168
DBMS_RLS

The DBMS_RLS package contains the fine-grained access control administrative interface, which is used to implement Virtual Private Database (VPD).

DBMS_RLS is available with the Enterprise Edition only.

> **See Also:**
>
> *Oracle Database Security Guide*for usage information about DBMS_RLS

This chapter contains the following topics:

DBMS_RLS Overview

The functionality to support fine-grained access control is based on dynamic predicates, where security rules are not embedded in views, but are acquired at the statement parse time, when the base table or view is referenced in a DML statement.

A dynamic predicate for a table, view, or synonym is generated by a PL/SQL function, which is associated with a security policy through a PL/SQL interface. For example:

```
DBMS_RLS.ADD_POLICY (
   'hr', 'employees', 'emp_policy', 'hr', 'emp_sec', 'select', 'user_ctx', 'time');
```

Whenever the EMPLOYEES table, under the HR schema, is referenced in a query or subquery (SELECT), the server calls the EMP_SEC function (under the HR schema). This function returns a predicate specific to the current schema for the EMP_POLICY policy. The policy function may generate the predicates based on the session environment variables available during the function call. These variables usually appear in the form of application contexts. The policy can specify any combination of security-relevant columns and of these statement types: INDEX, SELECT, INSERT, UPDATE, or DELETE.

The server then produces a transient view with the text:

```
SELECT * FROM hr.employees WHERE P1
```

Here, P1 (for example, where SAL > 10000, or even a subquery) is the predicate returned from the EMP_SEC function. The server treats the EMPLOYEES table as a view and does the view

expansion just like the ordinary view, except that the view text is taken from the transient view instead of the data dictionary.

If the predicate contains subqueries, then the owner (definer) of the policy function is used to resolve objects within the subqueries and checks security for those objects. In other words, users who have access privilege to the policy-protected objects do not need to know anything about the policy. They do not need to be granted object privileges for any underlying security policy. Furthermore, the users do not require `EXECUTE` privilege on the policy function, because the server makes the call with the function definer's right.

> **Note:**
>
> The transient view can preserve the updatability of the parent object because it is derived from a single table or view with predicate only; that is, no `JOIN`, `ORDER BY`, `GROUP BY`, and so on.

`DBMS_RLS` also provides the interface to drop or enable security policies. For example, you can drop or enable the `EMP_POLICY` with the following PL/SQL statements:

```
DBMS_RLS.DROP_POLICY('hr', 'employees', 'emp_policy');
DBMS_RLS.ENABLE_POLICY('hr', 'employees', 'emp_policy', TRUE);
```

DBMS_RLS Security Model

A security check is performed when the transient view is created with a subquery. The schema owning the policy function, which generates the dynamic predicate, is the transient view's definer for security check and object lookup.

DBMS_RLS Constants

The `DBMS_RLS` package includes constants that can be used for specifying parameter values.

Table 168-1 DBMS_RLS Constants

Constant	Type	Value	Description
`ADD_ATTRIBUTE_ASSOCIATION`	`BINARY_INTEGER`	1	Used with `DBMS_RLS.ALTER_POLICY` and `DBMS_RLS_ALTER_GROUPED_POLICY`: adds the specified namespace and attribute to the policy or grouped policy
`REMOVE_ATTRIBUTE_ASSOCIATION`	`BINARY_INTEGER`	2	Used with `DBMS_RLS.ALTER_POLICY` and `DBMS_RLS.ALTER_GROUPED_POLICY`: removes the specified namespace and attribute to the policy or grouped policy.

DBMS_RLS Operational Notes

The DBMS_RLS procedures cause current DML transactions, if any, to commit before the operation. However, the procedures do not cause a commit first if they are inside a DDL event trigger. With DDL transactions, the DBMS_RLS procedures are part of the DDL transaction.

For example, you may create a trigger for CREATE TABLE. Inside the trigger, you may add a column through ALTER TABLE, and you can add a policy through DBMS_RLS. All these operations are in the same transaction as CREATE TABLE, even though each one is a DDL statement. The CREATE TABLE succeeds only if the trigger is completed successfully.

Views of current cursors and corresponding predicates are available from V$VPD_POLICIES.

A synonym can reference only a view or a table.

DBMS_RLS Rules and Limits

Using long identifiers is supported for VPD. The maximum length for arguments such as object_schema, object_name, and policy_name, which apply to objects (table names, policy names, and subprogram names) and views is 128 bytes.

Summary of DBMS_RLS Subprograms

This table lists and briefly describes the subprograms available in DBMS_RLS.

