following `ALTER SYSTEM` statement to restore the `SMTP_OUT_SERVER` parameter to the previous value, from Step 4 under Step 1: Install and Configure the UTL_MAIL PL/SQL Package:

   ```
   ALTER SYSTEM SET SMTP_OUT_SERVER="previous_value";
   ```

   Enclose this setting in quotation marks. For example:

   ```
   ALTER SYSTEM SET SMTP_OUT_SERVER="some_imap_server.example.com"
   ```

5. Connect to the CDB root as a user who has the `SYSDBA` administrative privilege.

   ```
   CONNECT / AS SYSDBA
   ```

6. Close and then reopen the PDB.

   ```
   ALTER PLUGGABLE DATABASE pdb_name CLOSE IMMEDIATE;
   ALTER PLUGGABLE DATABASE pdb_name OPEN;
   ```

31.5 Fine-Grained Audit Policy Data Dictionary Views

You can query data dictionary and dynamic views to find detailed auditing information about fine-grained audit policies.

Table 30-20 lists these views.

> **Tip:**
>
> To find error information about audit policies, check the trace files. The `USER_DUMP_DEST` initialization parameter sets the location of the trace files.

Table 31-1 Views for Use with Fine-Grained Audit Policies

View	Description
`ALL_AUDIT_POLICIES`	Displays information about all fine-grained audit policies
`ALL_DEF_AUDIT_OPTS`	Lists default object-auditing options that are to be applied when objects are created
`AUDITABLE_SYSTEM_ACTIONS`	Maps the auditable system action numbers to the action names
`CDB_UNIFIED_AUDIT_TRAIL`	Similar to the `UNIFIED_AUDIT_TRAIL` view, displays the audit records but from all PDBs in a multitenant environment. This view is available only in the CDB root and must be queried from there.
`DBA_AUDIT_POLICIES`	Displays information about fine-grained audit policies
`DBA_SA_AUDIT_OPTIONS`	Describes audited Oracle Label Security events performed by users, and indicates if the user's action failed or succeeded
`SYSTEM_PRIVILEGE_MAP` (table)	Describes privilege (auditing option) type codes. This table can be used to map privilege (auditing option) type numbers to type names.
`USER_AUDIT_POLICIES`	Displays information about all fine-grained audit policies on table and views owned by the current user
`UNIFIED_AUDIT_TRAIL`	Displays all audit records

Related Topics

- *Oracle Database Reference*