Oracle® Label Security

Administrator's Guide

23ai

F46693-05

May 2024

Oracle Label Security Administrator's Guide, 23ai

F46693-05

Primary Author: Patricia Huey

Contributors: Chi Ching Chui, Rishabh Gupta, John Kati, Lakshmi Kethana, Gopal Mulagund, Paul Needham, Hozefa Palitanawala, Vikram Pesati, Amoghavarsha Ramappa, Digvijay Sirmukaddam, Srividya Tata, Kamal Tbeileh, Peter Wahl, Alan Williams

Contents

E Oracle Label Security Tables and Views

Preface

Oracle Label Security enables access control to reach specific (labeled) rows of a database. With Oracle Label Security in place, users with varying privilege levels automatically have (or are excluded from) the right to see or alter labeled rows of data.

Oracle Label Security Administrator’s Guide describes how to use Oracle Label Security to protect sensitive data. It explains the basic concepts behind label-based security and provides examples to show how it is used.

Audience

Oracle Label Security Administrator’s Guide is intended for database administrators (DBAs), application programmers, security administrators, system operators, and other Oracle users who perform the following tasks:

- Analyze application security requirements
- Create label-based security policies
- Administer label-based security policies
- Use label-based security policies

To use this document, you need a working knowledge of SQL and Oracle fundamentals. You should also be familiar with Oracle security features described in Related Documentation. To use SQL*Loader, you must know how to use the file management facilities of your operating system.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Related Documentation

For more information, see these Oracle resources:

- *Oracle Database Concepts*
- *Oracle Database Security Guide*
- *Oracle Database Enterprise User Security Administrator's Guide*
- *Oracle Database Development Guide*
- *Oracle Database Administrator’s Guide*
- *Oracle Database SQL Language Reference*
- *Oracle Database Reference*
- *Oracle Database Utilities*
- *Oracle Database Performance Tuning Guide*

Many of the examples in this book use the sample schemas, which are installed by default when you select the Basic Installation option with an Oracle Database installation. See *Oracle Database Sample Schemas* for information on how these schemas were created and how you can use them yourself.

Oracle Technical Services

To download the product data sheet, frequently asked questions, links to the latest product documentation, product download, and other collateral, visit Oracle Technical Resources (formerly Oracle Technology Network). You must register online before using Oracle Technical Services. Registration is free and can be done at

`https://www.oracle.com/technical-resources/`

My Oracle Support

You can find information about security patches, certifications, and the support knowledge base by visiting My Oracle Support (formerly Oracle*MetaLink*) at

`https://support.oracle.com`

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Changes in This Release for Oracle Label Security Administrator's Guide

This preface contains:

- Changes in Oracle Database 23ai
  Oracle Label Security Administrator’s Guide for Oracle Database 23ai has one new feature and two important desupported features.

Changes in Oracle Database 23ai

Oracle Label Security Administrator’s Guide for Oracle Database 23ai has one new feature and two important desupported features.

- Introduction of the LBAC_TRIGGER Schema for OLS DML Triggers
  Starting with Oracle Database release 23ai, the `LBAC_TRIGGER` schema is available to store DML triggers that Oracle Label Security uses.
- Desupport of Traditional Auditing in Oracle Label Security
  Starting with Oracle Database 23ai, traditional auditing is desupported.

Introduction of the LBAC_TRIGGER Schema for OLS DML Triggers

Starting with Oracle Database release 23ai, the `LBAC_TRIGGER` schema is available to store DML triggers that Oracle Label Security uses.

In previous releases, these triggers were stored in the `LBACSYS` schema. The `LBAC_TRIGGER` schema provides greater security than the `LBACSYS` schema.

New triggers that you create will be automatically stored in the `LBAC_TRIGGER` schema. Triggers that were created in previous releases will still be in the `LBACSYS` schema after you upgrade to release 23ai. You will need to disable and reenable the OLS policies to migrate these triggers to the `LBAC_TRIGGER` schema.

Related Topics

- Using the LBAC_TRIGGER Schema
  Oracle Label Security stores Oracle Label Security triggers in the `LBAC_TRIGGER` schema.

Desupport of Traditional Auditing in Oracle Label Security

Starting with Oracle Database 23ai, traditional auditing is desupported.

Unified auditing is the way forward to perform Oracle Label Security auditing. Unified auditing offers more flexibility to perform selective and effective auditing, which helps you focus on activities that really matter to your enterprise. Unified auditing has one single and secure unified trail, conditional policy for audit selectivity, and default predefined policies for simplicity. To improve security and compliance, Oracle strongly recommends that you use unified auditing.