Table 168-2 DBMS_RLS Package Subprograms

Subprogram	Description
ADD_GROUPED_POLICY Procedure	Adds a policy associated with a policy group
ADD_POLICY Procedure	Adds a fine-grained access control policy to a table, view, or synonym
ADD_POLICY_CONTEXT Procedure	Adds the context for the active application
ALTER_POLICY Procedure	Associates an application context attribute with VPD policies
ALTER_GROUPED_POLICY Procedure	Adds application context related changes
CREATE_POLICY_GROUP Procedure	Creates a policy group
DELETE_POLICY_GROUP Procedure	Deletes a policy group
DISABLE_GROUPED_POLICY Procedure	Disables a row-level group security policy
DROP_GROUPED_POLICY Procedure	Drops a policy associated with a policy group
DROP_POLICY Procedure	Drops a fine-grained access control policy from a table, view, or synonym
DROP_POLICY_CONTEXT Procedure	Drops a driving context from the object so that it will have one less driving context
ENABLE_GROUPED_POLICY Procedure	Enables or disables a row-level group security policy
ENABLE_POLICY Procedure	Enables or disables a fine-grained access control policy
REFRESH_GROUPED_POLICY Procedure	Reparses the SQL statements associated with a refreshed policy

Table 168-2 (Cont.) DBMS_RLS Package Subprograms

Subprogram	Description
REFRESH_POLICY Procedure	Causes all the cached statements associated with the policy to be reparsed

ADD_GROUPED_POLICY Procedure

This procedure adds a policy associated with a policy group.

Syntax

```
DBMS_RLS.ADD_GROUPED_POLICY(
   object_schema            IN  VARCHAR2        DEFAULT NULL,
   object_name              IN  VARCHAR2,
   policy_group             IN  VARCHAR2        DEFAULT 'SYS_DEFAULT',
   policy_name              IN  VARCHAR2,
   function_schema          IN  VARCHAR2        DEFAULT NULL,
   policy_function          IN  VARCHAR2,
   statement_types          IN  VARCHAR2        DEFAULT NULL,
   update_check             IN  BOOLEAN         DEFAULT FALSE,
   enable                   IN  BOOLEAN         DEFAULT TRUE,
   static_policy            IN  BOOLEAN         DEFAULT FALSE,
   policy_type              IN  BINARY_INTEGER  DEFAULT NULL,
   long_predicate           IN  BOOLEAN         DEFAULT FALSE,
   sec_relevant_cols        IN  VARCHAR2,
   sec_relevant_cols_opt    IN  BINARY_INTEGER  DEFAULT NULL,
   namespace                IN  VARCHAR2        DEFAULT NULL,
   attribute                IN  VARCHAR2        DEFAULT NULL);
```

Parameters

Table 168-3 ADD_GROUPED_POLICY Procedure Parameters

Parameter	Description
`object_schema`	Schema containing the table, view, or synonym. If no `object_schema` is specified or is `NULL`, then the current schema is used.
`object_name`	Name of the table, view, or synonym to which the policy is added
`policy_group`	Name of the policy group to which the policy belongs
`policy_name`	Name of the policy; must be unique for the same table or view
`function_schema`	Schema owning the policy function. If no `function_schema` is specified or is `NULL`, then the current schema is used.
`policy_function`	Name of the function that generates a predicate for the policy. If the function is defined within a package, the name of the package must be present.
`statement_types`	Statement types to which the policy applies. It can be any combination of `INDEX`, `SELECT`, `UPDATE`, or `DELETE`. The default is to apply to all of these types except `INSERT` and `INDEX`.

Table 168-3 (Cont.) ADD_GROUPED_POLICY Procedure Parameters

Parameter	Description
update_check	For INSERT and UPDATE statements only, setting update_check to TRUE causes the server to check the policy against the value after INSERT or UPDATE. The check applies only to the security relevant columns that are included in the policy definition. In other words, the INSERT or UPDATE operation will fail only if the security relevant column that is defined in the policy is added or updated in the INSERT or UPDATE statement.
enable	Indicates if the policy is enable when it is added. The default is TRUE.
static_policy	Default is FALSE. If it is set to TRUE, the server assumes that the policy function for the static policy produces the same predicate string for anyone accessing the object, except for SYS or the privilege user who has the EXEMPT ACCESS POLICY privilege.
policy_type	Default is NULL, which means policy_type is decided by the value of static_policy. The available policy types are listed in Table 168-5. Specifying any of these policy types overrides the value of static_policy.
long_predicate	Default is FALSE, which means the policy function can return a predicate with a length of up to 4000 bytes. TRUE means the predicate text string length can be up to 32K bytes.Policies existing prior to the availability of this parameter retain a 32K limit.
sec_relevant_cols	Enables column-level Virtual Private Database (VPD), which enforces security policies when a column containing sensitive information is referenced in a query. Applies to tables and views, but not to synonyms. Specify a list of comma- or space-separated valid column names of the policy-protected object. The policy is enforced only if a specified column is referenced (or, for an abstract datatype column, its attributes are referenced) in the user SQL statement or its underlying view definition. Default is all the user-defined columns for the object.
namespace	Name which determines the application context namespace
attribute	Attribute which determines the application context attribute name

