14

Using Oracle Virtual Private Database to Control Data Access

Oracle Virtual Private Database (VPD) enables you to filter users who access data.

- About Oracle Virtual Private Database
  Oracle Virtual Private Database (VPD) provides important benefits for filtering user access to data.
- Components of an Oracle Virtual Private Database Policy
  A VPD policy uses a function to generate the dynamic `WHERE` clause, and a policy to attach the function to objects to protect.
- Configuration of Oracle Virtual Private Database Policies
  The `DBMS_RLS` PL/SQL package can configure Oracle Virtual Private Database (VPD) policies.
- Tutorials: Creating Oracle Virtual Private Database Policies
  These tutorials show how to create a simple and a database session-based Oracle Virtual Private policy, and how to create policy groups.
- How Oracle Virtual Private Database Works with Other Oracle Features
  You should be aware of the impact of using Oracle Virtual Private Database with other Oracle features.
- Oracle Virtual Private Database Data Dictionary Views
  Oracle Database provides data dictionary views that list information about Oracle Virtual Private Database policies.

14.1 About Oracle Virtual Private Database

Oracle Virtual Private Database (VPD) provides important benefits for filtering user access to data.

- What Is Oracle Virtual Private Database?
  Oracle Virtual Private Database (VPD) creates security policies to control database access at the row and column level.
- Benefits of Using Oracle Virtual Private Database Policies
  Oracle Virtual Private Database policies provide the important benefits.
- Who Can Create Oracle Virtual Private Database Policies?
  The `DBMS_RLS` PL/SQL package enables you to create Oracle Virtual Private Database (VPD) policies.
- Privileges to Run Oracle Virtual Private Database Policy Functions
  You should be aware of the correct privileges for running Oracle Virtual Private Database (VPD) policy functions.
- Oracle Virtual Private Database Use with an Application Context
  You can use application contexts with Oracle Virtual Private Database policies.

- Oracle Virtual Private Database in a Multitenant Environment
  You can create Virtual Private Database policies in an application root for use throughout any associated application PDBs.

14.1.1 What Is Oracle Virtual Private Database?

Oracle Virtual Private Database (VPD) creates security policies to control database access at the row and column level.

Essentially, Oracle Virtual Private Database adds a dynamic `WHERE` clause to a SQL statement that is issued against the table, view, or synonym to which an Oracle Virtual Private Database security policy was applied.

Oracle Virtual Private Database enforces security, to a fine level of granularity, directly on database tables, views, or synonyms. Because you attach security policies directly to these database objects, and the policies are automatically applied whenever a user accesses data, there is no way to bypass security.

When a user directly or indirectly accesses a table, view, or synonym that is protected with an Oracle Virtual Private Database policy, Oracle Database dynamically modifies the SQL statement of the user. This modification creates a `WHERE` condition (called a predicate) returned by a function implementing the security policy. Oracle Database modifies the statement dynamically, transparently to the user, using any condition that can be expressed in or returned by a function. You can apply Oracle Virtual Private Database policies to `SELECT`, `INSERT`, `UPDATE`, `INDEX`, and `DELETE` statements.

For example, suppose a user performs the following query:

```
SELECT * FROM OE.ORDERS;
```

The Oracle Virtual Private Database policy dynamically appends the statement with a `WHERE` clause. For example:

```
SELECT * FROM OE.ORDERS
 WHERE SALES_REP_ID = 159;
```

In this example, the user can only view orders by Sales Representative 159.

If you want to filter the user based on the session information of that user, such as the ID of the user, then you can create the `WHERE` clause to use an application context. For example:

```
SELECT * FROM OE.ORDERS
 WHERE SALES_REP_ID = SYS_CONTEXT('USERENV','SESSION_USER');
```

Note the following:

- Oracle Database release 12c introduced Real Application Security (RAS) to supersede VPD. Oracle recommends that you use RAS for new projects that require row and column level access controls for their applications.
- Oracle Database does not protect tables and views that have VPD policies against the `SYS` user and against users who have an out-of-the-box database administrator role. The Oracle Database-supplied `DBA` role has privileges that can alter and remove VPD policies, and hence can access table and view data.

- Oracle Virtual Private Database does not support filtering for DDLs, such as `TRUNCATE` or `ALTER TABLE` statements.

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.

14.1.2 Benefits of Using Oracle Virtual Private Database Policies

Oracle Virtual Private Database policies provide the important benefits.

- Security Policies Based on Database Objects Rather Than Applications
  Oracle Virtual Private Database provides benefits in security, simplicity, and flexibility.
- Control Over How Oracle Database Evaluates Policy Functions
  Running policy functions multiple times can affect performance.

14.1.2.1 Security Policies Based on Database Objects Rather Than Applications

Oracle Virtual Private Database provides benefits in security, simplicity, and flexibility.

Attaching Oracle Virtual Private Database security policies to database tables, views, or synonyms, rather than implementing access controls in all your applications, provides the following benefits:

- **Security.** Associating a policy with a database table, view, or synonym can solve a potentially serious application security problem. Suppose a user is authorized to use an application, and then drawing on the privileges associated with that application, wrongfully modifies the database by using an ad hoc query tool, such as SQL*Plus. By attaching security policies directly to tables, views, or synonyms, fine-grained access control ensures that the same security is in force, no matter how a user accesses the data.
- **Simplicity.** You add the security policy to a table, view, or synonym only once, rather than repeatedly adding it to each of your table-based, view-based, or synonym-based applications.
- **Flexibility.** You can have one security policy for `SELECT` statements, another for `INSERT` statements, and still others for `UPDATE` and `DELETE` statements. For example, you might want to enable Human Resources clerks to have `SELECT` privileges for all employee records in their division, but to update only salaries for those employees in their division whose last names begin with `A` through `F`. Furthermore, you can create multiple policies for each table, view, or synonym.

14.1.2.2 Control Over How Oracle Database Evaluates Policy Functions

Running policy functions multiple times can affect performance.

You can control the performance of policy functions by configuring how Oracle Database caches the Oracle Virtual Private Database predicates.

The following options are available:

- Evaluate the policy once for each query (static policies).
- Evaluate the policy only when an application context within the policy function changes (context-sensitive policies).
- Evaluate the policy each time it is run (dynamic policies).

Related Topics

- Optimizing Performance by Using Oracle Virtual Private Database Policy Types
  You can optimize performance by using the Oracle Virtual Private Database (VPD) the dynamic, static, or shared policy types.

14.1.3 Who Can Create Oracle Virtual Private Database Policies?

The `DBMS_RLS` PL/SQL package enables you to create Oracle Virtual Private Database (VPD) policies.

You must be granted the `EXECUTE` privilege on the `DBMS_RLS` PL/SQL package to create Oracle Virtual Private Database policies. You must also be granted the `ADMINISTER ROW LEVEL SECURITY POLICY` system privilege in one of the following ways:

- Syntax of the `ADMINISTER ROW LEVEL SECURITY POLICY` privilege grant if the VPD policy is to apply to all non-`SYS` schemas across the database:

  ```
  GRANT ADMINISTER ROW LEVEL SECURITY POLICY TO grantee;
  ```

- Syntax of the `ADMINISTER ROW LEVEL SECURITY POLICY` privilege grant if the VPD policy is to be restricted to a specific schema:

  ```
  GRANT ADMINISTER ROW LEVEL SECURITY POLICY ON SCHEMA schema TO grantee;
  ```

As with all privileges, an administrator must only grant these privileges to trusted users. You can find the privileges that a user has been granted by querying the `DBA_SYS_PRIVS` data dictionary view.

14.1.4 Privileges to Run Oracle Virtual Private Database Policy Functions

You should be aware of the correct privileges for running Oracle Virtual Private Database (VPD) policy functions.

For greater security, the Oracle Virtual Private Database policy function runs as if it had been declared with definer's rights.

Do not declare it as invoker's rights because this can confuse yourself and other users who maintain the code.

Related Topics

- *Oracle Database PL/SQL Language Reference*

14.1.5 Oracle Virtual Private Database Use with an Application Context

You can use application contexts with Oracle Virtual Private Database policies.

When you create an application context, it securely caches user information. Only the designated application package can set the cached environment. It cannot be changed by the user or outside the package. In addition, because the data is cached, performance is increased.

For example, suppose you want to base access to the `ORDERS_TAB` table on the customer ID number. Rather than querying the customer ID number for a logged-in user each time you need it, you could store the number in the application context. Then, the customer number is available in the session when you need it.

Application contexts are especially helpful if your security policy is based on multiple security attributes. For example, if a policy function bases a `WHERE` predicate on four attributes (such as employee number, cost center, position, spending limit), then multiple subqueries must run to retrieve this information. Instead, if this data is available through an application context, then performance is much faster.

You can use an application context to return the correct security policy, enforced through a predicate. For example, consider an order entry application that enforces the following rules: customers only see their own orders, and clerks see all orders for all customers. These are two different policies. You could define an application context with a `position` attribute, and this attribute could be accessed within the policy function to return the correct predicate, depending on the value of the attribute. Thus, you can enable a user in the `clerk` position to retrieve all orders, but a user in the `customer` position can see only those records associated with that particular user.

To design a fine-grained access control policy that returns a specific predicate for an attribute, you need to access the application context within the function that implements the policy. For example, suppose you want to limit customers to seeing only their own records. The user performs the following query:

```
SELECT * FROM orders_tab
```

Fine-grained access control dynamically modifies this query to include the following `WHERE` predicate:

```
SELECT * FROM orders_tab
  WHERE custno = SYS_CONTEXT ('order_entry', 'cust_num');
```

Continuing with the preceding example, suppose you have 50,000 customers, and you do not want to have a different predicate returned for each customer. Customers all share the same `WHERE` predicate, which prescribes that they can only see their own orders. It is merely their customer numbers that are different.

Using application context, you can return one `WHERE` predicate within a policy function that applies to 50,000 customers. As a result, there is one shared cursor that executes differently for each customer, because the customer number is evaluated at execution time. This value is different for every customer. Use of application context in this case provides optimum performance, and at row-level security.

The `SYS_CONTEXT` function works much like a bind variable; only the `SYS_CONTEXT` arguments are constants.

Related Topics

- Using Application Contexts to Retrieve User Information
  An application context stores user identification that can enable or prevent a user from accessing data in the database.

14.1.6 Oracle Virtual Private Database in a Multitenant Environment

You can create Virtual Private Database policies in an application root for use throughout any associated application PDBs.

The CDB restriction applies to shared context sensitive policies and views related to Virtual Private Database policies as well. You cannot create a Virtual Private Database policy for an entire multitenant environment.

With regard to application containers, you can create Virtual Private Database policies to protect application common objects by applying the common policy to all PDBs that belong to

the application root. In other words, when you install an application in the application root, all the common Virtual Private Database policies that protect the common objects will be applied to and immediately enforced for all PDBs in the application container.

Note the following:

- You can only create the common Virtual Private Database policy and its associated PL/SQL function in the application root and only attach it to application common objects. If the function is not in the same location as the policy, then an error is raised at runtime.
- A Virtual Private Database policy that is applied to common objects is considered a common policy that will be automatically enforced in PDBs that belong to the application container when it accesses the application common objects from application PDBs.
- Application common Virtual Private Database policies can only protect application common objects.
- A Virtual Private Database policy that is applied to application common objects in the application root and is applied to all application PDBs is considered a common Virtual Private Database policy. A policy that is applied to a local database table and enforced in one PDB is considered a local Virtual Private Database policy.

  For example, if policy `VPD_P1` is applied to the application common table `T1` in the application root, then it is a considered to be a common policy. It will be enforced in each application PDB. If a policy named `VPD_P1` is applied to a local table called `T1` in `PDB1`, then it is considered a local policy, which means that it affects only `PDB1`. If a policy called `VPD_P1` is applied to a local table `T1` in the application root, then it is still considered a local policy because it affects only the application root. This concept applies to other operations, such as enabling, disabling, and removing Virtual Private Database policies.
- Application common Virtual Private Database policies only protect application common objects, while local Virtual Private Database policies only protect local objects.
- If you are using application contexts, then ensure common database session-based application contexts and common global application context objects are used in the common Virtual Private Database configuration.
- Application container Virtual Private Database policies are stored in the application root. PDBs store only local policies. If you plug a PDB into the application container, then the common policies are not converted to local policies. Instead, Oracle Database loads them from the application root and enforces them in the local PDB when the policies access common objects in the local PDB.

14.2 Components of an Oracle Virtual Private Database Policy

A VPD policy uses a function to generate the dynamic `WHERE` clause, and a policy to attach the function to objects to protect.

- Function to Generate the Dynamic WHERE Clause
  The Oracle Virtual Private Database (VPD) function defines the restrictions that you want to enforce.
- Policies to Attach the Function to the Objects You Want to Protect
  The Oracle Virtual Private Database policy associates the VPD function with a table, view, or synonym.

14.2.1 Function to Generate the Dynamic WHERE Clause

The Oracle Virtual Private Database (VPD) function defines the restrictions that you want to enforce.

To generate the Oracle Virtual Private Database (VPD) dynamic `WHERE` clause (predicate), you must create a function (not a procedure) that defines these restrictions. This function is a definer's rights function. Oracle Database generates the predicate with the VPD policy function authorized by the owner but in the same current user session such that the PL/SQL global variables that are defined in the function will be used.

Usually, the security administrator creates this function in their own schema. For more complex behavior, such as including calls to other functions or adding checks to track failed logon attempts, create these functions within a package.

The function must have the following behavior:

- **It must take as arguments a schema name and an object (table, view, or synonym) name as inputs.** Define input parameters to hold this information, but do not specify the schema and object name themselves within the function. The policy that you create to attach the function to the objects that you want to protect, using the `DBMS_RLS` package, provides the names of the schema, and object to which the policy will apply. You must create the parameter for the schema first, followed by the parameter for the object.
- **It must provide a return value for the WHERE clause predicate that will be generated.** The return value for the `WHERE` clause is always a `VARCHAR2` data type.
- **It must generate a valid WHERE clause.** This code can be as simple in that it applies to every user who logs in the database instance, but in most cases, you may want to design the `WHERE` clause to be different for each user, each group of users, or each application that accesses the objects you want to protect. For example, if a manager logs in, the `WHERE` clause can be specific to the rights of that particular manager. You can do this by incorporating an application context, which accesses user session information, into the `WHERE` clause generation code.

  You can create Oracle Virtual Private Database functions that do not use an application context, but an application context creates a much stronger Oracle Virtual Private Database policy, by securely basing user access on the session attributes of that user, such as the user ID.