Related Topics

- Auditing Oracle Label Security Using Unified Auditing
  Oracle recommends that you migrate all your Oracle Label Security audit policies to unified auditing.

Part I
Getting Started with Oracle Label Security

Part I introduces the terms, concepts, and relationships that constitute the basic elements of Oracle Label Security.

1 Introduction to Oracle Label Security

Oracle Label Security provides fine-grained access to individual table rows.

- About Oracle Label Security
  Oracle Label Security controls the display of individual table rows using labels that are assigned to specific individual table rows and application users.
- Benefits of Oracle Label Security
  Oracle Label Security provides several benefits for controlling row level management.
- Who Has Privileges to Use Oracle Label Security?
  When you install Oracle Label Security with a database, the registration process creates an administrative user named `LBACSYS`, who has the `LBAC_DBA` role.
- Duties of Oracle Label Security Administrators
  Oracle Label Security administrators have a set of package- and role-based privileges.
- Components of Oracle Label Security
  An Oracle Label Security policy has a standard set of components.
- Oracle Label Security Architecture
  The Oracle Label Security works with Oracle Database authentication to perform row level security.
- Oracle Label Security Administrative Interfaces
  You can perform Oracle Label Security development and administrative tasks using either of two interfaces.
- Oracle Label Security Demonstration File
  The `olsdemo.sql` file provides a demonstration on using Oracle Label Security.
- Oracle Label Security Integration in a Multitenant Environment
  You can use Oracle Label Security in a multitenant environment.

1.1 About Oracle Label Security

Oracle Label Security controls the display of individual table rows using labels that are assigned to specific individual table rows and application users.

Oracle Label Security works by comparing the row label with a user's label authorizations to enable you to easily restrict sensitive information to only authorized users. This way, users with different authorization levels (for example, managers and sales representatives) can have access to specific rows of data in a table. You can apply Oracle Label Security policies to one or more application tables. The design of Oracle Label Security is similar to Oracle Virtual Private Database (VPD). However, unlike VPD, Oracle Label Security provides the access mediation functions, data dictionary tables, and policy-based architecture out of the box, eliminating customized coding and providing a consistent label based access control model that can be used by multiple applications.

Oracle Label Security is based on multi-level security (MLS) requirements that are found in government and defense organizations.

Oracle Label Security software is installed by default, but not automatically enabled. You can enable Oracle Label Security in either SQL*Plus or by using the Oracle Database Configuration Assistant (DBCA). To administer Oracle Label Security, an Oracle Database administrator should create an Oracle Label Security administrative user and grant this user the `LBAC_DBA` role along with the necessary `EXECUTE` privileges on the Oracle Label Security `SA_*` packages. To manage Oracle Label Security, you can use either a set of PL/SQL packages and standalone functions at the command-line level or Oracle Enterprise Manager Cloud Control. To find information about Oracle Label Security policies, you can query `ALL_SA_*`, `DBA_SA_*`, or `USER_SA_*` data dictionary views.

1.2 Benefits of Oracle Label Security

Oracle Label Security provides several benefits for controlling row level management.

- It enables row level data classification and provides out-of-the-box access mediation based on the data classification and the user label authorization or security clearance.
- It enables you to assign label authorizations or security clearances to both database users and application users.
- It provides both APIs and a graphical user interface (in Oracle Enterprise Manager) for defining and storing data classification labels and user label authorizations.
- It integrates with Oracle Database Vault and Oracle Advanced Security Data Redaction, enabling security clearances to be use in both Database Vault command rules and Data Redaction policy definitions.

1.3 Who Has Privileges to Use Oracle Label Security?

When you install Oracle Label Security with a database, the registration process creates an administrative user named `LBACSYS`, who has the `LBAC_DBA` role.

You can grant the `LBAC_DBA` role to any database user who will be responsible for managing Oracle Label Security policies. (This is the recommended way of administering Oracle Label Security, instead of using the `LBACSYS` user.) In addition, you can grant Oracle Label Security administrators the `EXECUTE` privilege for the Oracle Label Security packages, and privileges to manage individual Oracle Label Security policies.

As with other Oracle administrative user accounts, Oracle strongly recommends that you maintain two accounts for the `LBAC_DBA`. One account, the primary named user account, will be used on a day-to-day basis and the other account will be used as a backup account in case the password of the primary account is lost and must be reset.

1.4 Duties of Oracle Label Security Administrators

Oracle Label Security administrators have a set of package- and role-based privileges.