Usage Notes

- This procedure adds a policy to the specified table, view, or synonym and associates the policy with the specified policy group.
- The policy group must have been created by using the CREATE_POLICY_GROUP Procedure.
- The policy name must be unique within a policy group for a specific object.
- Policies from the default policy group, SYS_DEFAULT, are always executed regardless of the active policy group; however, fine-grained access control policies do not apply to users with EXEMPT ACCESS POLICY system privilege.

ADD_POLICY Procedure

This procedure adds a fine-grained access control policy to a table, view, or synonym.

The procedure causes the current transaction, if any, to commit before the operation is carried out. However, this does not cause a commit first if it is inside a DDL event trigger.

> **See Also:**
>
> Operational Notes

A COMMIT is also performed at the end of the operation.

Syntax

```
DBMS_RLS.ADD_POLICY (
   object_schema            IN  VARCHAR2       DEFAULT NULL,
   object_name              IN  VARCHAR2,
   policy_name              IN  VARCHAR2,
   function_schema          IN  VARCHAR2       DEFAULT NULL,
   policy_function          IN  VARCHAR2,
   statement_types          IN  VARCHAR2       DEFAULT NULL,
   update_check             IN  BOOLEAN        DEFAULT FALSE,
   enable                   IN  BOOLEAN        DEFAULT TRUE,
   static_policy            IN  BOOLEAN        DEFAULT FALSE,
   policy_type              IN  BINARY_INTEGER DEFAULT NULL,
   long_predicate           IN  BOOLEAN        DEFAULT FALSE,
   sec_relevant_cols        IN  VARCHAR2       DEFAULT NULL,
   sec_relevant_cols_opt    IN  BINARY_INTEGER DEFAULT NULL,
   namespace                IN  VARCHAR2       DEFAULT NULL,
   attribute                IN  VARCHAR2       DEFAULT NULL);
```

Parameters

Table 168-4 ADD_POLICY Procedure Parameters

Parameter	Description
object_schema	Schema containing the table, view, or synonym. If no object_schema is specified or is NULL, then the current schema is used.
object_name	Name of table, view, or synonym to which the policy is added.
policy_name	Name of policy to be added. It must be unique for the same table or view. Do not enter special characters such as spaces or commas. If you want to use special characters for the policy name, then enclose the name in quotation marks.
function_schema	Schema owning the policy function. If no function_schema is specified or is NULL, then the current schema is used.
policy_function	Name of a function which generates a predicate for the policy. If the function is defined within a package, then the name of the package must be present.
statement_types	Statement types to which the policy applies. It can be any combination of INDEX, SELECT, UPDATE, or DELETE. The default is to apply to all of these types except INSERT and INDEX .
update_check	Optional argument for the INSERT or UPDATE statement type. The default is FALSE. If you plan to use the INSERT statement type, then you must set update_check to TRUE. Otherwise, an ORA-28104 input value for string is not valid error is generated. The check applies only to the security relevant columns that are included in the policy definition. In other words, the INSERT or UPDATE operation will fail only if the security relevant column that is defined in the policy is added or updated in the INSERT or UPDATE statement.