  In addition, you can embed C or Java calls to access operating system information or to return `WHERE` clauses from an operating system file or other source.
- **It must not select from a table within the associated policy function.** Although you can define a policy against a table, you cannot select that table from within the policy that was defined against the table.
- **It must be a pure function.** The VPD function must rely only on the application context and the arguments that are passed to the function to generate the `WHERE` clause. This function must not depend on the package variables.

> **Note:**
>
> If you plan to run the function across different editions, you can control the results of the function: whether the results are uniform across all editions, or specific to the edition in which the function is run.

Related Topics

- Policies to Attach the Function to the Objects You Want to Protect
  The Oracle Virtual Private Database policy associates the VPD function with a table, view, or synonym.

- Tutorial: Creating a Simple Oracle Virtual Private Database Policy
  This tutorial shows how to create a simple Oracle Virtual Private Database policy using the `OE` user account.
- Tutorial: Implementing a Session-Based Application Context Policy
  This tutorial demonstrates how to create an Oracle Virtual Private Database policy that uses a database session-based application context.
- Using Application Contexts to Retrieve User Information
  An application context stores user identification that can enable or prevent a user from accessing data in the database.
- 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.

14.2.2 Policies to Attach the Function to the Objects You Want to Protect

The Oracle Virtual Private Database policy associates the VPD function with a table, view, or synonym.

You create the policy by using the `DBMS_RLS` package. If you are not `SYS`, then you must be granted `EXECUTE` privileges to use the `DBMS_RLS` package. This package contains procedures that enable you to manage the policy and set fine-grained access control. For example, to attach the policy to a table, you use the `DBMS_RLS.ADD_POLICY` procedure. Within this setting, you set fine-grained access control, such as setting the policy to go into effect when a user issues a `SELECT` or `UPDATE` statement on the table or view.

The combination of creating the function and then applying it to a table or view is referred to as creating the Oracle Virtual Private Database policy.

Related Topics

- Configuration of Oracle Virtual Private Database Policies
  The `DBMS_RLS` PL/SQL package can configure Oracle Virtual Private Database (VPD) policies.
- Tutorials: Creating Oracle Virtual Private Database Policies
  These tutorials show how to create a simple and a database session-based Oracle Virtual Private policy, and how to create policy groups.

14.3 Configuration of Oracle Virtual Private Database Policies

The `DBMS_RLS` PL/SQL package can configure Oracle Virtual Private Database (VPD) policies.

- About Oracle Virtual Private Database Policies
  The Oracle Virtual Private Database policy associates the VPD function with a database table, view, or synonym.
- Attaching a Policy to a Database Table, View, or Synonym
  The `DBMS_RLS` PL/SQL package can attach a policy to a table, view, or synonym.
- Example: Attaching a Simple Oracle Virtual Private Database Policy to a Table
  The `DBMS_RLS.ADD_POLICY` procedure can attach an Oracle Virtual Private Database (VPD) policy to a table, view, or synomym.
- Enforcing Policies on Specific SQL Statement Types
  You can enforce Oracle Virtual Private Database policies for `SELECT`, `INSERT`, `UPDATE`, `INDEX`, and `DELETE` statements.

- Example: Specifying SQL Statement Types with DBMS_RLS.ADD_POLICY
  The `DBMS_RLS.ADD_POLICY` procedure `statement_types` parameter can specify the `SELECT` and `INDEX` statements for a policy.
- Control of the Display of Column Data with Policies
  You can create policies that enforce row-level security when a security-relevant column is referenced in a query.
- Oracle Virtual Private Database Policy Groups
  An Oracle Virtual Private Database policy group is a named collection of VPD policies that can be applied to an application.
- Optimizing Performance by Using Oracle Virtual Private Database Policy Types
  You can optimize performance by using the Oracle Virtual Private Database (VPD) the dynamic, static, or shared policy types.

14.3.1 About Oracle Virtual Private Database Policies

The Oracle Virtual Private Database policy associates the VPD function with a database table, view, or synonym.

This function defines the actions of the Oracle Virtual Private Database `WHERE` clause. You must then associate this function with the database table to which the Oracle Virtual Private Database (VPD) action applies.

You can do this by configuring an Oracle Virtual Private Database policy. The policy is a mechanism for managing the Virtual Private Database function. The policy also enables you to add fine-grained access control, such as specifying the types of SQL statements or particular table columns the policy affects. When a user tries to access the data in this database object, the policy goes into effect automatically.

Table 14-1 lists the procedures in the `DBMS_RLS` package.

Table 14-1 DBMS_RLS Procedures

Procedure	Description
For Handling Individual Policies	-
`DBMS_RLS.ADD_POLICY`	Adds a policy to a table, view, or synonym
`DBMS_RLS.ENABLE_POLICY`	Enables (or disables) a policy that is previously created on a table, view, or synonym
`DBMS_RLS.ALTER_POLICY`	Alters an existing policy to associate or disassociate attributes with the policy
`DBMS_RLS.REFRESH_POLICY`	Invalidates cursors associated with nonstatic policies
`DBMS_RLS.DROP_POLICY`	To drop a policy from a table, view, or synonym
For Handling Grouped Policies	-
`DBMS_RLS.CREATE_POLICY_GROUP`	Creates a policy group
`DBMS_RLS.ALTER_GROUPED_POLICY`	Alters a policy group
`DBMS_RLS.DELETE_POLICY_GROUP`	Drops a policy group
`DBMS_RLS.ADD_GROUPED_POLICY`	Adds a policy to the specified policy group
`DBMS_RLS.ENABLE_GROUPED_POLICY`	Enables a policy within a group

Table 14-1 (Cont.) DBMS_RLS Procedures

Procedure	Description
DBMS_RLS.REFRESH_GROUPED_POLICY	Parses again the SQL statements associated with a refreshed policy
DBMS_RLS.DISABLE_GROUPED_POLICY	Disables a policy within a group
DBMS_RLS.DROP_GROUPED_POLICY	Drops a policy that is a member of the specified group
For Handling Application Contexts	-
DBMS_RLS.ADD_POLICY_CONTEXT	Adds the application context for the active application
DBMS_RLS.DROP_POLICY_CONTEXT	Drops the context for the application

Related Topics

- Components of an Oracle Virtual Private Database Policy
  A VPD policy uses a function to generate the dynamic `WHERE` clause, and a policy to attach the function to objects to protect.
- Using Application Contexts to Retrieve User Information
  An application context stores user identification that can enable or prevent a user from accessing data in the database.

14.3.2 Attaching a Policy to a Database Table, View, or Synonym

The `DBMS_RLS` PL/SQL package can attach a policy to a table, view, or synonym.

- To attach a policy to a database table, view, or synonym, use the `DBMS_RLS.ADD_POLICY` procedure.

You must specify the table, view, or synonym to which you are adding a policy, and a name for the policy. You can also specify other information, such as the types of statements the policy controls (`SELECT`, `INSERT`, `UPDATE`, `DELETE`, `CREATE INDEX`, or `ALTER INDEX`).

Follow these guidelines:

- If a view has been created as an extended data-linked object, then Oracle recommends that you apply the same VPD policy on this type of view as you would on the underlying objects of the view.

  This applies to secondary tables made for use with hybrid vector indexes and Oracle Text indexes. For more information, see the Guidelines and Restrictions for Hybrid Vector Indexes in the *Oracle Database AI Vector Search User's Guide* and *Oracle Text Application Developer's Guide*, respectively.
- Determine if the base object to which you want to add the VPD policy has dependent objects. If it does have dependent objects, then these objects will become invalid when the VPD policy is added to the base object, and these objects will be recompiled automatically when they are used.

  Alternatively, you can proactively recompile them yourself by using an `ALTER ... COMPILE` statement. Be aware that invalidating dependent objects (by adding a VPD policy on their base object) and causing them to need to be recompiled can decrease performance in the overall system. Oracle recommends that you only add a VPD policy to an object that has dependent objects during off-peak hours or during a scheduled downtime.

- Be aware that the maximum number of policies that can be created for a single object is 255.

14.3.3 Example: Attaching a Simple Oracle Virtual Private Database Policy to a Table

The `DBMS_RLS.ADD_POLICY` procedure can attach an Oracle Virtual Private Database (VPD) policy to a table, view, or synomym.

Example 14-1 shows how to use `DBMS_RLS.ADD_POLICY` to attach an Oracle Virtual Private Database policy called `secure_update` to the `HR.EMPLOYEES` table. The function attached to the policy is `check_updates`.

Example 14-1 Attaching a Simple Oracle Virtual Private Database Policy to a Table

```
BEGIN
 DBMS_RLS.ADD_POLICY(
  object_schema   => 'hr',
  object_name     => 'employees',
  policy_name     => 'secure_update',
  policy_function => 'check_updates',
...
```

If the function was created inside a package, include the package name. For example:

```
 policy_function => 'pkg.check_updates',
...
```

Although you can define a policy against a table, you cannot select that table from within the policy that was defined against the table.

14.3.4 Enforcing Policies on Specific SQL Statement Types

You can enforce Oracle Virtual Private Database policies for `SELECT`, `INSERT`, `UPDATE`, `INDEX`, and `DELETE` statements.

- To specify a SQL statement type for the policy, use the `statement_types` parameter in the `DBMS_RLS.ADD_POLICY` procedure. If you want to specify more than one, separate each with a comma. Enclose the list in a pair of single quotation marks.

If you do not specify a statement type, then by default, Oracle Database specifies `SELECT`, `INSERT`, `UPDATE`, and `DELETE`, but not `INDEX`. You can enter any combination of these statement types.

When you specify the `statement_types` parameter, be aware of the following functionality:

- **The application code affected by the Virtual Private Database policy can include the MERGE INTO statement.** However, in the Virtual Private Database policy, you must ensure that the `statement_types` parameter includes all three of the `INSERT`, `UPDATE`, and `DELETE` statements for the policy to succeed. Alternatively, you can omit the `statement_types` parameter.
- **Be aware that a user who has privileges to maintain an index can see all the row data, even if the user does not have full table access under a regular query such as SELECT.** For example, a user can create a function-based index that contains a user-defined function with column values as its arguments. During index creation, Oracle Database passes column values of every row into the user function, making the row data available to the user who creates the index. You can enforce Oracle Virtual Private

Database policies on index maintenance operations by specifying `INDEX` with the `statement_types` parameter.

14.3.5 Example: Specifying SQL Statement Types with DBMS_RLS.ADD_POLICY

The `DBMS_RLS.ADD_POLICY` procedure `statement_types` parameter can specify the `SELECT` and `INDEX` statements for a policy.

Example 14-2 shows an how this works.

Example 14-2 Specifying SQL Statement Types with DBMS_RLS.ADD_POLICY

```
BEGIN
 DBMS_RLS.ADD_POLICY(
  object_schema   => 'hr',
  object_name     => 'employees',
  policy_name     => 'secure_update',
  policy_function => 'check_updates',
  statement_types => 'SELECT,INDEX');
END;
/
```

14.3.6 Control of the Display of Column Data with Policies

You can create policies that enforce row-level security when a security-relevant column is referenced in a query.

- Policies for Column-Level Oracle Virtual Private Database
  Column-level policies enforce row-level security when a query references a security-relevant column.
- Example: Creating a Column-Level Oracle Virtual Private Database Policy
  The `CREATE FUNCTION` statement and the `DBMS_RLS.ADD_POLICY` procedure can configure a column-level Oracle Virtual Private Database policy.
- Display of Only the Column Rows Relevant to the Query
  Be default, column-level Oracle Virtual Private Database restricts the number of rows a query returns that references columns containing sensitive information.
- Column Masking to Display Sensitive Columns as NULL Values
  If a query references a sensitive column, then by default column-level Oracle Virtual Private Database restricts the number of rows returned.
- Example: Adding Column Masking to an Oracle Virtual Private Database Policy
  The `DBMS_RLS.ADD_POLICY` procedure can configure column-level Oracle Virtual Private Database column masking.

14.3.6.1 Policies for Column-Level Oracle Virtual Private Database

Column-level policies enforce row-level security when a query references a security-relevant column.

You can apply a column-level Oracle Virtual Private Database policy to tables and views, but not to synonyms. To apply the policy to a column, specify the security-relevant column by using the `SEC_RELEVANT_COLS` parameter of the `DBMS_RLS.ADD_POLICY` procedure. This parameter applies the security policy whenever the column is referenced, explicitly or implicitly, in a query.

For example, users who are not in a Human Resources department typically are allowed to view only their own Social Security numbers. A sales clerk initiates the following query:

```
SELECT fname, lname, ssn FROM emp;
```

The function implementing the security policy returns the predicate `ssn='my_ssn'`. Oracle Database rewrites the query and executes the following:

```
SELECT fname, lname, ssn FROM emp
 WHERE ssn = 'my_ssn';
```

14.3.6.2 Example: Creating a Column-Level Oracle Virtual Private Database Policy

The `CREATE FUNCTION` statement and the `DBMS_RLS.ADD_POLICY` procedure can configure a column-level Oracle Virtual Private Database policy.

Example 14-3 shows an Oracle Virtual Private Database policy in which sales department users cannot see the salaries of people outside the department (department number 30) of the sales department users. The relevant columns for this policy are `sal` and `comm`. First, the Oracle Virtual Private Database policy function is created, and then it is added by using the `DBMS_RLS` PL/SQL package.

Example 14-3 Creating a Column-Level Oracle Virtual Private Database Policy

```
CREATE OR REPLACE FUNCTION hide_sal_comm (
 v_schema IN VARCHAR2,
 v_objname IN VARCHAR2)

RETURN VARCHAR2 AS
con VARCHAR2 (200);

BEGIN
 con := 'deptno=30';
 RETURN (con);
END hide_sal_comm;
```

Then you configure the policy with the `DBMS_RLS.ADD_POLICY` procedure as follows:

```
BEGIN
 DBMS_RLS.ADD_POLICY (
  object_schema     => 'scott',
  object_name       => 'emp',
  policy_name       => 'hide_sal_policy',
  policy_function   => 'hide_sal_comm',
  sec_relevant_cols => 'sal,comm');
END;
```

14.3.6.3 Display of Only the Column Rows Relevant to the Query

Be default, column-level Oracle Virtual Private Database restricts the number of rows a query returns that references columns containing sensitive information.

You specify these security-relevant columns by using the `SEC_RELEVANT_COLUMNS` parameter of the `DBMS_RLS.ADD_POLICY` procedure.