These privileges are:

- **Package-specific privileges:** Most of the Oracle Label Security PL/SQL packages, except for the public `SA_SESSION` and `SA_UTL` packages, require the

EXECUTE privilege. The other packages are SA_AUDIT_ADMIN, SA_COMPONENTS, SA_LABEL_ADMIN, SA_POLICY_ADMIN, SA_SYSDBA, and SA_USER_ADMIN.

- **Role-based privileges:** The Oracle Label Security-specific roles are:
  - The *policy*_DBA role, which is created and granted to the user when they create a policy. For example, for a policy named ols_hr_pol, the role created is named ols_hr_pol_DBA. This role adds a layer of granularity for access control for your site's Oracle Label Security policies.
  - The LBAC_DBA role, which provides the EXECUTE privilege for the SA_SYSDBA package. This role is owned by the LBACSYS user account. The SA_SYSDBA package enables the user to create, alter, enable, disable, and drop Oracle Label Security policies.

You can use the Oracle Label Security package EXECUTE privilege grants along with grants of the *policy*_DBA role to achieve additional separation of duty. The packages are categorized based on different tasks. For example, you could grant the EXECUTE privilege on the SA_COMPONENTS and SA_LABEL_ADMIN packages to one user or role to manage label definitions, and then grant EXECUTE on SA_USER_ADMIN to a different user or role to manage user labels and privileges. Both of these users or roles must also be granted the *policy*_DBA role for the policies for which they are responsible. In this way, different users can be responsible for the management of different aspects of the policies for which they are responsible. For example, user psmith could be responsible for the label definitions of the ols_hr_pol policy, and user tjones could be responsible for the label definitions of the ols_oe_pol policy. However, user psmith cannot modify label definitions for the ols_oe_pol policy, nor can tjones modify the ols_hr_pol policy label definitions.

Related Topics

- Oracle Label Security Packages
  Oracle Label Security packages provide a direct, command-line interface for ease of administration.

1.5 Components of Oracle Label Security

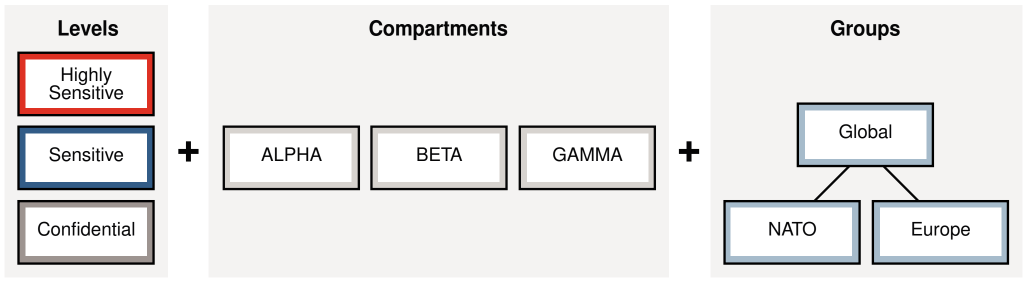
An Oracle Label Security policy has a standard set of components.

These components are as follows:

- **Labels.** Labels for data and users, along with authorizations for users and program units, govern access to specified protected objects. Labels are composed of the following:
  - **Levels.** Levels indicate the type of sensitivity that you want to assign to the row (for example, SENSITIVE or HIGHLY SENSITIVE). Levels are mandatory.
  - **Compartments.** (Optional) Data can have the same level (for example, Public, Confidential and Secret), but can belong to different projects inside a company (for example, ACME Merger and IT Security). Compartments represent the projects in this example that help define more precise access controls. They are most often used in government environments.
  - **Groups.** (Optional) Groups identify organizations owning or accessing the data (for example, UK, US, Asia, Europe). Groups are used both in commercial and government environments, and frequently used in place of compartments due to their flexibility.
- **Policy.** A policy is a name associated with these labels, rules, authorizations, and protected tables.

For example, assume that a user has the `SELECT` privilege on an application table. As illustrated in Figure 1-1, when the user runs a `SELECT` statement, Oracle Label Security evaluates each row selected to determine whether the user can access using the privileges and labels assigned to the user and the label on the row. You can configure Oracle Label Security to perform security checks on `UPDATE`, `DELETE`, and `INSERT` statements as well.