Table 168-4 (Cont.) ADD_POLICY Procedure Parameters

Parameter	Description
enable	Indicates if the policy is enabled when it is added. The default is TRUE.
static_policy	The default is FALSE. If it is set to TRUE, the server assumes that the policy function for the static policy produces the same predicate string for anyone accessing the object, except for SYS or the privileged user who has the EXEMPT ACCESS POLICY privilege.
policy_type	Default is NULL, which means policy_type is decided by the value of static_policy. The available policy types are listed in Table 168-5. Specifying any of these policy types overrides the value of static_policy.
long_predicate	Default is FALSE, which means the policy function can return a predicate with a length of up to 4000 bytes. TRUE means the predicate text string length can be up to 32K bytes.Policies existing prior to the availability of this parameter retain a 32K limit.
sec_relevant_cols	Enables column-level Virtual Private Database (VPD), which enforces security policies when a column containing sensitive information is referenced in a query. Applies to tables and views, but not to synonyms. Specify a list of comma- or space-separated valid column names of the policy-protected object. The policy is enforced only if a specified column is referenced (or, for an abstract datatype column, its attributes are referenced) in the user SQL statement or its underlying view definition. Default is all the user-defined columns for the object.
sec_relevant_cols_opt	Use with sec_relevant_cols to display all rows for column-level VPD filtered queries (SELECT only), but where sensitive columns appear as NULL. Default is set to NULL, which allows the filtering defined with sec_relevant_cols to take effect. Set to dbms_rls.ALL_ROWS to display all rows, but with sensitive column values, which are filtered by sec_relevant_cols, displayed as NULL. See Usage Notes for restrictions and additional information about this option.
namespace	Name which determines the application context namespace
attribute	Attribute which determines the application context attribute name

Table 168-5 DBMS_RLS.ADD_POLICY Policy Types

Policy Type	Description
STATIC	Predicate is assumed to be the same regardless of the runtime environment. Static policy functions are executed once and then cached in SGA. Statements accessing the same object do not reexecute the policy function. However, each execution of the same cursor could produce a different row set even for the same predicate because the predicate may filter the data differently based on attributes such as SYS_CONTEXT or SYSDATE. Applies to only one object.
SHARED_STATIC	Same as STATIC except that the server first looks for a cached predicate generated by the same policy function of the same policy type. Shared across multiple objects.
CONTEXT_SENSITIVE	Server re-evaluates the policy function at statement execution time if it detects context changes since the last use of the cursor. For session pooling where multiple clients share a database session, the middle tier must reset context during client switches. Note that the server does not cache the value returned by the function for this policy type; it always executes the policy function on statement parsing. Applies to only one object.

Table 168-5 (Cont.) DBMS_RLS.ADD_POLICY Policy Types

Policy Type	Description
SHARED_CONTEXT_SENSITIVE	Same as CONTEXT_SENSITIVE except that the server first looks for a cached predicate generated by the same policy function of the same policy type within the same database session. If the predicate is found in the session memory, the policy function is not reexecuted and the cached value is valid until session private application context changes occur. Shared across multiple objects.
DYNAMIC	The default policy type. Server assumes the predicate may be affected by any system or session environment at any time, and so always reexecutes the policy function upon each statement parsing and execution. Applies to only one object.

Usage Notes

- SYS is free of any security policy.
- The policy functions are called by the server. Following is the interface for the function:

```
FUNCTION policy_function (object_schema IN VARCHAR2, object_name VARCHAR2)
    RETURN VARCHAR2
--- object_schema is the schema owning the table or view.
--- object_name is the name of table, view, or synonym to which the policy applies.
```

- The policy functions must have the purity level of WNDS (write no database state).

> **See Also:**
>
> The *Oracle Database Development Guide* has more details about the RESTRICT_REFERENCES pragma.

- Predicates generated from different VPD policies for the same object have the combined effect of a conjunction (ANDed) of all the predicates.
- The security check and object lookup are performed against the owner of the policy function for objects in the subqueries of the dynamic predicates.
- If the function returns a zero length predicate, then it is interpreted as no restriction being applied to the current user for the policy.
- When a table alias is required (for example, parent object is a type table) in the predicate, the name of the table or view itself must be used as the name of the alias. The server constructs the transient view as something like

```
"select c1, c2, ... from tab tab where <predicate>"
```

- Validity of the function is checked at runtime for ease of installation and other dependency issues during import and export.
- Column-level VPD column masking behavior (specified with sec_relevant_cols_opt => dbms_rls.ALL_ROWS) is fundamentally different from all other VPD policies, which return only a subset of rows. Instead the column masking behavior returns all rows specified by the user's query, but the sensitive column values display as NULL. The restrictions for this option are as follows:
  - Only applies to SELECT statements

  - Unlike regular VPD predicates, the masking condition that is generated by the policy function must be a simple boolean expression.
  - If your application performs calculations, or does not expect `NULL` values, then you should use the default behavior of column-level VPD, which is specified with the `sec_relevant_cols` parameter.
  - If you use `UPDATE AS SELECT` with this option, then only the values in the columns you are allowed to see will be updated.
  - This option may prevent some rows from displaying. For example:

    ```
    SELECT * FROM employees
    WHERE salary = 10
    ```

    This query may not return rows if the `salary` column returns a `NULL` value because the column masking option has been set.