For example, consider sales department users with the `SELECT` privilege on the `emp` table, which is protected with the column-level Oracle Virtual Private Database policy created earlier that illustrates how to create a column-level Oracle Virtual Private Database policy. The user (for example, user `SCOTT`) runs the following query:

```
SELECT ENAME, d.dname, JOB, SAL, COMM
 FROM emp e, dept d
 WHERE d.deptno = e.deptno;
```

The database returns the following rows:

```
ENAME      DNAME          JOB              SAL       COMM
---------- -------------- --------- ---------- ----------
ALLEN      SALES          SALESREP         1600        300
WARD       SALES          SALESREP         1250        500
MARTIN     SALES          SALESREP         1250       1400
BLAKE      SALES          MANAGER          2850
TURNER     SALES          SALESREP         1500          0
JAMES      SALES          CLERK             950

6 rows selected.
```

The only rows that are displayed are those that the user has privileges to access all columns in the row.

Related Topics

- Example: Creating a Column-Level Oracle Virtual Private Database Policy
  The `CREATE FUNCTION` statement and the `DBMS_RLS.ADD_POLICY` procedure can configure a column-level Oracle Virtual Private Database policy.

14.3.6.4 Column Masking to Display Sensitive Columns as NULL Values

If a query references a sensitive column, then by default column-level Oracle Virtual Private Database restricts the number of rows returned.

With column-masking behavior, all rows display, even those that reference sensitive columns. However, the sensitive columns display as `NULL` values. To enable column-masking, set the `SEC_RELEVANT_COLS_opt` parameter of the `DBMS_RLS.ADD_POLICY` procedure.

For example, consider the results of the sales clerk query, described in the previous example. If column-masking is used, then instead of seeing only the row containing the details and Social Security number of the sales clerk, the clerk would see all rows from the `emp` table, but the `ssn` column values would be returned as `NULL`. Note that this behavior is fundamentally different from all other types of Oracle Virtual Private Database policies, which return only a subset of rows.

In contrast to the default action of column-level Oracle Virtual Private Database, column-masking displays all rows, but returns sensitive column values as `NULL`. To include column-masking in your policy, set the `SEC_RELEVANT_COLS_OPT` parameter of the `DBMS_RLS.ADD_POLICY` procedure to `DBMS_RLS.ALL_ROWS`.

The following considerations apply to column masking:

- Column-masking applies only to `SELECT` statements.
- Column-masking conditions generated by the policy function must be simple Boolean expressions, unlike regular Oracle Virtual Private Database predicates.
- For applications that perform calculations, or do not expect `NULL` values, use standard column-level Oracle Virtual Private Database, specifying `SEC_RELEVANT_COLS` rather than the `SEC_RELEVANT_COLS_OPT` column-masking option.
- Do not include columns of the object data type (including the `XMLtype`) in the `sec_relevant_cols` setting. This column type is not supported for the `sec_relevant_cols` setting.

- Column-masking used with `UPDATE AS SELECT` updates only the columns that users are allowed to see.
- For some queries, column-masking may prevent some rows from displaying. For example:

```
SELECT * FROM emp
 WHERE sal = 10;
```

  Because the column-masking option was set, this query may not return rows if the `salary` column returns a `NULL` value.

14.3.6.5 Example: Adding Column Masking to an Oracle Virtual Private Database Policy

The `DBMS_RLS.ADD_POLICY` procedure can configure column-level Oracle Virtual Private Database column masking.

Example 14-4 shows column-level Oracle Virtual Private Database column masking. It uses the same VPD policy as the one created earlier that uses a column-level policy, but with `sec_relevant_cols_opt` specified as `DBMS_RLS.ALL_ROWS`.

Example 14-4 Adding Column Masking to an Oracle Virtual Private Database Policy

```
BEGIN
 DBMS_RLS.ADD_POLICY(
   object_schema         => 'scott',
   object_name           => 'emp',
   policy_name           => 'hide_sal_policy',
   policy_function       => 'hide_sal_comm',
   sec_relevant_cols     =>' sal,comm',
   sec_relevant_cols_opt => dbms_rls.ALL_ROWS);
END;
```

Assume that a sales department user with `SELECT` privilege on the `emp` table (such as user `SCOTT`) runs the following query:

```
SELECT ENAME, d.dname, job, sal, comm
 FROM emp e, dept d
 WHERE d.deptno = e.deptno;
```

The database returns all rows specified in the query, but with certain values masked because of the Oracle Virtual Private Database policy:

```
ENAME      DNAME          JOB              SAL       COMM
---------- -------------- --------- ---------- ----------
CLARK      ACCOUNTING     MANAGER
KING       ACCOUNTING     PRESIDENT
MILLER     ACCOUNTING     CLERK
JONES      RESEARCH       MANAGER
FORD       RESEARCH       ANALYST
ADAMS      RESEARCH       CLERK
SMITH      RESEARCH       CLERK
SCOTT      RESEARCH       ANALYST
WARD       SALES          SALESREP        1250        500
TURNER     SALES          SALESREP        1500          0
ALLEN      SALES          SALESREP        1600        300
JAMES      SALES          CLERK            950
BLAKE      SALES          MANAGER         2850
MARTIN     SALES          SALESREP        1250       1400

14 rows selected.
```

The column-masking returned all rows requested by the sales user query, but made the `sal` and `comm` columns `NULL` for employees outside the sales department.

Related Topics

- Example: Creating a Column-Level Oracle Virtual Private Database Policy
  The `CREATE FUNCTION` statement and the `DBMS_RLS.ADD_POLICY` procedure can configure a column-level Oracle Virtual Private Database policy.

14.3.7 Oracle Virtual Private Database Policy Groups

An Oracle Virtual Private Database policy group is a named collection of VPD policies that can be applied to an application.

- About Oracle Virtual Private Database Policy Groups
  You can group multiple security policies together, and apply them to an application.
- Creation of a New Oracle Virtual Private Database Policy Group
  The `DBMS_RLS.ADD_GROUPED_POLICY` procedure adds a VPD policy to a VPD policy group.
- Default Policy Group with the SYS_DEFAULT Policy Group
  Within a group of security policies, you can designate one security policy to be the default security policy.
- Multiple Policies for Each Table, View, or Synonym
  You can establish several policies for the same table, view, or synonym.
- Validation of the Application Used to Connect to the Database
  The package implementing the driving context must correctly validate the application that is being used to connect to the database.

14.3.7.1 About Oracle Virtual Private Database Policy Groups

You can group multiple security policies together, and apply them to an application.

A policy group is a set of security policies that belong to an application. You can designate an application context (known as a *driving context* or *policy context*) to indicate the policy group in effect. Then, when a user accesses the table, view, or synonym column, Oracle Database looks up the driving context to determine the policy group in effect. It enforces all the associated policies that belong to the policy group.

Policy groups are useful for situations where multiple applications with multiple security policies share the same table, view, or synonym. This enables you to identify those policies that should be in effect when the table, view, or synonym is accessed.

For example, in a hosting environment, Company A can host the `BENEFIT` table for Company B and Company C. The table is accessed by two different applications, Human Resources and Finance, with two different security policies. The Human Resources application authorizes users based on ranking in the company, and the Finance application authorizes users based on department. Integrating these two policies into the `BENEFIT` table requires joint development of policies between the two companies, which is not a feasible option. By defining an application context to drive the enforcement of a particular set of policies to the base objects, each application can implement a private set of security policies.

To do this, you organize security policies into groups. By referring to the application context, Oracle Database determines which group of policies should be in effect at run time. The server enforces all the policies that belong to that policy group.

14.3.7.2 Creation of a New Oracle Virtual Private Database Policy Group

The `DBMS_RLS.ADD_GROUPED_POLICY` procedure adds a VPD policy to a VPD policy group.

To specify which policies will be effective, you can add a driving context using the `DBMS_RLS.ADD_POLICY_CONTEXT` procedure. If the driving context returns an unknown policy group, then an error is returned.

If the driving context is not defined, then Oracle Database runs all policies. Likewise, if the driving context is `NULL`, then policies from all policy groups are enforced. An application accessing the data cannot bypass the security setup module (which sets up application context) to avoid any applicable policies.

You can apply multiple driving contexts to the same table, view, or synonym, and each of them will be processed individually. This enables you to configure multiple active sets of policies to be enforced.

Consider, for example, a hosting company that hosts Benefits and Financial applications, which share some database objects. Both applications are striped for hosting using a `SUBSCRIBER` policy in the `SYS_DEFAULT` policy group. Data access is partitioned first by subscriber ID, then by whether the user is accessing the Benefits or Financial applications (determined by a driving context). Suppose that Company A, which uses the hosting services, wants to apply a custom policy that relates only to its own data access. You could add an additional driving context (such as `COMPANY A SPECIAL`) to ensure that the additional, special policy group is applied for data access for Company A only. You would not apply this under the `SUBSCRIBER` policy, because the policy relates only to Company A, and it is more efficient to segregate the basic hosting policy from other policies.

14.3.7.3 Default Policy Group with the SYS_DEFAULT Policy Group

Within a group of security policies, you can designate one security policy to be the default security policy.

This is useful in situations where you partition security policies by application, so that they will be always be in effect. Default security policies enable developers to base security enforcement under all conditions, while partitioning security policies by application (using security groups) enables layering of additional, application-specific security on top of default security policies. To implement default security policies, you add the policy to the `SYS_DEFAULT` policy group.

Policies defined in this group for a particular table, view, or synonym are run with the policy group specified by the driving context. As described earlier, a driving context is an application context that indicates the policy group in effect. The `SYS_DEFAULT` policy group may or may not contain policies. You cannot to drop the `SYS_DEFAULT` policy group. If you do, then Oracle Database displays an error.

If, to the `SYS_DEFAULT` policy group, you add policies associated with two or more objects, then each object will have a separate `SYS_DEFAULT` policy group associated with it. For example, the `emp` table in the `scott` schema has one `SYS_DEFAULT` policy group, and the `dept` table in the `scott` schema has a different `SYS_DEFAULT` policy group associated with it. Think of them as being organized in the tree structure as follows:

```
SYS_DEFAULT
  - policy1 (scott/emp)
  - policy3 (scott/emp)
SYS_DEFAULT
  - policy2 (scott/dept)
```

You can create policy groups with identical names. When you select a particular policy group, its associated schema and object name are displayed in the property sheet on the right side of the screen.

14.3.7.4 Multiple Policies for Each Table, View, or Synonym

You can establish several policies for the same table, view, or synonym.

Suppose, for example, you have a base application for Order Entry, and each division of your company has its own rules for data access. You can add a division-specific policy function to a table without having to rewrite the policy function of the base application.

All policies applied to a table are enforced with `AND` syntax. If you have three policies applied to the `CUSTOMERS` table, then each policy is applied to the table. You can use policy groups and an application context to partition fine-grained access control enforcement so that different policies apply, depending upon which application is accessing data. This eliminates the requirement for development groups to collaborate on policies, and simplifies application development. You can also have a default policy group that is always applicable (for example, to enforce data separated by subscriber in a hosting environment).

14.3.7.5 Validation of the Application Used to Connect to the Database

The package implementing the driving context must correctly validate the application that is being used to connect to the database.

Although Oracle Database checks the call stack to ensure that the package implementing the driving context sets context attributes, inadequate validation can still occur within the package. For example, in applications where database users or enterprise users are known to the database, the user needs the `EXECUTE` privilege on the package that sets the driving context. Consider a user who knows that the `BENEFITS` application enables more liberal access than the `HR` application. The `setctx` procedure (which sets the correct policy group within the driving context) does not perform any validation to determine which application is actually connecting. That is, the procedure does not check either the IP address of the incoming connection (for a three-tier system) or the `proxy_user` attribute of the user session.

This user could pass to the driving context package an argument setting the context to the more liberal `BENEFITS` policy group, and then access the `HR` application instead. Because the `setctx` does no further validation of the application, this user bypasses the more restrictive HR security policy.

By contrast, if you implement proxy authentication with Oracle Virtual Private Database, then you can determine the identity of the middle tier (and the application) that is connecting to the database on behalf of a user. The correct policy will be applied for each application to mediate data access.

For example, a developer using the proxy authentication feature could determine that the application (the middle tier) connecting to the database is `HRAPPSERVER`. The package that implements the driving context can thus verify whether the `proxy_user` in the user session is `HRAPPSERVER`. If so, then it can set the driving context to use the `HR` policy group. If `proxy_user` is not `HRAPPSERVER`, then it can deny access.

In this case, the following query is executed:

```
SELECT * FROM apps.benefit;
```

Oracle Database picks up policies from the default policy group (`SYS_DEFAULT`) and active namespace `HR`. The query is internally rewritten as follows:

```
SELECT * FROM apps.benefit
 WHERE company = SYS_CONTEXT('ID','MY_COMPANY')
 AND SYS_CONTEXT('ID','TITLE') = 'MANAGER';
```

14.3.8 Optimizing Performance by Using Oracle Virtual Private Database Policy Types

You can optimize performance by using the Oracle Virtual Private Database (VPD) the dynamic, static, or shared policy types.

- About Oracle Virtual Private Database Policy Types
  Specifying a policy type for your policies can optimize performance each the Oracle Virtual Private Database policy runs.
- Dynamic Policy Type to Automatically Rerun Policy Functions
  The `DYNAMIC` policy type runs the policy function each time a user accesses the Virtual Private Database-protected database objects.
- Example: Creating a DYNAMIC Policy with DBMS_RLS.ADD_POLICY
  The `DBMS_RLS.ADD_POLICY` procedure can create a dynamic Oracle Virtual Private Database policy.
- Static Policy to Prevent Policy Functions from Rerunning for Each Query
  The static policy type enforces the same predicate for all users in the instance.
- Example: Creating a Static Policy with DBMS_RLS.ADD_POLICY
  The `DBMS_RLS.ADD_POLICY` procedure can create a static Oracle Virtual Private Database (VPD) policy.
- Example: Shared Static Policy to Share a Policy with Multiple Objects
  The `DBMS_RLS.ADD_POLICY` procedure can create a shared static Oracle Virtual Private Database policy to share the policy with multiple objects.
- When to Use Static and Shared Static Policies
  Static policies are ideal when every query requires the same predicate and fast performance is essential, such as hosting environments.
- Context-Sensitive Policy for Application Context Attributes That Change
  Context-sensitive policies are useful when different predicates must be applied depending on which user executes the query.
- Example: Creating a Context-Sensitive Policy with DBMS_RLS.ADD_POLICY
  The `DBMS_RLS.ADD_POLICY` procedure can create an Oracle Virtual Private Database context-sensitive policy.
- Example: Refreshing Cached Statements for a VPD Context-Sensitive Policy
  The `DBMS_RLS.REFRESH_POLICY` statement can refresh cached statements for Oracle Virtual Private Database context-sensitive policies.
- Example: Altering an Existing Context-Sensitive Policy
  The `DBMS_RLS.ALTER_POLICY` procedure can modify an Oracle Virtual Private Database policy.
- Example: Using a Shared Context Sensitive Policy to Share a Policy with Multiple Objects
  The `DBMS_RLS.ADD_POLICY` procedure can create a shared context-sensitive Oracle Virtual Private Database to share a policy that has multiple objects.
- When to Use Context-Sensitive and Shared Context-Sensitive Policies
  Use context-sensitive policies when a predicate does not need to change for a user session, but the policy must enforce multiple predicates for different users or groups.