Figure 1-1 Oracle Label Security Label-Based Security

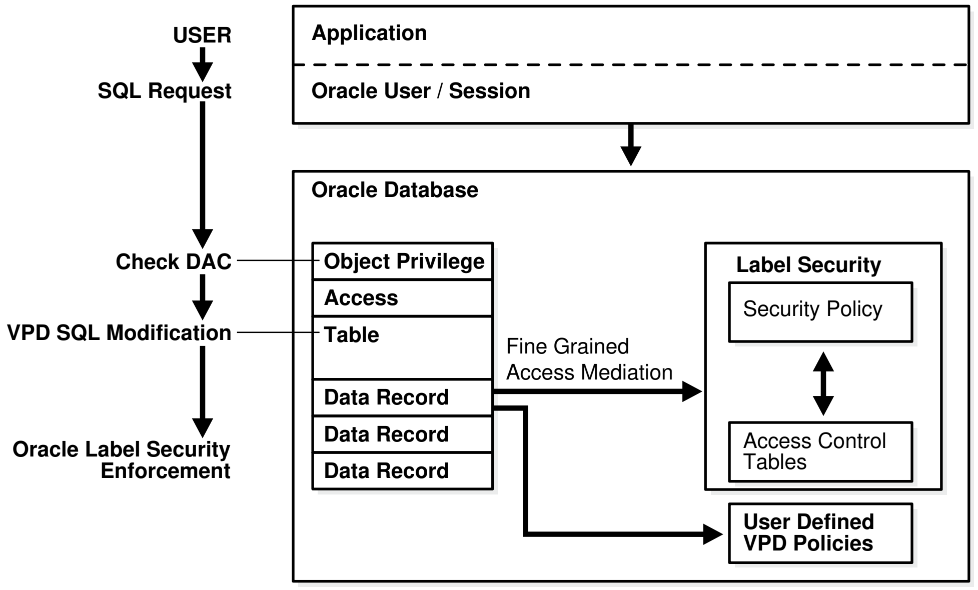


1.6 Oracle Label Security Architecture

The Oracle Label Security works with Oracle Database authentication to perform row level security.

Figure 1-2 shows how data is accessed under Oracle Label Security and the sequence of label security checks.

Figure 1-2 Oracle Label Security Architecture



In this scenario, the following actions take place:

1. An application user in an Oracle Database session sends a SQL request to query a table.
2. Oracle Database checks the user's data access control (DAC) privileges for performing a `SELECT` statement on the table.
3. If the user does have the appropriate privileges, then Oracle Database checks if there are any Oracle Virtual Private Database (VPD) policies attached to the table.
4. Oracle Database then checks if there are any Oracle Label Security policies that are assigned to the table.
5. Oracle Label Security then compares the labels that are assigned to individual rows with the users' label authorizations, allowing or denying access. The session label is based on label authorizations that are assigned to the user.

1.7 Oracle Label Security Administrative Interfaces

You can perform Oracle Label Security development and administrative tasks using either of two interfaces.

- Oracle Label Security Packages
  Oracle Label Security packages provide a direct, command-line interface for ease of administration.
- Oracle Enterprise Manager Cloud Control
  The Oracle Enterprise Manager Cloud Control Web interface can be used to administer Oracle Label Security.

1.7.1 Oracle Label Security Packages

Oracle Label Security packages provide a direct, command-line interface for ease of administration.

Table 1-1 lists the available Oracle Label Security administrative packages.

Table 1-1 Oracle Label Security Administrative Packages

Package	Purpose
`SA_SYSDBA`	To create, alter, and drop Oracle Label Security policies See SA_SYSDBA Policy Management PL/SQL Package
`SA_COMPONENTS`	To define the levels, compartments, and groups for the policy See SA_COMPONENTS Label Components PL/SQL Package
`SA_LABEL_ADMIN`	To perform standard label policy administrative functions, such as creating labels See SA_LABEL_ADMIN Label Management PL/SQL Package
`SA_POLICY_ADMIN`	To apply policies to schemas and tables See SA_POLICY_ADMIN Policy Administration PL/SQL Package
`SA_USER_ADMIN`	To manage user authorizations for levels, compartments, and groups, as well as program unit privileges. Also to administer user privileges. See SA_USER_ADMIN.SET_USER_PRIVS and SA_USER_ADMIN.SET_PROG_PRIVS
`SA_AUDIT_ADMIN`	To set options to audit administrative tasks and use of privileges See SA_AUDIT_ADMIN Oracle Label Security Auditing PL/SQL Package