- When you add a VPD policy to a synonym, it causes all the dependent objects of the synonym, including policy functions that reference the synonym, to be marked `INVALID`.
- You cannot associate a global application context with a context sensitive policy or a context shared sensitive policy.
- The maximum number of policies that can be created for a single object is 255.

Examples

As the first of two examples, the following creates a policy that applies to the `hr.employee` table. This is a column-level VPD policy that will be enforced only if a `SELECT` or an `INDEX` statement refers to the `salary`, `birthdate`, or `SSN` columns of the table explicitly, or implicitly through a view. It is also a `CONTEXT_SENSITIVE` policy, so the server will invoke the policy function `hr.hrfun` at parse time. The namespace and attribute application context parameters restrict the policy evaluation only when the application context values change. During execution, it will only invoke the function if there has been any session private context change since the last use of the statement cursor. The predicate generated by the policy function must not exceed 4000 bytes, the default length limit, since the `long_predicate` parameter is omitted from the call.

```
BEGIN
   DBMS_RLS.ADD_POLICY(
      object_schema      => 'hr',
      object_name        => 'employee',
      policy_name        => 'hr_policy',
      function_schema    => 'hr',
      policy_function    => 'hrfun',
      statement_types    => 'select,index',
      policy_type        => DBMS_RLS.CONTEXT_SENSITIVE,
      sec_relevant_cols  => 'salary,birthdate,ssn',
      namespace          => 'empno_ctx',
      attribute          => 'emp_id');
END;
```

As the second example, the following command creates another policy that applies to the same object for hosting, so users can access only data based on their subscriber ID. Since it is defined as a `SHARED_STATIC` policy type, the server will first try to find the predicate in the SGA cache. The server will only invoke the policy function, `subfun`, if that search fails.

```
BEGIN
   DBMS_RLS.ADD_POLICY(
      object_schema      => 'hr',
      object_name        => 'employee',
```

```
      policy_name        => 'hosting_policy',
      function_schema    => 'hr',
      policy_function    => 'subfun',
      policy_type        => dbms_rls.SHARED_STATIC);
END;
```

ADD_POLICY_CONTEXT Procedure

This procedure adds the context for the active application.

Syntax

```
DBMS_RLS.ADD_POLICY_CONTEXT (
   object_schema   IN VARCHAR2 NULL,
   object_name     IN VARCHAR2,
   namespace       IN VARCHAR2,
   attribute       IN VARCHAR2);
```

Parameters

Table 168-6 ADD_POLICY_CONTEXT Procedure Parameters

Parameter	Description
`object_schema`	Schema containing the table, view, or synonym. If no `object_schema` is specified or is `NULL`, then the current schema is used.
`object_name`	Name of the table, view, or synonym to which the policy is added.
`namespace`	Name which determines the application context namespace
`attribute`	Attribute which determines the application context attribute name

Usage Notes

Note the following:

- This procedure indicates the application context that drives the enforcement of policies; this is the context that determines which application is running.
- The driving context can be session or global.
- At execution time, the server retrieves the name of the active policy group from the value of this context.
- There must be at least one driving context defined for each object that has fine-grained access control policies; otherwise, all policies for the object will be executed.
- Adding multiple context to the same object will cause policies from multiple policy groups to be enforced.
- If the driving context is `NULL`, policies from all policy groups are used.
- If the driving context is a policy group with policies, all enabled policies from that policy group will be applied, along with all policies from the `SYS_DEFAULT` policy group.
- To add a policy to table `HR.EMPLOYEES` in group `access_control_group`, the following command is issued:

  ```
  DBMS_RLS.ADD_GROUPED_POLICY('hr','employees','access_control_group','policy1','SYS',
  'HR.ACCESS');
  ```

ALTER_POLICY Procedure

This procedure associates an application context attribute with VPD policies.