- Summary of the Five Oracle Virtual Private Database Policy Types
  Oracle Virtual Private Database provides five policy types, based on user needs such as hosting environments.

14.3.8.1 About Oracle Virtual Private Database Policy Types

Specifying a policy type for your policies can optimize performance each the Oracle Virtual Private Database policy runs.

Policy types control how Oracle Database caches Oracle Virtual Private Database policy predicates. Consider setting a policy type for your policies, because the execution of policy functions can use a significant amount of system resources. Minimizing the number of times that a policy function can run optimizes database performance.

You can choose from five policy types: `DYNAMIC`, `STATIC`, `SHARED_STATIC`, `CONTEXT_SENSITIVE`, and `SHARED_CONTEXT_SENSITIVE`. These enable you to precisely specify how often a policy predicate should change. To specify the policy type, set the `policy_type` parameter of the `DBMS_RLS.ADD POLICY` procedure.

14.3.8.2 Dynamic Policy Type to Automatically Rerun Policy Functions

The `DYNAMIC` policy type runs the policy function each time a user accesses the Virtual Private Database-protected database objects.

If you do not specify a policy type in the `DBMS_RLS.ADD_POLICY` procedure, then, by default, your policy will be dynamic. You can specifically configure a policy to be dynamic by setting the `policy_type` parameter of the `DBMS_RLS.ADD_POLICY` procedure to `DYNAMIC`.

This policy type does not optimize database performance as the static and context sensitive policy types do. However, Oracle recommends that before you set policies as either static or context-sensitive, you should first test them as `DYNAMIC` policy types, which run every time. Testing policy functions as `DYNAMIC` policies first enables you to observe how the policy function affects each query, because nothing is cached. This ensures that the functions work properly before you enable them as static or context-sensitive policy types to optimize performance.

You can use the `DBMS_UTILITY.GET_TIME` function to measure the start and end times for a statement to run. For example:

```
-- 1. Get the start time:
SELECT DBMS_UTILITY.GET_TIME FROM DUAL;

  GET_TIME
----------
   2312721

-- 2. Run the statement:
SELECT COUNT(*) FROM HR.EMPLOYEES;

  COUNT(*)
----------
       107

-- 3. Get the end time:
SELECT DBMS_UTILITY.GET_TIME FROM DUAL;

  GET_TIME
----------
   2314319
```

Related Topics

- Auditing Functions, Procedures, Packages, and Triggers
  You can audit functions, procedures, PL/SQL packages, and triggers.

14.3.8.3 Example: Creating a DYNAMIC Policy with DBMS_RLS.ADD_POLICY

The `DBMS_RLS.ADD_POLICY` procedure can create a dynamic Oracle Virtual Private Database policy.

Example 14-5 shows how to create the `DYNAMIC` policy type.

Example 14-5 Creating a DYNAMIC Policy with DBMS_RLS.ADD_POLICY

```
BEGIN
 DBMS_RLS.ADD_POLICY(
  object_schema   => 'hr',
  object_name     => 'employees',
  policy_name     => 'secure_update',
  policy_function => 'hide_fin',
  policy_type     => dbms_rls.DYNAMIC);
END;
/
```

14.3.8.4 Static Policy to Prevent Policy Functions from Rerunning for Each Query

The static policy type enforces the same predicate for all users in the instance.

Oracle Database stores static policy predicates in SGA, so policy functions do not rerun for each query. This results in faster performance.

You can enable static policies by setting the `policy_type` parameter of the `DBMS_RLS.ADD_POLICY` procedure to either `STATIC` or `SHARED_STATIC`, depending on whether or not you want the policy to be shared across multiple objects.

Each execution of the same cursor could produce a different row set for the same predicate, because the predicate may filter the data differently based on attributes such as `SYS_CONTEXT` or `SYSDATE`.

For example, suppose you enable a policy as either a `STATIC` or `SHARED_STATIC` policy type, which appends the following predicate to all queries made against policy protected database objects:

```
WHERE dept = SYS_CONTEXT ('hr_app','deptno')
```

Although the predicate does not change for each query, it applies to the query based on session attributes of the `SYS_CONTEXT`. In the case of the preceding example, the predicate returns only those rows where the department number matches the `deptno` attribute of the `SYS_CONTEXT`, which is the department number of the user who is querying the policy-protected database object.

> **Note:**
>
> When using shared static policies, ensure that the policy predicate does not contain attributes that are specific to a particular database object, such as a column name.

Related Topics

- Auditing Functions, Procedures, Packages, and Triggers
  You can audit functions, procedures, PL/SQL packages, and triggers.

14.3.8.5 Example: Creating a Static Policy with DBMS_RLS.ADD_POLICY

The `DBMS_RLS.ADD_POLICY` procedure can create a static Oracle Virtual Private Database (VPD) policy.

Example 14-6 shows how to create the `STATIC` policy type.

Example 14-6 Creating a Static Policy with DBMS_RLS.ADD_POLICY

```
BEGIN
 DBMS_RLS.ADD_POLICY(
  object_schema   => 'hr',
  object_name     => 'employees',
  policy_name     => 'secure_update',
  policy_function => 'hide_fin',
  policy_type     => DBMS_RLS.STATIC);
END;
/
```

14.3.8.6 Example: Shared Static Policy to Share a Policy with Multiple Objects

The `DBMS_RLS.ADD_POLICY` procedure can create a shared static Oracle Virtual Private Database policy to share the policy with multiple objects.

If, for example, you wanted to apply the static policy that was created earlier to a second table in the `HR` schema that may contain financial data that you want to hide, you could use the `SHARED_STATIC` setting for both tables.

Example 14-7 shows how to set the `SHARED_STATIC` policy type for two tables that share the same policy.

Example 14-7 Creating a Shared Static Policy to Share a Policy with Multiple Objects

```
-- 1. Create a policy for the first table, employees:
BEGIN
 DBMS_RLS.ADD_POLICY(
  object_schema   => 'hr',
  object_name     => 'employees',
  policy_name     => 'secure_update',
  policy_function => 'hide_fin',
  policy_type     => dbms_rls.SHARED_STATIC);
END;
/
-- 2. Create a policy for the second table, fin_data:
BEGIN
 DBMS_RLS.ADD_POLICY(
  object_schema   => 'hr',
  object_name     => 'fin_data',
  policy_name     => 'secure_update',
  policy_function => 'hide_fin',
  policy_type     => dbms_rls.SHARED_STATIC);
END;
/
```

Related Topics

- Example: Creating a Static Policy with DBMS_RLS.ADD_POLICY
  The `DBMS_RLS.ADD_POLICY` procedure can create a static Oracle Virtual Private Database (VPD) policy.

14.3.8.7 When to Use Static and Shared Static Policies

Static policies are ideal when every query requires the same predicate and fast performance is essential, such as hosting environments.

For these situations when the policy function appends the same predicate to every query, rerunning the policy function each time adds unnecessary overhead to the system. For example, consider a data warehouse that contains market research data for customer organizations that are competitors. The warehouse must enforce the policy that each organization can see only their own market research, which is expressed by the following predicate:

```
WHERE subscriber_id = SYS_CONTEXT('customer', 'cust_num')
```

Using `SYS_CONTEXT` for the application context enables the database to dynamically change the rows that are returned. You do not need to rerun the function, so the predicate can be cached in the SGA, thus conserving system resources and improving performance.

14.3.8.8 Context-Sensitive Policy for Application Context Attributes That Change

Context-sensitive policies are useful when different predicates must be applied depending on which user executes the query.

For example, consider the case where managers should have the predicate `WHERE group` set to `managers`, and employees should have the predicate `WHERE empno_ctx` set to `emp_id`. A context-sensitive policy will enable you to present only the information that the managers must see when the managers log in, and only the information that the employees must see when they log in. The policy uses application contexts to determine which predicate to use.

In contrast to static policies, context-sensitive policies do not always cache the predicate. With context-sensitive policies, the database assumes that the predicate will change after statement parse time. But if there is no change in the local application context, then Oracle Database does not rerun the policy function within the user session. If there is a change in any attribute of any application context during the user session, then by default the database re-executes the policy function to ensure that it captures all changes to the predicate since the initial parsing. This results in unnecessary re-executions of the policy function if none of the associated attributes have changed. You can restrict the evaluation to a specific application context by including both the `namespace` and `attribute` parameters.

If you plan to use the `namespace` and `attribute` parameters in your policy, then follow these guidelines:

- Ensure that you specify both `namespace` and `attribute` parameters, not just one.
- Ensure that your policy has the `policy_type` argument set to `DBMS_RLS.CONTEXT_SENSITIVE` or `SHARED_CONTEXT_SENSITIVE`. You cannot use the `namespace` and `attribute` parameters in static or dynamic policies.

If there are no attributes associated with the Virtual Private Database policy function, then Oracle Database evaluates the context-sensitive function for any application context changes.

Shared context-sensitive policies operate in the same way as regular context-sensitive policies, except they can be shared across multiple database objects. For this policy type, all

objects can share the policy function from the UGA, where the predicate is cached until the local session context changes.

Related Topics

- Example: Using a Shared Context Sensitive Policy to Share a Policy with Multiple Objects
  The `DBMS_RLS.ADD_POLICY` procedure can create a shared context-sensitive Oracle Virtual Private Database to share a policy that has multiple objects.
- Tutorial: Implementing a Session-Based Application Context Policy
  This tutorial demonstrates how to create an Oracle Virtual Private Database policy that uses a database session-based application context.
- Tutorial: Implementing an Oracle Virtual Private Database Policy Group
  This tutorial demonstrates how to create an Oracle Virtual Private Database policy group.

14.3.8.9 Example: Creating a Context-Sensitive Policy with DBMS_RLS.ADD_POLICY

The `DBMS_RLS.ADD_POLICY` procedure can create an Oracle Virtual Private Database context-sensitive policy.

Example 14-8shows how to create a `CONTEXT_SENSITIVE` policy in which the policy is evaluated only for changes to the `empno_ctx` namespace and `emp_id` attribute.

Example 14-8 Creating a Context-Sensitive Policy with DBMS_RLS.ADD_POLICY

```
BEGIN
 DBMS_RLS.ADD_POLICY(
  object_schema   => 'hr',
  object_name     => 'employees',
  policy_name     => 'secure_update',
  policy_function => 'hide_fin',
  policy_type     => dbms_rls.CONTEXT_SENSITIVE,
  namespace       => 'empno_ctx',
  attribute       => 'emp_id');
END;
/
```

14.3.8.10 Example: Refreshing Cached Statements for a VPD Context-Sensitive Policy

The `DBMS_RLS.REFRESH_POLICY` statement can refresh cached statements for Oracle Virtual Private Database context-sensitive policies.

Example 14-9 shows you can manually refresh all the cached statements that are associated with a Virtual Private Database context-sensitive policy by running the `DBMS_RLS.REFRESH_POLICY` procedure.

Example 14-9 Refreshing Cached Statements for a VPD Context-Sensitive Policy

```
BEGIN
 DBMS_RLS.REFRESH_POLICY(
  object_schema   => 'hr',
  object_name     => 'employees',
  policy_name     => 'secure_update');
END;
/
```

14.3.8.11 Example: Altering an Existing Context-Sensitive Policy

The `DBMS_RLS.ALTER_POLICY` procedure can modify an Oracle Virtual Private Database policy.

Example 14-10 shows how you can use the `DBMS_RLS.ALTER_POLICY` statement to alter an existing context-sensitive policy so that the `order_update_pol` policy function is executed only if the relevant context attributes change.

Example 14-10 Altering an Existing Context-Sensitive Policy

```
BEGIN
 DBMS_RLS.ALTER_POLICY(
  object_schema   => 'oe',
  object_name     => 'orders',
  policy_name     => 'order_update_pol',
  alter_option    =>  DBMS_RLS.ADD_ATTRIBUTE_ASSOCIATION,
  namespace       => 'empno_ctx',
  attribute       => 'emp_role');
END;
/
```

14.3.8.12 Example: Using a Shared Context Sensitive Policy to Share a Policy with Multiple Objects

The `DBMS_RLS.ADD_POLICY` procedure can create a shared context-sensitive Oracle Virtual Private Database to share a policy that has multiple objects.

Example 14-11 shows how to create two shared context sensitive policies that share a policy with multiple tables, and how to restrict the evaluation only for changes to the `empno_ctx` namespace and `emp_id` attribute.

Example 14-11 Shared Context-Sensitive Policy with DBMS_RLS.ADD_POLICY

```
-- 1. Create a policy for the first table, employees:
BEGIN
 DBMS_RLS.ADD_POLICY(
  object_schema   => 'hr',
  object_name     => 'employees',
  policy_name     => 'secure_update',
  policy_function => 'hide_fin',
  policy_type     => dbms_rls.SHARED_CONTEXT_SENSITIVE,
  namespace       => 'empno_ctx',
  attribute       => 'emp_id');
END;
/
--2. Create a policy for the second table, fin_data:
BEGIN
 DBMS_RLS.ADD_POLICY(
  object_schema   => 'hr',
  object_name     => 'fin_data',
  policy_name     => 'secure_update',
  policy_function => 'hide_fin',
  policy_type     => dbms_rls.SHARED_CONTEXT_SENSITIVE,
  namespace       => 'empno_ctx',
  attribute       => 'emp_id');
END;
/
```

Note the following:

- When using shared context-sensitive policies, ensure that the policy predicate does not contain attributes that are specific to a particular database object, such as a column name.
- To manually refresh all the cached statements that are associated with a Virtual Private Database shared context-sensitive policy, run the `DBMS_RLS.REFRESH_GROUPED_POLICY` procedure.