Table 1-1 (Cont.) Oracle Label Security Administrative Packages

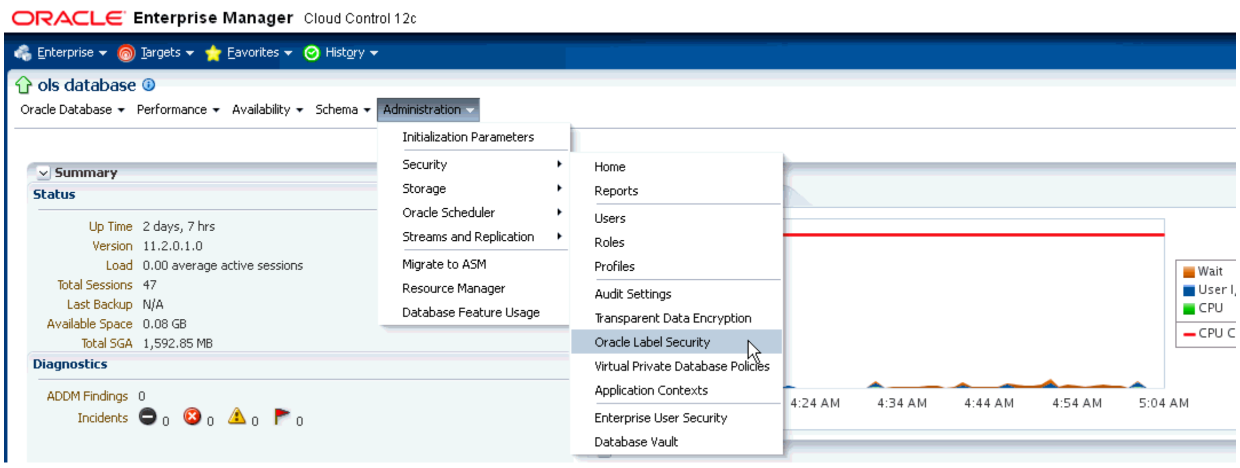
Package	Purpose
`SA_SESSION`	To change labels during a session within the authorizations set by the administrator See SA_SESSION Session Management PL/SQL Package
`SA_UTL`	A set of utility functions designed for use within PL/SQL programs to return information about the current values of the session security attributes, as numeric label values See SA_UTL PL/SQL Utility Functions and Procedures

1.7.2 Oracle Enterprise Manager Cloud Control

The Oracle Enterprise Manager Cloud Control Web interface can be used to administer Oracle Label Security.

Figure 1-3 illustrates the Oracle Enterprise Manager interface.

Figure 1-3 Using Enterprise Manager to Configure Oracle Label Security Policies



Related Topics

- Logging in to Cloud Control or SQL*Plus for Oracle Label Security
  After you complete the Oracle Label Security registration and enablement process, you can begin using it.
- Registering and Logging in to Oracle Label Security
  Before using Oracle Label Security, you must register (configure) it with the database and then you can log in to Oracle Label Security.

1.8 Oracle Label Security Demonstration File

The `olsdemo.sql` file provides a demonstration on using Oracle Label Security.

This file show to create and develop an Oracle Label Security policy using the supplied packages. You can install this script from the `ORACLE_HOME/rdbms/demo` directory.

1.9 Oracle Label Security Integration in a Multitenant Environment

You can use Oracle Label Security in a multitenant environment.

Note:

A multitenant container database is the only supported architecture in Oracle Database 21c and later releases. While the documentation is being revised, legacy terminology may persist. In most cases, "database" and "non-CDB" refer to a CDB or PDB, depending on context. In some contexts, such as upgrades, "non-CDB" refers to a non-CDB from a previous release.

In a multitenant environment, pluggable databases (PDBs) can be plugged in and out of a multitenant container database (CDB) or an application container.

- `rdbms/admin/catols.sql` script on the database to install the label-based framework, data dictionary, data types, and packages. This script creates the `LBACSYS` account.
- Because Oracle Label Security policies are scoped to individual PDBs, you can create individual policies for each PDB. A policy defined for a PDB can be enforced on the local tables and schema objects contained in the PDB.
- In a single CDB, there can be multiple PDBs, each configured with Oracle Label Security.
- You cannot create Oracle Label Security policies in the CDB root or the application root.
- You cannot enforce a local Oracle Label Security policy on a common CDB object or a common application object.
- You cannot assign Oracle Label Security policy labels and privileges to common users and application common users in a pluggable database.
- You cannot assign Oracle Label Security privileges to common procedures or functions and application common procedures or functions in a pluggable database.
- You can uninstall Oracle Label Security by using the `/rdbms/admin/catnools.sql` script.

Related Topics

- *Oracle Database Security Guide*

2 Understanding Data Labels and User Labels

You should understand fundamental concepts of data labels and user labels.

- About Label-Based Security
  Label-based security provides a flexible way of controlling access to sensitive data.
- About User Label and Privilege Management
  To manage user labels and privileges, you must have the `EXECUTE` privilege for the `SA_USER_ADMIN` package and be granted the `*policy*_DBA` role.
- Label Components
  You should understand the elements that are used in labels.
- Label Syntax and Type
  After label components are defined, you can create data labels by combining particular sets of level, compartments, and groups.
- How Data Labels and User Labels Work Together
  A user can access data only within the range of their own label authorizations.
- Administration of Labels
  Oracle Label Security provides administrative interfaces to define and manage the labels used in a database.