Syntax

```
DBMS_RLS.ALTER_POLICY (
   object_schema    IN VARCHAR2 DEFAULT NULL,
   object_name      IN VARCHAR2,
   policy_name      IN VARCHAR2,
   alter_option     IN NUMBER,
   namespace        IN VARCHAR2 DEFAULT NULL,
   attribute        IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 168-7 ALTER_POLICY Procedure Parameters

Parameter	Description
`object_schema`	Schema containing the table, view, or synonym. If no `object_schema` is specified or is `NULL`, then the current schema is used.
`object_name`	Name of the table, view, or synonym to which the policy is added
`policy_name`	Name of the policy, unique for the same table or view
`alter_option`	Used to determine whether the application context is being added or removed from an Oracle Virtual Private Database policy
`namespace`	Name which determines the application context namespace
`attribute`	Attribute which determines the application context attribute name

Usage Notes

Note the following:

- This procedure associates an application context namespace and application context attribute to context sensitive and shared context sensitive policies only. Specifying application context namespace and application context attribute for `DYNAMIC`, `STATIC` or `SHARED_STATIC` policies will result in an error. If `namespace` is specified, `attribute` should also be specified for the procedure call.
- You cannot associate a global application context with a context sensitive policy or a context shared sensitive policy.
- Invocations of `ALTER_POLICY` which modify a shared context sensitive VPD policy have an effect on all shared context sensitive VPD policies that have the same VPD policy function.
- The driving context can be session or global.
- At execution time, the server retrieves the name of the active policy group from the value of this context.
- There must be at least one driving context defined for each object that has fine-grained access control policies; otherwise, all policies for the object will be executed.
- Adding multiple context to the same object will cause policies from multiple policy groups to be enforced.
- If the driving context is `NULL`, policies from all policy groups are used.

- If the driving context is a policy group with policies, all enabled policies from that policy group will be applied, along with all policies from the `SYS_DEFAULT` policy group.
- To add a policy to table `hr.employees` in group `access_control_group`, the following command is issued:

```
DBMS_RLS.ADD_GROUPED_POLICY(
   'hr','employees','access_control_group','policy1','SYS', 'HR.ACCESS');
```

ALTER_GROUPED_POLICY Procedure

This procedure adds application context related changes.

Syntax

```
DBMS_RLS.ALTER_GROUPED_POLICY (
   object_schema    IN VARCHAR2 DEFAULT NULL,
   object_name      IN VARCHAR2,
   policy_group     IN VARCHAR2 DEFAULT SYS_DEFAULT,
   policy_name      IN VARCHAR2,
   alter_option     IN NUMBER,
   namespace        IN VARCHAR2 DEFAULT NULL,
   attribute        IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 168-8 ALTER_GROUPED_POLICY Procedure Parameters

Parameter	Description
`object_schema`	Schema containing the table, view, or synonym. If no `object_schema` is specified or is `NULL`, then the current schema is used.
`object_name`	Name of the table, view, or synonym to which the policy is added
`policy_group`	Name of the policy group to which this policy belongs; must be unique for the same table or view
`policy_name`	Name of the policy, unique for the same table or view
`alter_option`	Used to determine whether the application context is being added or removed from the Oracle Virtual Private Database policy
`namespace`	Name that determines the application context namespace
`attribute`	Attribute determines the application context attribute name

Usage Notes

Note the following:

- This procedure will associate an application context namespace and application context attribute to context sensitive and shared context sensitive policies only. Specifying application context namespace and application context attribute for `DYNAMIC`, `STATIC` or `SHARED_STATIC` policies will result in an error. If `namespace` is specified, `attribute` should also be specified for the procedure call.
- You cannot associate a global application context with a context sensitive policy or a context shared sensitive policy.
- Invocations of `ALTER_GROUPED_POLICY` which modify a shared context sensitive VPD policy have an effect on all shared context sensitive VPD policies that have the same VPD policy function.

- The driving context can be session or global.
- At execution time, the server retrieves the name of the active policy group from the value of this context.
- There must be at least one driving context defined for each object that has fine-grained access control policies; otherwise, all policies for the object will be executed.
- Adding multiple context to the same object will cause policies from multiple policy groups to be enforced.
- If the driving context is `NULL`, policies from all policy groups are used.
- If the driving context is a policy group with policies, all enabled policies from that policy group will be applied, along with all policies from the `SYS_DEFAULT` policy group.
- To add a policy to table `hr.employees` in group `access_control_group`, the following command is issued:

```
DBMS_RLS.ADD_GROUPED_POLICY (
   'hr','employees','access_control_group','policy1','SYS', 'HR.ACCESS');
```

CREATE_POLICY_GROUP Procedure

This procedure creates a policy group.

Syntax

```
DBMS_RLS.CREATE_POLICY_GROUP (
   object_schema    IN VARCHAR2 NULL,
   object_name      IN VARCHAR2,
   policy_group     IN VARCHAR2);
```

Parameters

Table 168-9 CREATE_POLICY_GROUP Procedure Parameters

Parameter	Description
`object_schema`	Schema containing the table, view, or synonym. If no `object_schema` is specified or is `NULL`, then the current schema is used.
`object_name`	Name of the table, view, or synonym to which the policy is added
`policy_group`	Name of the policy group that the policy belongs to

Usage Notes

The group must be unique for each table or view.