14.3.8.13 When to Use Context-Sensitive and Shared Context-Sensitive Policies

Use context-sensitive policies when a predicate does not need to change for a user session, but the policy must enforce multiple predicates for different users or groups.

For example, consider a `sales_history` table with a single policy. This policy states that analysts can see only their own products and regional employees can see only their own region. In this case, the database must rerun the policy function each time the type of user changes. The performance gain is realized when a user can log in and issue several DML statements against the protected object without causing the server to rerun the policy function.

> **Note:**
>
> For session pooling where multiple clients share a database session, the middle tier must reset the context during client switches.

14.3.8.14 Summary of the Five Oracle Virtual Private Database Policy Types

Oracle Virtual Private Database provides five policy types, based on user needs such as hosting environments.

Table 14-2 summarizes the types of policy types available.

Table 14-2 DBMS_RLS.ADD_POLICY Policy Types

Policy Types	When the Policy Function Runs	Usage Example	Shared Across Multiple Objects ?
`DYNAMIC`	Policy function re-runs every time a policy-protected database object is accessed.	Applications where policy predicates must be generated for each query, such as time-dependent policies where users are denied access to database objects at certain times during the day	No
`STATIC`	Once, then the predicate is cached in the SGA. Each execution of the same cursor could produce a different row set for the same predicate because the predicate may filter the data differently based on attributes such as `SYS_CONTEXT` or `SYSDATE`.	View replacement	No
`SHARED_STATIC`	Same as `STATIC`	Hosting environments, such as data warehouses where the same predicate must be applied to multiple database objects	Yes

Table 14-2 (Cont.) DBMS_RLS.ADD_POLICY Policy Types

Policy Types	When the Policy Function Runs	Usage Example	Shared Across Multiple Objects ?
CONTEXT_SENSITIVE	• At statement parse time • At statement execution time when the local application context changed since the last use of the cursor	Three-tier, session pooling applications where policies enforce two or more predicates for different users or groups	No
SHARED_CONTEXT_SENSITIVE	First time the object is reference in a database session. Predicates are cached in the private session memory UGA so policy functions can be shared among objects.	Same as CONTEXT_SENSITIVE, but multiple objects can share the policy function from the session UGA	Yes

14.4 Tutorials: Creating Oracle Virtual Private Database Policies

These tutorials show how to create a simple and a database session-based Oracle Virtual Private policy, and how to create policy groups.

- Tutorial: Creating a Simple Oracle Virtual Private Database Policy
  This tutorial shows how to create a simple Oracle Virtual Private Database policy using the OE user account.
- Tutorial: Implementing a Session-Based Application Context Policy
  This tutorial demonstrates how to create an Oracle Virtual Private Database policy that uses a database session-based application context.
- Tutorial: Implementing an Oracle Virtual Private Database Policy Group
  This tutorial demonstrates how to create an Oracle Virtual Private Database policy group.

14.4.1 Tutorial: Creating a Simple Oracle Virtual Private Database Policy

This tutorial shows how to create a simple Oracle Virtual Private Database policy using the OE user account.

- About This Tutorial
  This tutorial shows how to create a VPD policy that limits access to orders created by Sales Representative 159 in the OE.ORDERS table.
- Step 1: Ensure That the OE User Account Is Active
  First, you must ensure that OE user account is active.
- Step 2: Create a Policy Function
  Next, you are ready to create a policy function.
- Step 3: Create the Oracle Virtual Private Database Policy
  After you create the policy function, you are ready to associate it with a VPD policy.
- Step 4: Test the Policy
  After you create the Oracle Virtual Private Database policy, it goes into effect immediately.
- Step 5: Remove the Components of This Tutorial
  If you no longer need the components of this tutorial, then you can remove them.

14.4.1.1 About This Tutorial

This tutorial shows how to create a VPD policy that limits access to orders created by Sales Representative 159 in the `OE.ORDERS` table.

In essence, the policy translates the following statement:

```
SELECT * FROM OE.ORDERS;
```

To the following statement:

```
SELECT * FROM OE.ORDERS WHERE SALES_REP_ID = 159;
```

14.4.1.2 Step 1: Ensure That the OE User Account Is Active

First, you must ensure that `OE` user account is active.

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. Query the `DBA_USERS` data dictionary view to find the account status of `OE`.

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

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

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

   Replace *password* with a password that is secure. For greater security, do not reuse the same password that was used in previous releases of Oracle Database.

Related Topics

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

14.4.1.3 Step 2: Create a Policy Function

Next, you are ready to create a policy function.

- As user `SYS`, create the following function, which will append the `WHERE SALES_REP_ID = 159` clause to any `SELECT` statement on the `OE.ORDERS` table.

```
CREATE OR REPLACE FUNCTION auth_orders(
  schema_var IN VARCHAR2,
  table_var  IN VARCHAR2
 )
 RETURN VARCHAR2
 IS
  return_val VARCHAR2 (400);
 BEGIN
  return_val := 'SALES_REP_ID = 159';
```

```
  RETURN return_val;
 END auth_orders;
/
```

In this example:

- `schema_var` and `table_var` create input parameters to specify to store the schema name, `OE`, and table name, `ORDERS`. First, define the parameter for the schema, and then define the parameter for the object, in this case, a table. Always create them in this order. The Virtual Private Database policy you create will need these parameters to specify the `OE.ORDERS` table.
- `RETURN VARCHAR2` returns the string that will be used for the `WHERE` predicate clause. Remember that return value is always a `VARCHAR2` data type.
- `IS ... RETURN return_val` encompasses the creation of the `WHERE SALES_REP_ID = 159` predicate.

14.4.1.4 Step 3: Create the Oracle Virtual Private Database Policy

After you create the policy function, you are ready to associate it with a VPD policy.

- Create the following policy by using the `ADD_POLICY` procedure in the `DBMS_RLS` package.

```
BEGIN
  DBMS_RLS.ADD_POLICY (
    object_schema    => 'oe',
    object_name      => 'orders',
    policy_name      => 'orders_policy',
    function_schema  => 'sys',
    policy_function  => 'auth_orders',
    statement_types  => 'select'
   );
 END;
/
```

In this example:

- `object_schema => 'oe'` specifies the schema that you want to protect, that is, `OE`.
- `object_name => 'orders'` specifies the object within the schema to protect, that is, the `ORDERS` table.
- `policy_name => 'orders_policy'` names this policy `orders_policy`.
- `function_schema => 'sys'` specifies the schema in which the `auth_orders` function was created. In this example, `auth_orders` was created in the `SYS` schema. But typically, it should be created in the schema of a security administrator.
- `policy_function => 'auth_orders'` specifies a function to enforce the policy. Here, you specify the `auth_orders` function that you created in the preceding step, when you created the policy function.
- `statement_types => 'select'` specifies the operations to which the policy applies. In this example, the policy applies to all `SELECT` statements that the user may perform.

Related Topics

- Step 2: Create a Policy Function
  Next, you are ready to create a policy function.

14.4.1.5 Step 4: Test the Policy

After you create the Oracle Virtual Private Database policy, it goes into effect immediately.

The next time a user, including the owner of the schema, performs a `SELECT` on `OE.ORDERS`, only the orders by Sales Representative 159 will be accessed.

1. Connect as user `OE`.

```
CONNECT oe@pdb_name
Enter password: password
```

2. Enter the following `SELECT` statement:

```
SELECT COUNT(*) FROM ORDERS;
```

   The following output should appear:

```
 COUNT(*)
---------
         7
```

   The policy is in effect for user `OE`: As you can see, only 7 of the 105 rows in the orders table are returned.

   But users with administrative privileges still have access to all the rows in the table.

3. Connect as user `SYS` with the `SYSDBA` administrative privilege.

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

4. Enter the following `SELECT` statement:

```
SELECT COUNT(*) FROM OE.ORDERS;
```

   The following output should appear:

```
 COUNT(*)
---------
       105
```

14.4.1.6 Step 5: Remove the Components of This Tutorial

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

1. As user `SYS` in the PDB in which you created the tutorial components, remove the function and policy as follows:

```
DROP FUNCTION auth_orders;
EXEC DBMS_RLS.DROP_POLICY('OE','ORDERS','ORDERS_POLICY');
```

2. If you need to lock and expire the `OE` account, then enter the following statement:

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

14.4.2 Tutorial: Implementing a Session-Based Application Context Policy

This tutorial demonstrates how to create an Oracle Virtual Private Database policy that uses a database session-based application context.

- About This Tutorial
  This tutorial shows how to use a database session-based application context to implement a policy in which customers see only their own orders.
- Step 1: Create User Accounts and Sample Tables
  First, create user accounts and the sample tables.
- Step 2: Create a Database Session-Based Application Context
  Next, you are ready to create the database session-based application context.
- Step 3: Create a PL/SQL Package to Set the Application Context
  After you create the application context, you are ready to create a package to set the context.
- Step 4: Create a Logon Trigger to Run the Application Context PL/SQL Package
  The logon trigger runs the PL/SQL package procedure so that the next time a user logs on, the application context is set.
- Step 5: Test the Logon Trigger
  The logon trigger sets the application context for the user when the trigger runs the `sysadmin_vpd.orders_ctx_pkg.set_custnum` procedure.
- Step 6: Create a PL/SQL Policy Function to Limit User Access to Their Orders
  The next step is to create a PL/SQL function to control the display of the user's query.
- Step 7: Create the New Security Policy
  Finally, you are ready to create the VPD security policy.
- Step 8: Test the New Policy
  Now that you have created all the components, you are ready to test the policy.
- Step 9: Remove the Components of This Tutorial
  If you no longer need the components of this tutorial, then you can remove them.

14.4.2.1 About This Tutorial

This tutorial shows how to use a database session-based application context to implement a policy in which customers see only their own orders.

In this tutorial, you create the following layers of security:

1. When a user logs on, a database session-based application context checks whether the user is a customer. If a user is not a customer, the user still can log on, but this user cannot access the orders entry table you will create for this example.
2. If the user is a customer, then they can log on. After the customer has logged on, an Oracle Virtual Private Database policy restricts this user to see only their orders.
3. As a further restriction, the Oracle Virtual Private Database policy prevents users from adding, modifying, or removing orders.

14.4.2.2 Step 1: Create User Accounts and Sample Tables

First, create user accounts and the sample tables.

1. Log in to a PDB as a user who has administrative privileges.

```
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. Create the following administrative user, who will administer the Oracle Virtual Private Database policy.

   The following SQL statements create this user and then grant the user the necessary privileges for completing this tutorial.

```
CREATE USER sysadmin_vpd IDENTIFIED BY password CONTAINER = CURRENT;
GRANT CREATE SESSION, CREATE ANY CONTEXT, CREATE PROCEDURE, CREATE TRIGGER, ADMINISTER DATABASE
TRIGGER TO sysadmin_vpd;
GRANT EXECUTE ON DBMS_SESSION TO sysadmin_vpd;
GRANT EXECUTE ON DBMS_RLS TO sysadmin_vpd;
GRANT ADMINISTER ROW LEVEL SECURITY POLICY TO sysadmin_vpd;
```

   Replace *password* with a password that is secure.

3. Create the following local users:

```
CREATE USER tbrooke IDENTIFIED BY password CONTAINER = CURRENT;
CREATE USER owoods IDENTIFIED BY password CONTAINER = CURRENT;

GRANT CREATE SESSION TO tbrooke, owoods;
```

   Replace *password* with a password that is secure.

4. Check the account status of the sample user `SCOTT`, who you will use for this tutorial:

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

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

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

   Replace *password* with a password that is secure. For greater security, do not reuse the same password that was used in previous releases of Oracle Database.

5. Connect as user `SCOTT`.

```
CONNECT SCOTT@pdb_name
Enter password: password
```

6. Create and populate the `customers` table.

```
CREATE TABLE customers (
 cust_no    NUMBER(4),
 cust_email VARCHAR2(20),
 cust_name  VARCHAR2(20));

INSERT INTO customers VALUES (1234, 'TBROOKE', 'Thadeus Brooke');
INSERT INTO customers VALUES (5678, 'OWOODS', 'Oberon Woods');
```

   When you enter the user email IDs, enter them in upper-case letters. Later on, when you create the application context PL/SQL package, the `SESSION_USER` parameter of the `SYS_CONTEXT` function expects the user names to be in upper case. Otherwise, you will be unable to set the application context for the user.

7. User `sysadmin_vpd` will need `SELECT` privileges for the customers table, so as user `SCOTT`, grant him this privilege.

```
GRANT READ ON customers TO sysadmin_vpd;
```

8. Create and populate the `orders_tab` table.

```
CREATE TABLE orders_tab (
  cust_no  NUMBER(4),
  order_no NUMBER(4));

INSERT INTO orders_tab VALUES (1234, 9876);
INSERT INTO orders_tab VALUES (5678, 5432);
INSERT INTO orders_tab VALUES (5678, 4592);
```

9. Users `tbrooke` and `owoods` need to query the `orders_tab` table, so grant them the `READ` object privilege.

```
GRANT READ ON orders_tab TO tbrooke, owoods;
```

At this stage, the two sample customers, `tbrooke` and `owoods`, have a record of purchases in the `orders_tab` order entry table, and if they tried right now, they can see all the orders in this table.

Related Topics

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

14.4.2.3 Step 2: Create a Database Session-Based Application Context

Next, you are ready to create the database session-based application context.

1. Connect as user `sysadmin_vpd`.

```
CONNECT sysadmin_vpd@pdb_name
Enter password: password
```

2. Enter the following statement:

```
CREATE OR REPLACE CONTEXT orders_ctx USING orders_ctx_pkg;
```

This statement creates the `orders_ctx` application context. Remember that even though user `sysadmin_vpd` has created this context and it is associated with the `sysadmin_vpd` schema, the `SYS` schema owns the application context.

14.4.2.4 Step 3: Create a PL/SQL Package to Set the Application Context

After you create the application context, you are ready to create a package to set the context.

- As user `sysadmin_vpd`, create the following PL/SQL package, which will set the database session-based application context when the customers `tbrooke` and `owoods` log onto their accounts.