2.1 About Label-Based Security

Label-based security provides a flexible way of controlling access to sensitive data.

Oracle Label Security controls data access based on the identity and label of the user, and the sensitivity and label of the data. Label security adds protections beyond the discretionary access controls that determine the operations users can perform upon data in an *object*, such as a table or view.

Table 2-1 shows the three dimensions with which an Oracle Label Security policy controls access to data.

Table 2-1 Oracle Label Security Data Dimensions

Data Dimension	Explanation
Data Labels	A data row label indicates the level and nature of the row's sensitivity and specifies the additional criteria that a user must meet to gain access to that row.
User Labels	A user label specifies that user's sensitivity level plus any compartments and groups that constrain the user's access to labeled data. Each user is assigned a range of levels, compartments, and groups, and each session can operate within that authorized range to access labeled data within that range.
Policy Privileges	Users can be given specific rights (privileges) to perform special operations or to access data beyond their label authorizations.

Note that the discussion here concerns *access* to data. The particular *type* of access, such as reading or writing the data.

When an Oracle Label Security policy is applied to a database table, a column is added to the table to contain each row's label. The administrator can choose to display or hide this column.

Related Topics

- Access Controls and Privileges
  Oracle provides access controls and privileges that determine the *type* of access users can have to labeled rows.

2.2 About User Label and Privilege Management

To manage user labels and privileges, you must have the `EXECUTE` privilege for the `SA_USER_ADMIN` package and be granted the `*policy*_DBA` role.

The `SA_USER_ADMIN` package provides the procedures and functions to manage the Oracle Label Security user security attributes. It contains several procedures to manage user labels by component: that is, specifying user levels, compartments, and groups. For convenience, there are additional procedures that accept character string representations of full labels, rather than components. Note that the level, compartment, and group parameters use the short name defined for each component.

All of the label and privilege information is stored in Oracle Label Security data dictionary tables. When a user connects to the database, their session labels are established based on the information stored in the Oracle Label Security data dictionary.

Note that a user can be authorized under multiple policies.

Related Topics

- SA_USER_ADMIN PL/SQL Package
  The `SA_USER_ADMIN` PL/SQL package manages user labels by label component.
- SA_USER_ADMIN.SET_USER_PRIVS
  The `SA_USER_ADMIN.SET_USER_PRIVS` procedure sets policy-specific privileges for users.
- Duties of Oracle Label Security Administrators
  Oracle Label Security administrators have a set of package- and role-based privileges.

2.3 Label Components

You should understand the elements that are used in labels.

- Label Component Definitions and Valid Characters
  A sensitivity label is a single attribute with multiple components.
- Level Sensitivity Components
  A *level* is a ranking that denotes the sensitivity of the information it labels.
- Compartment Components
  Compartments identify areas that describe the sensitivity of the labeled data, providing a finer level of granularity within a level.

- Group Components
  Groups identify organizations owning or accessing the data, such as `EASTERN_REGION`, `WESTERN_REGION`, `WR_SALES`.
- Industry Examples of Levels, Compartments, and Groups
  Oracle Label Security levels, compartments, groups are designed to be implemented in various industries.

2.3.1 Label Component Definitions and Valid Characters

A sensitivity label is a single attribute with multiple components.

All data labels must contain a level component, but the compartment and group components are optional. An administrator must define the label components before creating labels.

Although the administrator defines both long and short names for the label components, only the short form of the name is displayed upon retrieval. When users manipulate the labels, they use only the short form of the component names. Examples of short forms are illustrated in the **Examples** column of the following table.