DELETE_POLICY_GROUP Procedure

This procedure deletes a policy group.

Syntax

```
DBMS_RLS.DELETE_POLICY_GROUP (
  object_schema    IN VARCHAR2 NULL,
  object_name      IN VARCHAR2,
  policy_group     IN VARCHAR2);
```

Parameters

Table 168-10 DELETE_POLICY_GROUP Procedure Parameters

Parameter	Description
object_schema	Schema containing the table, view, or synonym. If no object_schema is specified or is NULL, then the current schema is used.
object_name	Name of the table, view, or synonym to which the policy is added
policy_group	Name of the policy group that the policy belongs to

Usage Notes

Note the following:

- This procedure deletes a policy group for the specified table, view, or synonym.
- No policy can be in the policy group.

DISABLE_GROUPED_POLICY Procedure

This procedure disables a row-level group security policy.

Syntax

```
DBMS_RLS.DISABLE_GROUPED_POLICY (
   object_schema   IN VARCHAR2 NULL,
   object_name     IN VARCHAR2,
   group_name      IN VARCHAR2,
   policy_name     IN VARCHAR2);
```

Parameters

Table 168-11 DISABLE_GROUPED_POLICY Procedure Parameters

Parameter	Description
object_schema	Schema containing the table, view, or synonym
object_name	Name of the table, view, or synonym with which the policy is associated
group_name	Name of the group of the policy
policy_name	Name of the policy to be enabled or disabled

Usage Notes

- The procedure causes the current transaction, if any, to commit before the operation is carried out.
- A commit is performed at the end of the operation.
- A policy is disabled when this procedure is executed or when the ENABLE_GROUPED_POLICY procedure is executed with "enable" set to FALSE.

DROP_GROUPED_POLICY Procedure

This procedure drops a policy associated with a policy group.

Syntax

```
DBMS_RLS.DROP_GROUPED_POLICY (
   object_schema   IN VARCHAR2 NULL,
   object_name     IN VARCHAR2,
   policy_group    IN VARCHAR2 'SYS_DEFAULT',
   policy_name     IN VARCHAR2);
```

Parameters

Table 168-12 DROP_GROUPED_POLICY Procedure Parameters

Parameter	Description
`object_schema`	Schema containing the table, view, or synonym. If no `object_schema` is specified or is `NULL`, then the current schema is used.
`object_name`	Name of the table, view, or synonym to which the policy is dropped
`policy_group`	Name of the policy group to which the policy belongs
`policy_name`	Name of the policy

DROP_POLICY Procedure

This procedure drops a fine-grained access control policy from a table, view, or synonym.

The procedure causes the current transaction, if any, to commit before the operation is carried out. However, this does not cause a commit first if it is inside a DDL event trigger.

> **See Also:**
>
> Operational Notes

A COMMIT is also performed at the end of the operation.

Syntax

```
DBMS_RLS.DROP_POLICY (
   object_schema   IN VARCHAR2 NULL,
   object_name     IN VARCHAR2,
   policy_name     IN VARCHAR2);
```

Parameters

Table 168-13 DROP_POLICY Procedure Parameters

Parameter	Description
`object_schema`	Schema containing the table, view or synonym. If no `object_schema` is specified, or `NULL` is provided, then the current user's schema is assumed.

Table 168-13 (Cont.) DROP_POLICY Procedure Parameters

Parameter	Description
object_name	Name of the table, view, or synonym for which the policy is dropped
policy_name	Name of policy to be dropped from table, view, or synonym

Usage Notes

- When you drop a VPD policy from a synonym, it causes all the dependent objects of the synonym, including policy functions that reference the synonym, to be marked INVALID.

DROP_POLICY_CONTEXT Procedure

This procedure drops a driving context from the object so that it will have one less driving context.

Syntax

```
DBMS_RLS.DROP_POLICY_CONTEXT (
   object_schema    IN VARCHAR2 NULL,
   object_name      IN VARCHAR2,
   namespace        IN VARCHAR2,
   attribute        IN VARCHAR2);
```

Parameters

Table 168-14 DROP_POLICY_CONTEXT Procedure Parameters

Parameter	Description
object_schema	Schema containing the table, view, or synonym. If no object_schema is specified or is NULL, then the current schema is used.
object_name	Name of the table, view, or synonym to which the policy is dropped
namespace	Namespace of the driving context
attribute	Attribute of the driving context

ENABLE_GROUPED_POLICY Procedure

This procedure enables or disables a row-level group security policy.