```
CREATE OR REPLACE PACKAGE orders_ctx_pkg IS
  PROCEDURE set_custnum;
 END;
/
CREATE OR REPLACE PACKAGE BODY orders_ctx_pkg IS
  PROCEDURE set_custnum
  AS
    custnum NUMBER;
```

```
  BEGIN
     SELECT cust_no INTO custnum FROM SCOTT.CUSTOMERS
        WHERE cust_email = SYS_CONTEXT('USERENV', 'SESSION_USER');
     DBMS_SESSION.SET_CONTEXT('orders_ctx', 'cust_no', custnum);
  EXCEPTION
   WHEN NO_DATA_FOUND THEN NULL;
  END set_custnum;
END;
/
```

In this example:

- `custnum NUMBER` creates the `custnum` variable, which will hold the customer ID.
- `SELECT cust_no INTO custnum` performs a `SELECT` statement to copy the customer ID that is stored in the `cust_no` column data from the `scott.customers` table into the `custnum` variable.
- `WHERE cust_email = SYS_CONTEXT('USERENV', 'SESSION_USER')` uses a `WHERE` clause to find all the customer IDs that match the user name of the user who is logging on.
- `DBMS_SESSION.SET_CONTEXT('orders_ctx', 'cust_no', custnum)` sets the `orders_ctx` application context values by creating the `cust_no` attribute and then setting it to the value stored in the `custnum` variable.
- `EXCEPTION ... WHEN` adds a `WHEN NO_DATA_FOUND` system exception to catch any `no data found` errors that may result from the `SELECT` statement in the `SELECT cust_no INTO custnum ...` statement.

To summarize, the `sysadmin_vpd.set_custnum` procedure identifies whether or not the session user is a registered customer by attempting to select the user's customer ID into the `custnum` variable. If the user is a registered customer, then Oracle Database sets an application context value for this user. The policy function uses the context value to control the access a user has to data in the `orders_tab` table.

14.4.2.5 Step 4: Create a Logon Trigger to Run the Application Context PL/SQL Package

The logon trigger runs the PL/SQL package procedure so that the next time a user logs on, the application context is set.

- As user `sysadmin_vpd`, create the following logon trigger:

```
CREATE TRIGGER set_custno_ctx_trig AFTER LOGON ON DATABASE
 BEGIN
  sysadmin_vpd.orders_ctx_pkg.set_custnum;
 END;
/
```

Related Topics

- Logon Triggers to Run a Database Session Application Context Package
  Users must run database session application context package after when they log in to the database instance.

14.4.2.6 Step 5: Test the Logon Trigger

The logon trigger sets the application context for the user when the trigger runs the `sysadmin_vpd.orders_ctx_pkg.set_custnum` procedure.

1. Connect as user `tbrooke`.

```
CONNECT tbrooke@pdb_name
Enter password: password
```

2. Run the following query:

```
SELECT SYS_CONTEXT('orders_ctx', 'cust_no') custnum FROM DUAL;
```

The following output should appear:

```
EMP_ID
----------------------------------------------------------------
1234
```

14.4.2.7 Step 6: Create a PL/SQL Policy Function to Limit User Access to Their Orders

The next step is to create a PL/SQL function to control the display of the user's query.

When the user who has logged in performs a `SELECT * FROM SCOTT.ORDERS_TAB` query, the function should cause the output to display only the orders of that user.

1. Connect as user `sysadmin_vpd`.

```
CONNECT sysadmin_vpd@pdb_name
Enter password: password
```

2. Create the following function:

```
CREATE OR REPLACE FUNCTION get_user_orders(
  schema_p   IN VARCHAR2,
  table_p    IN VARCHAR2)
 RETURN VARCHAR2
 AS
  orders_pred VARCHAR2 (400);
 BEGIN
  orders_pred := 'cust_no = SYS_CONTEXT(''orders_ctx'', ''cust_no'')';
 RETURN orders_pred;
END;
/
```

This function creates and returns a `WHERE` predicate that translates to "where the orders displayed belong to the user who has logged in." It then appends this `WHERE` predicate to any queries this user may run against the `scott.orders_tab` table. Next, you are ready to create an Oracle Virtual Private Database policy that applies this function to the `orders_tab` table.

14.4.2.8 Step 7: Create the New Security Policy

Finally, you are ready to create the VPD security policy.

- As user `sysadmin_vpd`, use the `DBMS_RLS.ADD_POLICY` procedure to create the policy as follows:

```
BEGIN
 DBMS_RLS.ADD_POLICY (
```

```
  object_schema    => 'scott',
  object_name      => 'orders_tab',
  policy_name      => 'orders_policy',
  function_schema  => 'sysadmin_vpd',
  policy_function  => 'get_user_orders',
  statement_types  => 'select',
  policy_type      => DBMS_RLS.CONTEXT_SENSITIVE,
  namespace        => 'orders_ctx',
  attribute        => 'cust_no');
END;
/
```

This statement creates a policy named `orders_policy` and applies it to the `orders_tab` table, which customers will query for their orders, in the `SCOTT` schema. The `get_user_orders` function implements the policy, which is stored in the `sysadmin_vpd` schema. The policy further restricts users to issuing `SELECT` statements only. The `namespace` and `attribute` parameters specify the application context that you created earlier.

14.4.2.9 Step 8: Test the New Policy

Now that you have created all the components, you are ready to test the policy.

1. Connect as user `tbrooke`.

   ```
   CONNECT tbrooke@pdb_name
   Enter password: password
   ```

   User `tbrooke` can log on because he has passed the requirements that you defined in the application context.

2. As user `tbrooke`, access your purchases.

   ```
   SELECT * FROM SCOTT.ORDERS_TAB;
   ```

   The following output should appear:

   ```
      CUST_NO    ORDER_NO
   ----------  ----------
         1234        9876
   ```

   User `tbrooke` has passed the second test. This user can access their own orders in the `scott.orders_tab` table.

3. Connect as user `owoods`, and then access your purchases.

   ```
   CONNECT owoods@pdb_name
   Enter password: password

   SELECT * FROM SCOTT.ORDERS_TAB
   ```

   The following output should appear:

   ```
      CUST_NO    ORDER_NO
   ----------  ----------
         5678        5432
         5678        4592
   ```

   As with user `tbrooke`, user `owoods` can log on and see a listing of their own orders.

Note the following:

- You can create several predicates based on the position of a user. For example, a sales representative would be able to see records only for their customers, and an order entry

clerk would be able to see any customer order. You could expand the `custnum_sec` function to return different predicates based on the user position context value.

- The use of an application context in a fine-grained access control package effectively gives you a bind variable in a parsed statement. For example:

```
SELECT * FROM SCOTT.ORDERS_TAB
WHERE cust_no = SYS_CONTEXT('order_entry', 'cust_num');
```

This is fully parsed and optimized, but the evaluation of the `cust_num` attribute value of the user for the `order_entry` context takes place at run-time. This means that you get the benefit of an optimized statement that executes differently for each user who issues the statement.

> **Note:**
>
> You can improve the performance of the function in this tutorial by indexing `cust_no`.

- You can set context attributes based on data from a database table or tables, or from a directory server using Lightweight Directory Access Protocol (LDAP).

Related Topics

- *Oracle Database PL/SQL Language Reference*

14.4.2.10 Step 9: Remove the Components of This Tutorial

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

1. Connect as user `SCOTT`.

```
CONNECT SCOTT@pdb_name
Enter password: password
```

2. Remove the `orders_tab` and `customers` tables.

```
DROP TABLE ORDERS_TAB;
DROP TABLE customers;
```

3. Connect as user `SYS`, connecting with `AS SYSDBA`.

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

4. Run the following statements to drop the components for this tutorial:

```
DROP CONTEXT orders_ctx;
DROP USER sysadmin_vpd CASCADE;
DROP USER tbrooke;
DROP USER owoods;
```

14.4.3 Tutorial: Implementing an Oracle Virtual Private Database Policy Group

This tutorial demonstrates how to create an Oracle Virtual Private Database policy group.

14.4.3.1 About This Tutorial

This tutorial shows how you can use Oracle Virtual Private Database (VPD) to create a policy group.

A policy group enables you to group a set of policies for use in an application. When a nondatabase user logs onto the application, Oracle Database grants the user access based on the policies defined within the appropriate policy group.

For column-level access control, every column or set of hidden columns is controlled by one policy. In this tutorial, you must hide two sets of columns. So, you must create two policies, one for each set of columns that you want to hide. You only want one policy for each user; the driving application context separates the policies for you.

Related Topics

- Oracle Virtual Private Database Policy Groups
  An Oracle Virtual Private Database policy group is a named collection of VPD policies that can be applied to an application.

14.4.3.2 Step 1: Create User Accounts and Other Components for This Tutorial

First, you must create user accounts and tables for this tutorial, and grant the appropriate privileges.

1. Log on to the appropriate PDB as user `SYS` with the `SYSDBA` administrative privilege.

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

To find the available PDBs, run the `show pdbs` command. To check the current PDB, run the `show con_name` command.

2. Create the following local users:

```
CREATE USER apps_user IDENTIFIED BY password CONTAINER = CURRENT;
GRANT CREATE SESSION TO apps_user;
CREATE USER sysadmin_pg  IDENTIFIED BY password CONTAINER = CURRENT;
GRANT CREATE SESSION, CREATE PROCEDURE, CREATE ANY CONTEXT TO sysadmin_pg;
```

Replace *password* with a password that is secure.

3. Grant the following additional privileges to user `sysadmin_pg`:

```
GRANT EXECUTE ON DBMS_RLS TO sysadmin_pg;
GRANT ADMINISTER ROW LEVEL SECURITY POLICY TO sysadmin_pg;
```

4. Log on as user `OE`.

```
CONNECT OE@ pdb_name
Enter password: password
```

If the `OE` account is locked and expired, then reconnect as user `SYS` with the `SYSDBA` administrative privilege and enter the following statement to unlock the account and give it s new password:

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

Replace *password* with a password that is secure. For greater security, do not reuse the same password that was used in previous releases of Oracle Database.

5. Create the `product_code_names` table:

```
CREATE TABLE product_code_names(
group_a     varchar2(32),
year_a      varchar2(32),
group_b     varchar2(32),
year_b      varchar2(32));
```

6. Insert some values into the `product_code_names` table:

```
INSERT INTO product_code_names values('Biffo','2008','Beffo','2004');
INSERT INTO product_code_names values('Hortensia','2008','Bunko','2008');
INSERT INTO product_code_names values('Boppo','2006','Hortensia','2003');

COMMIT;
```

7. Grant the `apps_user` user `SELECT` privileges on the `product_code_names` table.

```
GRANT SELECT ON product_code_names TO apps_user;
```

Related Topics

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

14.4.3.3 Step 2: Create the Two Policy Groups

Next, you must create a policy group for each of the two nondatabase users, `provider_a` and `provider_b`.

1. Connect as user `sysadmin_pg`.

```
CONNECT sysadmin_pg@ pdb_name
Enter password: password
```

2. Create the `provider_a_group` policy group, to be used by user `provider_a`:

```
BEGIN
 DBMS_RLS.CREATE_POLICY_GROUP(
 object_schema   => 'oe',
 object_name     => 'product_code_names',
 policy_group    => 'provider_a_group');
END;
/
```

3. Create the `provider_b_group` policy group, to be used by user `provider_b`:

```
BEGIN
 DBMS_RLS.CREATE_POLICY_GROUP(
 object_schema   => 'oe',
 object_name     => 'product_code_names',
 policy_group    => 'provider_b_group');
END;
/
```

14.4.3.4 Step 3: Create PL/SQL Functions to Control the Policy Groups

A policy group must have a function that defines how the application can control data access for users.

The function that you will create for this policy group applies to users `provider_a` and `provider_b`.

1. Create the `vpd_function_provider_a` function, which restricts the data accessed by user `provider_a`.

```
CREATE OR REPLACE FUNCTION vpd_function_provider_a
 (schema in varchar2, tab in varchar2) return varchar2 as
  predicate  varchar2(8) default NULL;
  BEGIN
   IF LOWER(SYS_CONTEXT('USERENV','CLIENT_IDENTIFIER')) = 'provider_a'
    THEN predicate := '1=2';
   ELSE NULL;
  END IF;
  RETURN predicate;
END;
/
```

   This function checks that the user logging in is really user `provider_a`. If this is true, then only the data in the `product_code_names` table columns `group_a` and `year_a` will be visible to `provider_a`. Data in columns `group_b` and `year_b` will not appear for `provider_a`. This works as follows: Setting `predicate := '1=2'` hides the relevant columns. In a later step, you will specify these columns in the `SEC_RELEVANT_COLS` parameter.

2. Create the `vpd_function_provider_b`, function, which restricts the data accessed by user `provider_b`.

```
CREATE OR REPLACE FUNCTION vpd_function_provider_b
 (schema in varchar2, tab in varchar2) return varchar2 as
  predicate  varchar2(8) default NULL;
  BEGIN
   IF LOWER(SYS_CONTEXT('USERENV','CLIENT_IDENTIFIER')) = 'provider_b'
    THEN predicate := '1=2';
   ELSE NULL;
  END IF;
  RETURN predicate;
```

```
END;
/
```

Similar to the `vpd_function_provider_a` function, this function checks that the user logging in is really user `provider_b`. If this is true, then only the data in the columns `group_b` and `year_b` will be visible to `provider_b`, with data in the `group_a` and `year_a` not appearing for `provider_b`. Similar to the `vpd_function_provider_a` function, `predicate := '1=2'` hides the relevant columns that will be specified in the `SEC_RELEVANT_COLS` parameter.

Related Topics

- Function to Generate the Dynamic WHERE Clause
  The Oracle Virtual Private Database (VPD) function defines the restrictions that you want to enforce.

14.4.3.5 Step 4: Create the Driving Application Context

The application context determines which policy the nondatabase user who is the logging on should use.

1. As user `sysadmin_pg`, create the driving application context as follows:

```
CREATE OR REPLACE CONTEXT provider_ctx USING provider_package;
```

2. Create the PL/SQL `provider_package` package for the application context.

```
CREATE OR REPLACE PACKAGE provider_package IS
 PROCEDURE set_provider_context (policy_group varchar2 default NULL);
END;
/
CREATE OR REPLACE PACKAGE BODY provider_package AS
 PROCEDURE set_provider_context (policy_group varchar2 default NULL) IS
 BEGIN
  CASE LOWER(SYS_CONTEXT('USERENV', 'CLIENT_IDENTIFIER'))
   WHEN 'provider_a' THEN
    DBMS_SESSION.SET_CONTEXT('provider_ctx','policy_group','PROVIDER_A_GROUP');
   WHEN 'provider_b' THEN
    DBMS_SESSION.SET_CONTEXT('provider_ctx','policy_group','PROVIDER_B_GROUP');
  END CASE;
 END set_provider_context;
END;
/
```

3. Associate the `provider_ctx` application context with the `product_code_names` table, and then provide a name.