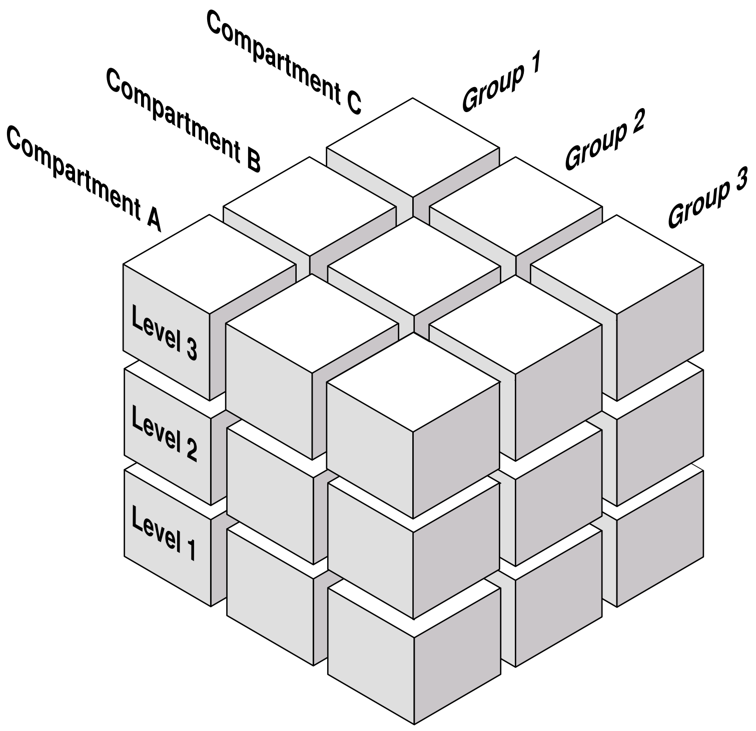
Table 2-2 Sensitivity Label Components

Component	Description	Examples
Level	A single specification of the sensitivity of labeled data within the ordered ranks established	• `C` (for `CONFIDENTIAL`) (1) • `S` (for `SENSITIVE`) (2) • `HS` (for `HIGHLY_SENSITIVE`) (3)
Compartments	Zero or more categories associated with the labeled data	• `FINCL` (for `FINANCIAL`) • `CHEM` (for `CHEMICAL`) • `OP` (for `OPERATIONAL`)
Groups	Zero or more identifiers for organizations owning or accessing the data	• `ER` (for `EASTERN_REGION`) • `WR` (for `WESTERN_REGION`)

Valid characters for specifying all label components include alphanumeric characters, underscores, and spaces. (Leading and trailing spaces are ignored.)

The following figure illustrates the three dimensions in which data can be logically classified, using levels, compartments, and groups.

Figure 2-1   Data Categorization with Levels, Compartments, and Groups



2.3.2 Level Sensitivity Components

A *level* is a ranking that denotes the sensitivity of the information it labels.

The more sensitive the information, the higher its level. The less sensitive the information, the lower its level.

Every label must include one level. Oracle Label Security permits defining up to 10,000 levels in a policy. For each level, the Oracle Label Security administrator defines a numeric form, a long character form, and the required short character form.

Table 2-2 shows examples of levels.

Table 2-3   Level Example

Numeric Form	Long Form	Short Form
`40`	`HIGHLY_SENSITIVE`	`HS`
`30`	`SENSITIVE`	`S`
`20`	`CONFIDENTIAL`	`C`
`10`	`PUBLIC`	`P`

Table 2-4 shows different ways of specifying levels.

Table 2-4 Forms of Specifying Levels

Form	Explanation
Numeric form, also called "tag"	The numeric form of the level can range from 0 to 9999. Sensitivity is ranked by this numeric value, so you must assign higher numbers to levels that are more sensitive, and lower numbers to levels that are less sensitive. In Table 2-3, 40 (`HIGHLY_SENSITIVE`) is a higher level than 30, 20, and 10. Administrators should avoid using sequential numbers for the numeric form of levels. A good strategy is to use even increments (such as 50 or 100) between levels. You can then insert additional levels between two preexisting levels, at a later date.
Long form	The long form of the level name can contain up to 80 characters.
Short form	The short form can contain up to 30 characters.

Although the administrator defines both long and short names for the level (and for each of the other label components), only the short form of the name is displayed upon retrieval of the records when the Oracle Label Security policy is in effect. When users manipulate the labels, they use only the short form of the component names.

Other sets of levels that users commonly define include `TOP_SECRET`, `SECRET`, `CONFIDENTIAL`, and `UNCLASSIFIED` or `TRADE_SECRET`, `PROPRIETARY`, `COMPANY_CONFIDENTIAL`, and `PUBLIC_DOMAIN`.

If only levels are used, a level 40 user (in this example) can access or alter any data row whose level is 40 or less.

Note:

All levels and labels (including `TOP_SECRET`, `SECRET`, `CONFIDENTIAL`, and so on) in this guide, are used as illustrations only.

2.3.3 Compartment Components

Compartments identify areas that describe the sensitivity of the labeled data, providing a finer level of granularity within a level.

Compartments associate the data with one or more security areas. All data related to a particular project can be labeled with the same compartment.

Table 2-5 shows examples of compartments.

Table 2-5 Compartment Example

Numeric Form	Long Form	Short Form
`85`	`FINANCIAL`	`FINCL`
`65`	`CHEMICAL`	`CHEM`
`45`	`OPERATIONAL`	`OP`

Table 2-6 shows different ways of specifying compartments.