Syntax

```
DBMS_RLS.ENABLE_GROUPED_POLICY (
   object_schema    IN VARCHAR2 NULL,
   object_name      IN VARCHAR2,
   group_name       IN VARCHAR2,
   policy_name      IN VARCHAR2,
   enable           IN BOOLEAN TRUE);
```

Parameters

Table 168-15 ENABLE_GROUPED_POLICY Procedure Parameters

Parameter	Description
`object_schema`	Schema containing the table, view, or synonym. If no `object_schema` is specified or is `NULL`, then the current schema is used.
`object_name`	Name of the table, view, or synonym with which the policy is associated
`group_name`	Name of the group of the policy
`policy_name`	Name of the policy to be enabled or disabled
`enable`	`TRUE` enables the policy; `FALSE` disables the policy

Usage Notes

- The procedure causes the current transaction, if any, to commit before the operation is carried out.
- A commit is performed at the end of the operation.
- A policy is enabled when it is created.

ENABLE_POLICY Procedure

This procedure enables or disables a fine-grained access control policy. A policy is enabled when it is created.

The procedure causes the current transaction, if any, to commit before the operation is carried out. However, this does not cause a commit first if it is inside a DDL event trigger.

> **See Also:**
>
> Operational Notes

A `COMMIT` is also performed at the end of the operation.

Syntax

```
DBMS_RLS.ENABLE_POLICY (
   object_schema IN VARCHAR2 NULL,
   object_name   IN VARCHAR2,
   policy_name   IN VARCHAR2,
   enable        IN BOOLEAN TRUE);
```

Parameters

Table 168-16 ENABLE_POLICY Procedure Parameters

Parameter	Description
`object_schema`	Schema containing the table, view, or synonym. If no `object_schema` is specified or is `NULL`, then the current schema is used.

Table 168-16 (Cont.) ENABLE_POLICY Procedure Parameters

Parameter	Description
object_name	Name of table, view, or synonym with which the policy is associated
policy_name	Name of policy to be enabled or disabled
enable	TRUE to enable the policy, FALSE to disable the policy

REFRESH_GROUPED_POLICY Procedure

This procedure reparses the SQL statements associated with a refreshed policy.

Syntax

```
DBMS_RLS.REFRESH_GROUPED_POLICY (
   object_schema   IN VARCHAR2 NULL,
   object_name     IN VARCHAR2 NULL,
   group_name      IN VARCHAR2 NULL,
   policy_name     IN VARCHAR2 NULL);
```

Parameters

Table 168-17 REFRESH_GROUPED_POLICY Procedure Parameters

Parameter	Description
object_schema	Schema containing the table, view, or synonym. If no object_schema is specified or is NULL, then the current schema is used.
object_name	Name of the table, view, or synonym with which the policy is associated
group_name	Name of the group of the policy
policy_name	Name of the policy

Usage Notes

- This procedure causes all the cached statements associated with the policy to be reparsed. This guarantees that the latest change to the policy has immediate effect after the procedure is executed.
- The procedure causes the current transaction, if any, to commit before the operation is carried out.
- A commit is performed at the end of the operation.
- The procedure returns an error if it tries to refresh a disabled policy.
- The procedure removes the cached results of context and shared sensitive VPD policies.

REFRESH_POLICY Procedure

This procedure causes all the cached statements associated with the policy to be reparsed. This guarantees that the latest change to this policy will have immediate effect after the procedure is executed.

The procedure causes the current transaction, if any, to commit before the operation is carried out. However, this does not cause a commit first if it is inside a DDL event trigger.

> **See Also:**
>
> Operational Notes

A `COMMIT` is also performed at the end of the operation.

Syntax

```
DBMS_RLS.REFRESH_POLICY (
   object_schema IN VARCHAR2 NULL,
   object_name   IN VARCHAR2 NULL,
   policy_name   IN VARCHAR2 NULL);
```

Parameters

Table 168-18 REFRESH_POLICY Procedure Parameters

Parameter	Description
`object_schema`	Schema containing the table, view, or synonym. If no `object_schema` is specified or is `NULL`, then the current schema is used.
`object_name`	Name of table, view, or synonym with which the policy is associated
`policy_name`	Name of policy to be refreshed

Usage Notes

- The procedure returns an error if it tries to refresh a disabled policy.
- The procedure removes the cached results of context and shared sensitive VPD policies.