```
BEGIN
 DBMS_RLS.ADD_POLICY_CONTEXT(
 object_schema  =>'oe',
 object_name    =>'product_code_names',
 namespace      =>'provider_ctx',
 attribute      =>'policy_group');
END;
/
```

4. Grant the `apps_user` account the `EXECUTE` privilege for the `provider_package` package.

```
GRANT EXECUTE ON provider_package TO apps_user;
```

14.4.3.6 Step 5: Add the PL/SQL Functions to the Policy Groups

Now that you have created the necessary functions, you are ready to associate them with their appropriate policy groups.

1. Add the `vpd_function_provider_a` function to the `provider_a_group` policy group.

```
BEGIN
 DBMS_RLS.ADD_GROUPED_POLICY(
 object_schema         => 'oe',
 object_name           => 'product_code_names',
 policy_group          => 'provider_a_group',
 policy_name           => 'filter_provider_a',
 function_schema       => 'sysadmin_pg',
 policy_function       => 'vpd_function_provider_a',
 statement_types       => 'select',
 policy_type           => DBMS_RLS.CONTEXT_SENSITIVE,
 sec_relevant_cols     => 'group_b,year_b',
 sec_relevant_cols_opt => DBMS_RLS.ALL_ROWS,
 namespace             => 'provider_ctx',
 attribute             => 'provider_group');
END;
/
```

   The `group_b` and `year_b` columns specified in the `sec_relevant_cols` parameter are hidden from user `provider_a`.

2. Add the `vpd_function_provider_b` function to the `provider_b_group` policy group.

```
BEGIN
 DBMS_RLS.ADD_GROUPED_POLICY(
 object_schema         => 'oe',
 object_name           => 'product_code_names',
 policy_group          => 'provider_b_group',
 policy_name           => 'filter_provider_b',
 function_schema       => 'sysadmin_pg',
 policy_function       => 'vpd_function_provider_b',
 statement_types       => 'select',
 policy_type           => DBMS_RLS.CONTEXT_SENSITIVE,
 sec_relevant_cols     => 'group_a,year_a',
 sec_relevant_cols_opt => DBMS_RLS.ALL_ROWS,
 namespace             => 'provider_ctx',
 attribute             => 'provider_group');
END;
/
```

   The `group_a` and `year_a` columns specified in the `sec_relevant_cols` parameter are hidden from user `provider_b`.

14.4.3.7 Step 6: Test the Policy Groups

Now you are ready to test the two policy groups.

1. Connect as user `apps_user` and then enter the following statements to ensure that the output you will create later on is nicely formatted.

```
CONNECT apps_user@pdb_name
Enter password: password

col group_a format a16
```

```
col group_b format a16;
col year_a format a16;
col year_b format a16;
```

2. Set the session identifier to `provider_a`.

```
EXEC DBMS_SESSION.SET_IDENTIFIER('provider_a');
```

Here, the application sets the identifier. Setting the identifier to `provider_a` sets the `apps_user` user to a user who should only see the products available to products in the `provider_a_group` policy group.

3. Run the `provider_package` to set the policy group based on the context.

```
EXEC sysadmin_pg.provider_package.set_provider_context;
```

At this stage, you can check the application context was set, as follows:

```
SELECT SYS_CONTEXT('USERENV', 'CLIENT_IDENTIFIER') AS END_USER FROM DUAL;
```

The following output should appear:

```
END_USER
----------------
provider_a
```

4. Enter the following `SELECT` statement:

```
SELECT * FROM oe.product_code_names;
```

The following output should appear:

```
GROUP_A          YEAR_A           GROUP_B          YEAR_B
---------------- ---------------- ---------------- ----------------
Biffo            2008
Hortensia        2008
Boppo            2006
```

5. Set the client identifier to `provider_b` and then enter the following statements:

```
EXEC DBMS_SESSION.SET_IDENTIFIER('provider_b');
EXEC sysadmin_pg.provider_package.set_provider_context;
SELECT * FROM oe.product_code_names;
```

The following output should appear:

```
GROUP_A          YEAR_A           GROUP_B          YEAR_B
---------------- ---------------- ---------------- ----------------
                                  Beffo            2004
                                  Bunko            2008
                                  Hortensia        2003
```

14.4.3.8 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 as user `OE`.

```
CONNECT OE@ pdb_name
Enter password: password
```

2. Drop the `product_code_names` table.

```
DROP TABLE product_code_names;
```

3. Connect as user `SYS` with the `SYSDBA` administrative privilege.

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

4. Drop the application context and users for this tutorial.

```
DROP CONTEXT provider_ctx;
DROP USER sysadmin_pg cascade;
DROP USER apps_user;
```

14.5 How Oracle Virtual Private Database Works with Other Oracle Features

You should be aware of the impact of using Oracle Virtual Private Database with other Oracle features.

- Oracle Virtual Private Database Policies with Editions
  You should be aware of how to use Oracle VPD with editions.
- SELECT FOR UPDATE Statement in User Queries on VPD-Protected Tables
  As a general rule, users should not include the `FOR UPDATE` clause when querying Virtual Private Database-protected tables.
- Oracle Virtual Private Database Policies and Outer or ANSI Joins
  Oracle Virtual Private Database rewrites SQL by using dynamic views.
- Oracle Virtual Private Database Security Policies and Applications
  An Oracle Virtual Private Database security policy is applied within the database itself, rather than within an application.
- Automatic Reparsing for Fine-Grained Access Control Policies Functions
  Queries against objects enabled with fine-grained access control run the policy function so that the most current predicate is used for each policy.
- Oracle Virtual Private Database Policies and Flashback Queries
  Operations on the database use the most recently committed data available.
- Oracle Virtual Private Database and Oracle Label Security
  You can use Oracle Virtual Private Database with Oracle Label Security, but be aware of security exceptions.
- Export of Data Using the EXPDP Utility access_method Parameter
  Be aware if you try to export data from objects that have VPD policies defined on them.
- Oracle Virtual Private Database Policies and Oracle Flashback Time Travel
  Oracle Virtual Private Database policies do not automatically work with Oracle Flashback Time Travel.
- User Models and Oracle Virtual Private Database
  You can use Oracle Virtual Private Database in several types of user models.
- Oracle Virtual Private Database and JSON
  You should be aware of how to use Oracle VPD with JSON.

14.5.1 Oracle Virtual Private Database Policies with Editions

You should be aware of how to use Oracle VPD with editions.

If you are preparing an application for edition-based redefinition, and you cover each table that the application uses with an editioning view, then you must move the Virtual Private Database polices that protect these tables to the editioning view.

When an editioned object has a Virtual Private Database policy, then it applies in all editions in which the object is visible. When an editioned object is actualized, any VPD policies that are attached to it are newly attached to the new actual occurrence. When you newly apply a VPD policy to an inherited editioned object, this action will actualize it.

Related Topics

- *Oracle Database Development Guide*

14.5.2 SELECT FOR UPDATE Statement in User Queries on VPD-Protected Tables

As a general rule, users should not include the `FOR UPDATE` clause when querying Virtual Private Database-protected tables.

The Virtual Private Database technology depends on rewriting the user's query against an inline view that includes the VPD predicate generated by the VPD policy function. Because of this, the same limitations on views also apply to VPD-protected tables. If a user's query against a VPD-protected table includes the `FOR UPDATE` clause in a `SELECT` statement, in most cases, the query may not work. However, the user's query may work in some situations if the inline view generated by VPD is sufficiently simple.

Related Topics

- *Oracle Database SQL Language Reference*

14.5.3 Oracle Virtual Private Database Policies and Outer or ANSI Joins

Oracle Virtual Private Database rewrites SQL by using dynamic views.

For SQL that contains outer join or ANSI operations, some views may not merge and some indexes may not be used. This problem is a known optimization limitation. To remedy this problem, rewrite the SQL to not use outer joins or ANSI operations.

14.5.4 Oracle Virtual Private Database Security Policies and Applications

An Oracle Virtual Private Database security policy is applied within the database itself, rather than within an application.

Hence, a user trying to access data by using a different application cannot bypass the Oracle Virtual Private Database security policy. Another advantage of creating the security policy in the database is that you maintain it in one central place, rather than maintaining individual security policies in multiple applications. Oracle Virtual Private Database provides stronger security than application-based security, at a lower cost of ownership.

You may want to enforce different security policies depending on the application that is accessing data. Consider a situation in which two applications, Order Entry and Inventory, both access the `orders` table. You may want to have the Inventory application use a policy that limits access based on type of product. At the same time, you may want to have the Order Entry application use a policy that limits access based on customer number.

In this case, you must partition the use of fine-grained access by application. Otherwise, both policies would be automatically concatenated together, which may not be the result that you want. You can specify two or more policy groups, and a driving application context that determines which policy group is in effect for a given transaction. You can also designate default policies that always apply to data access. In a hosted application, for example, data access should be limited by subscriber ID.

Related Topics

- Tutorial: Implementing an Oracle Virtual Private Database Policy Group
  This tutorial demonstrates how to create an Oracle Virtual Private Database policy group.

14.5.5 Automatic Reparsing for Fine-Grained Access Control Policies Functions

Queries against objects enabled with fine-grained access control run the policy function so that the most current predicate is used for each policy.

For example, in the case of a time-based policy function, in which queries are only allowed between 8:00 a.m. and 5:00 p.m., a cursor execution parsed at noon runs the policy function at that time, ensuring that the policy is consulted again for the query. Even if the curser was parsed at 9 a.m., when it runs later on (for example, at noon), then the Virtual Private Database policy function runs again to ensure that the execution of the cursor is still permitted at the current time (noon). This ensures that the security check it must perform is the most recent.

Automatic re-execution of the Virtual Private Database policy function does not occur when you set the `DBMS_RLS.ADD_POLICY` setting `STATIC_POLICY` to `TRUE` while adding the policy. This setting causes the policy function to return the same predicate.

14.5.6 Oracle Virtual Private Database Policies and Flashback Queries

Operations on the database use the most recently committed data available.

The flashback query feature enables you to query the database at some point in the past.

To write an application that uses flashback query, you can use the `AS OF` clause in SQL queries to specify either a time or a system change number (SCN), and then query against the committed data from the specified time. You can also use the `DBMS_FLASHBACK` PL/SQL package, which requires more code, but enables you to perform multiple operations, all of which refer to the same point in time.

However, if you use flashback query against a database object that is protected with Oracle Virtual Private Database policies, then the current policies are applied to the old data. Applying the current Oracle Virtual Private Database policies to flashback query data is more secure because it reflects the most current business policy.

Related Topics

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

14.5.7 Oracle Virtual Private Database and Oracle Label Security

You can use Oracle Virtual Private Database with Oracle Label Security, but be aware of security exceptions.

- Using Oracle Virtual Private Database to Enforce Oracle Label Security Policies
  Oracle Virtual Private Database policies provide column or row-level access control based on Oracle Label Security user authorizations.
- Oracle Virtual Private Database and Oracle Label Security Exceptions
  Be aware of the security exceptions when you use Oracle Virtual Private Database, Oracle Label Security, and Oracle Real Application Security.

14.5.7.1 Using Oracle Virtual Private Database to Enforce Oracle Label Security Policies

Oracle Virtual Private Database policies provide column or row-level access control based on Oracle Label Security user authorizations.

You must perform the following actions:

1. When you create the Oracle Label Security policy, do not apply the policy to the table that you want to protect. (The Virtual Private Database policy that you create handles this for you.) In the `SA_SYSDBA.CREATE_POLICY` procedure, set the `default_options` parameter to `NO_CONTROL`.
2. Create the Oracle Label Security label components and authorize users as you normally would.
3. When you create the Oracle Virtual Private Database policy, do the following:
   - In the PL/SQL function you create for the policy, use the Oracle Label Security `DOMINATES` function to compare the authorization of the user with the label that you created. The `DOMINATES` function determines if the user authorization is equal to, or if it is more sensitive than, the label used in the comparison. If the user authorization passes, then the user is granted access to the column. Otherwise, the user is denied access.
   - In the Virtual Private Database policy definition, apply this function to the table that you want to protect. In the `DBMS_RLS.ADD_POLICY` procedure, use the sensitive column (`SEC_RELEVANT_COLS` parameter) and column masking (`SEC_RELEVANT_COLS_OPT` parameter) functionality to show or hide columns based on Oracle Label Security user authorizations.

Related Topics

- *Oracle Label Security Administrator's Guide*

14.5.7.2 Oracle Virtual Private Database and Oracle Label Security Exceptions

Be aware of the security exceptions when you use Oracle Virtual Private Database, Oracle Label Security, and Oracle Real Application Security.

These security exceptions are as follows:

- **When you are exporting data, Oracle Virtual Private Database and Oracle Label Security policies are not enforced during a direct path export operation.** In a direct path export operation, Oracle Database reads data from disk into the buffer cache and transfers rows directly to the Export client.
- **You cannot apply Oracle Virtual Private Database policies and Oracle Label Security policies to objects in the SYS schema.** The `SYS` user and users making a DBA-privileged connection to the database (for example, `CONNECT/AS SYSDBA`) do not have Oracle Virtual Private Database or Oracle Label Security policies applied to their actions. The database user `SYS` is thus always exempt from Oracle Virtual Private Database or Oracle Label Security enforcement, regardless of the export mode, application, or utility used to extract data from the database.

  However, you can audit `SYSDBA` actions by enabling auditing upon installation and specifying that this audit trail be stored in a secure location in the operating system. You can also closely monitor the `SYS` user by using Oracle Database Vault.

- **Database users who were granted the `EXEMPT ACCESS POLICY` system privilege, either directly or through a database role, are exempt from Oracle Virtual Private Database, Label Security, and Real Application Security policy enforcements.** The system privilege `EXEMPT ACCESS POLICY` allows a user to be exempted from all fine-grained access control policies on any `SELECT` or DML operation (`INSERT`, `UPDATE`, and `DELETE`). This provides ease of use for administrative activities, such as installation and import and export of the database, through a non-`SYS` schema.

  However, the following policy enforcement options remain in effect even when `EXEMPT ACCESS POLICY` is granted:

  – `INSERT_CONTROL`, `UPDATE_CONTROL`, `DELETE_CONTROL`, `WRITE_CONTROL`, `LABEL_UPDATE`, and `LABEL_DEFAULT`

  – If the Oracle Label Security policy specifies the `ALL_CONTROL` option, then all enforcement controls are applied except `READ_CONTROL` and `CHECK_CONTROL`.

  Because `EXEMPT ACCESS POLICY` negates the effect of fine-grained access control, you should only grant this privilege to users who have legitimate reasons for bypassing fine-grained access control enforcement. Do not grant this privilege using the `WITH ADMIN OPTION`. If you do, users could pass the `EXEMPT ACCESS POLICY` privilege to other users, and thus propagate the ability to bypass fine-grained access control.

> **Note:**
>
> - The `EXEMPT ACCESS POLICY` system privilege does not affect the enforcement of object privileges such as `SELECT`, `INSERT`, `UPDATE`, and `DELETE`. These privileges are enforced even if a user was granted the `EXEMPT ACCESS POLICY` system privilege.
> - The `SYS_CONTEXT` values that Oracle Virtual Private Database uses are not propagated to secondary databases for failover.

Related Topics

- *Oracle Database Utilities*

14.5.8 Export of Data Using the EXPDP Utility access_method Parameter

Be aware if you try to export data from objects that have VPD policies defined on them.

If you try to use the Oracle Data Pump Export (`EXPDP`) utility with the `access_method` parameter set to `direct_path` to export data from a schema that contains an object that has a Virtual Private Database policy defined on it, then an `ORA-31696` error message may appear and the export operation will fail.

The error message is as follows:

```
ORA-31696: unable to export/import TABLE_DATA:"schema.table" using client specified
DIRECT_PATH method
```

This problem occurs when you perform a schema-level export or a full database export, which requires the `EXP_FULL_DATABASE` role. To perform an export with VPD policies in place using the `access_method=direct_path` parameter, the exporting user must be granted the system privilege `EXEMPT ACCESS POLICY`. `EXEMPT ACCESS POLICY` bypasses Virtual Private Database

policies. Note that the `EXP_FULL_DATABASE` role does **not** include the `EXEMPT ACCESS POLICY` system privilege.

To find the underlying problem, try the `EXPDP` invocation again, but do not set the `access_method` parameter to `direct_path`. Instead, use either `automatic` or `external_table`. The underlying problem could be a permissions problem, for example:

```
ORA-39181: Only partial table data may be exported due to fine grain access control on
"schema_name"."object_name"
```

14.5.9 Oracle Virtual Private Database Policies and Oracle Flashback Time Travel

Oracle Virtual Private Database policies do not automatically work with Oracle Flashback Time Travel.

After you create an Oracle Virtual Private Database (VPD) policy for a table, consider creating an equivalent policy for the Flashback Archive history table. The following example demonstrates how to do so.

Example 14-12 Creating an Equivalent Policy for an Flashback Archive History Table

1. Create a temporary VPD administrative user.

```
CREATE USER sysadmin_vpd IDENTIFIED BY password CONTAINER = CURRENT;
GRANT CREATE SESSION, CREATE ANY CONTEXT, CREATE PROCEDURE TO sysadmin_vpd;
GRANT EXECUTE ON DBMS_SESSION TO sysadmin_vpd;
GRANT EXECUTE ON DBMS_FLASHBACK, DBMS_FLASHBACK_ARCHIVE TO sysadmin_vpd;
GRANT EXECUTE ON DBMS_RLS TO sysadmin_vpd;
GRANT UPDATE ON SCOTT.EMP TO sysadmin_vpd;
```

2. Connect to the PDB as the `sysadmin_vpd` user.

```
connect sysadmin_vpd@pdb_name
Enter password: password
Connected.
```

3. Create the VPD function.
   For example, the following function shows only rows with department number (`deptno`) 30 to users other than user `SCOTT`:

```
CREATE OR REPLACE FUNCTION emp_policy_func (
 v_schema IN VARCHAR2,
 v_objname IN VARCHAR2)

RETURN VARCHAR2 AS
condition VARCHAR2 (200);

BEGIN
 condition := 'deptno=30';
 IF sys_context('userenv', 'session_user') IN ('SCOTT') THEN
   RETURN NULL;
 ELSE
   RETURN (condition);
 END IF;
```

```
END emp_policy_func;
/
```

4. Create the following VPD procedure to attach the `emp_policy_func` function to the `SCOTT.EMP` table.

```
BEGIN
 DBMS_RLS.ADD_POLICY (
  object_schema     => 'scott',
  object_name       => 'emp',
  policy_name       => 'emp_policy',
  function_schema   => 'sysadmin_vpd',
  policy_function   => 'emp_policy_func',
  policy_type       =>  dbms_rls.dynamic);
END;
/
```

5. Create the following `test` user and grant privileges, including those related to Flashback Archive.

```
CREATE USER test IDENTIFIED BY password;
GRANT CREATE SESSION TO test;
GRANT CONNECT, RESOURCE TO test;
GRANT SELECT ON SCOTT.EMP TO test;
GRANT FLASHBACK ARCHIVE ON ftest TO test;
GRANT EXECUTE ON DBMS_FLASHBACK_ARCHIVE TO test;
GRANT EXECUTE ON DBMS_FLASHBACK TO test;
GRANT FLASHBACK ANY TABLE TO PUBLIC;
GRANT EXECUTE ON emp_policy_func TO PUBLIC;
```

6. Enable the `SCOTT.EMP` table for flashback archive, and for transactions

```
ALTER TABLE SCOTT.EMP FLASHBACK ARCHIVE;
```

7. Perform an update to the `SCOTT.EMP` table.

```
UPDATE SCOTT.EMP SET SAL=SAL+1;
COMMIT;
```

8. Put the preceding procedure to sleep for 60 seconds.

```
EXEC DBMS_LOCK.SLEEP(60);
```

9. Connect as user `test`.

```
connect test@pdb_name
Enter password: password
Connected.
```

10. Perform the following query to show only rows that have `deptno=30`, per the VPD policy:

```
SELECT EMPNO,DEPTNO,SAL FROM SCOTT.EMP;
```

The VPD policy is not working because all rows are shown.

```
SELECT EMPNO,DEPTNO,SAL FROM SCOTT.EMP AS OF TIMESTAMP SYSDATE-1;
```

11. Connect as user `sysadmin_vpd`.

```
connect sysadmin_vpd@pdb_name
Enter password: password
Connected.
```

12. Find the object ID for the `EMP` table.

```
SELECT OBJECT_ID FROM DBA_OBJECTS WHERE OBJECT_NAME='EMP';
```

13. Define a similar VPD policy on the `SYS_FBA_HIST_object_id_of_EMP_table` table. This table is internally created by Flashback Archive

```
BEGIN
 DBMS_RLS.ADD_POLICY (
  object_schema     => 'scott',
  object_name       => 'sys_fba_hist_object_id_of_EMP_table',
  policy_name       => 'emp_hist_policy',
  function_schema   => 'sysadmin_vpd,
  policy_function   => 'emp_policy_func',
  policy_type       =>  dbms_rls.dynamic);
END;
/
```

14. Connect as the `test` user.

```
connect test@pdb_name
Enter password: password
Connected.
```

15. Test the policy again:

```
SELECT EMPNO,DEPTNO,SAL FROM SCOTT.EMP AS OF TIMESTAMP SYSDATE-1;
```

Now the VPD policy works, because the query only shows rows with `deptno=30`.

16. Connect as a user who can drop user accounts.
For example:

```
connect sec_admin@pdb_name
Enter password: password
Connected.
```

17. Drop the `sysadmin_vpd` user and its objects as follows:

```
DROP USER sysadmin_vpd CASCADE;
```

14.5.10 User Models and Oracle Virtual Private Database

You can use Oracle Virtual Private Database in several types of user models.

These user models are as follows:

- **Application users who are also database users.** Oracle Database enables applications to enforce fine-grained access control for each user, regardless of whether that user is a database user or an application user unknown to the database. When application users are also database users, Oracle Virtual Private Database enforcement works as follows: users connect to the database, and then the application sets up application contexts for each session. (You can use the default `USERENV` application context namespace, which provides many parameters for retrieve different types of user session data.) As each session is initiated under a different user name, it can enforce different fine-grained access control conditions for each user.
- **Proxy authentication using OCI or JDBC/OCI.** Proxy authentication permits different fine-grained access control for each user, because each session (OCI or JDBC/OCI) is a distinct database session with its own application context.
- **Proxy authentication integrated with Enterprise User Security.** If you have integrated proxy authentication by using Enterprise User Security, you can retrieve user roles and other attributes from Oracle Internet Directory to enforce Oracle Virtual Private Database policies. (In addition, globally initialized application context can also be retrieved from the directory.)

  > **Note:**
  >
  > Enterprise User Security (EUS) is deprecated with Oracle Database 23ai. Oracle recommends that you migrate to using Centrally Managed Users (CMU). This feature enables you to directly connect with Microsoft Active Directory without an intervening directory service for enterprise user authentication and authorization to the database. If your Oracle Database is in the cloud, you can also choose to move to one of the newer integrations with a cloud identity provider.

- **Users connecting as One Big Application User.** Applications connecting to the database as a single user on behalf of all users can have fine-grained access control for each user. The user for that single session is often called *One Big Application User*. Within the context of that session, however, an application developer can create a global application context attribute to represent the individual application user (for example, `REALUSER`). Although all database sessions and audit records are created for One Big Application User, the attributes for each session can vary, depending on who the end user is. This model works best for applications with a limited number of users and no reuse of sessions. The scope of roles and database auditing is diminished because each session is created as the same database user.
- **Web-based applications.** Web-based applications typically have hundreds of users. Even when there are persistent connections to the database, supporting data retrieval for many user requests, these connections are not specific to particular Web-based users. Instead, Web-based applications typically set up and reuse connections, to provide scalability, rather than having different sessions for each user. For example, when Web users Jane and Ajit connect to a middle tier application, it may establish a single database session that it uses on behalf of both users. Typically, neither Jane nor Ajit is known to the database.

The application is responsible for switching the user name on the connection, so that, at any given time, it is either Jane or Ajit using the session.

Oracle Virtual Private Database helps with connection pooling by allowing multiple connections to access more than one global application context. This ability makes it unnecessary to establish a separate application context for each distinct user session.

Table 14-3 summarizes how Oracle Virtual Private Database applies to user models.

Table 14-3 Oracle Virtual Private Database in Different User Models

User Model Scenario	Individual Database Connection	Separate Application Context per User	Single Database Connection	Application Must Switch User Name
Application users are also database users	Yes	Yes	No	No
Proxy authentication using OCI or JDBC/OCI	Yes	Yes	No	No
Proxy authentication integrated with Enterprise User Security[1]	No	No	Yes	Yes
One Big Application User	No	No[2]	No	Yes[2]
Web-based applications	No	No	Yes	Yes

1 User roles and other attributes, including globally initialized application context, can be retrieved from Oracle Internet Directory to enforce Oracle Virtual Private Database.
2 Application developers can create a global application context attribute representing individual application users (for example, `REALUSER`), which can then be used for controlling each session attributes, or for auditing.

Related Topics

- Global Application Contexts
  You can use a global application context to access application values across database sessions, including an Oracle Real Application Clusters environment.

14.5.11 Oracle Virtual Private Database and JSON

You should be aware of how to use Oracle VPD with JSON.

You cannot create VPD policies on JASN relational duality views. Any attempt to do so results in an `ORA-42623: Virtual Private Database (VPD) cannot be applied on JSON Relational Duality Views` error. However, you can create VPD policies on base tables of JSON relational duality views.

14.6 Oracle Virtual Private Database Data Dictionary Views

Oracle Database provides data dictionary views that list information about Oracle Virtual Private Database policies.

Table 14-4 lists Virtual Private Database-specific views

Table 14-4 Data Dictionary Views That Display Information about VPD Policies

View	Description
ALL_POLICIES	Describes all Oracle Virtual Private Database security policies for objects accessible to the current user.
ALL_POLICY_ATTRIBUTES	Describes all the application context namespaces, attributes, and Virtual Private Database policy associations where the logged in user is the owner of the VPD policy or the VPD policy belongs to PUBLIC.
ALL_POLICY_CONTEXTS	Describes the driving contexts defined for the synonyms, tables, and views accessible to the current user. A driving context is an application context used in an Oracle Virtual Private Database policy.
ALL_POLICY_GROUPS	Describes the Oracle Virtual Private Database policy groups defined for the synonyms, tables, and views accessible to the current user
ALL_SEC_RELEVANT_COLS	Describes the security relevant columns of the security policies for the tables and views accessible to the current user
DBA_POLICIES	Describes all Oracle Virtual Private Database security policies in the database.
DBA_POLICY_ATTRIBUTES	Describes all the application context namespaces, attributes, and Virtual Private Database policy associations for context-sensitive and shared context-sensitive Virtual Private Database policies
DBA_POLICY_GROUPS	Describes all policy groups in the database.
DBA_POLICY_CONTEXTS	Describes all driving contexts in the database. Its columns are the same as those in ALL_POLICY_CONTEXTS.
DBA_SEC_RELEVANT_COLS	Describes the security relevant columns of all security policies in the database
UNIFIED_AUDIT_TRAIL	Captures the VPD predicates in the RLS_INFO column, for unified auditing and fine-grained auditing
USER_POLICIES	Describes all Oracle Virtual Private Database security policies associated with objects owned by the current user. This view does not display the OBJECT_OWNER column.
USER_POLICY_ATTRIBUTES	Describes all the application context namespaces, attributes, and Virtual Private Database policy associations where the owner of the Virtual Private Database policy is the current user
USER_POLICY_CONTEXTS	Describes the driving contexts defined for the synonyms, tables, and views owned by the current user. Its columns (except for OBJECT_OWNER) are the same as those in ALL_POLICY_CONTEXTS.
USER_SEC_RELEVANT_COLS	Describes the security relevant columns of the security policies for the tables and views owned by the current user. Its columns (except for OBJECT_OWNER) are the same as those in ALL_SEC_RELEVANT_COLS.
USER_POLICY_GROUPS	Describes the policy groups defined for the synonyms, tables, and views owned by the current user. This view does not display the OBJECT_OWNER column.
V$VPD_POLICY	For the current PDB, displays all the fine-grained security policies and predicates associated with the cursors currently in the library cache. This view is useful for finding the policies that were applied to a SQL statement.

> **Tip:**
>
> In addition to these views, check the database trace file if you find errors in application that use Virtual Private Database policies. The `USER_DUMP_DEST` initialization parameter specifies the current location of the trace files. You can find the value of this parameter by issuing `SHOW PARAMETER USER_DUMP_DEST` in SQL*Plus.

Related Topics

- *Oracle Database Reference*
- *Oracle Database SQL Tuning Guide*