Table 2-6 Forms of Specifying Compartments

Form	Explanation
Numeric form	The numeric form can range from 0 to 9999. It is unrelated to the numbers used for the levels. The numeric form of the compartment does not indicate greater or less sensitivity. Instead, it controls the display order of the short form compartment name in the label character string. For example, assume a label is created that has all three compartments listed in Table 2-5, and a level of `SENSITIVE`, whose short form is `S`. When this label is displayed in string format, it looks like the following, meaning `SENSITIVE`: `OPERATIONAL`, `CHEMICAL`, `FINANCIAL`: `S:OP,CHEM,FINCL` The display order follows the order of the numbers assigned to the compartments: 45 is lower than 65, and 65 is lower than 85. By contrast, if the number assigned to the `FINCL` compartment were 5, the character string format of the label would look like this: `S:FINCL,OP,CHEM`
Long form	The long form of the compartment name scan have up to 80 characters.
Short form	The short form can contain up to 30 characters.

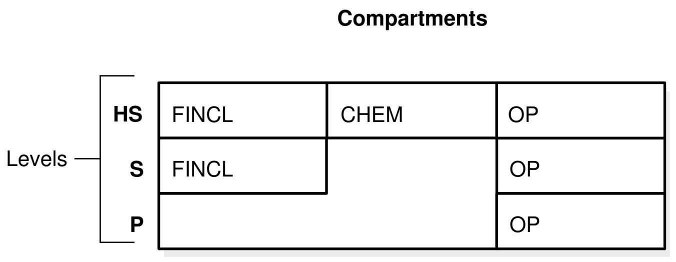
Compartments are optional. A label can contain zero or more compartments. Oracle Label Security permits defining up to 10,000 compartments.

Not all labels need to have compartments. For example, you can specify `HIGHLY_SENSITIVE` and `CONFIDENTIAL` levels with no compartments, and a `SENSITIVE` level that does contain compartments.

When you analyze the sensitivity of data, you may find that some compartments are only useful at specific levels.

The following figure shows how compartments can be used to categorize data.

Figure 2-2 Label Matrix



Here, compartments `FINCL`, `CHEM`, and `OP` are used with the level `HIGHLY_SENSITIVE` (40). The label `HIGHLY_SENSITIVE:FINCL`, `CHEM` indicates a level of 40 with the two named compartments. Compartment `FINCL` is not more sensitive than `CHEM`, nor is `CHEM` more sensitive than `FINCL`. Note also that some data in the protected table may not belong to any compartment.

If compartments are specified, then a user whose level would normally permit access to a row's data will nevertheless be prevented from such access unless the user's label also contains all the compartments appearing in that row's label.

2.3.4 Group Components

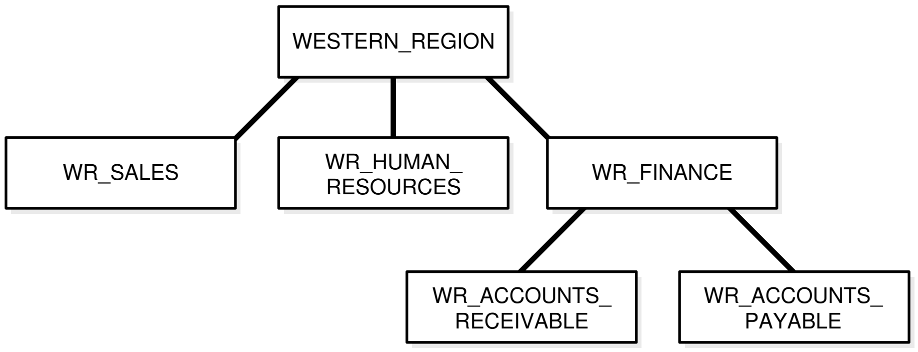
Groups identify organizations owning or accessing the data, such as EASTERN_REGION, WESTERN_REGION, WR_SALES.

All data pertaining to a certain department can have that department's group in the label. Groups are useful for the controlled dissemination of data and for timely reaction to organizational change. When a company reorganizes, data access can change right along with the reorganization.

Groups are hierarchical. You can label data based upon your organizational infrastructure. A group can thus be associated with a parent group.

Figure 2-3 shows how you can define a set of groups corresponding to the following organizational hierarchy.

Figure 2-3 Group Example



The WESTERN_REGION group includes three subgroups: WR_SALES, WR_HUMAN_RESOURCES, and WR_FINANCE. The WR_FINANCE subgroup is subdivided into WR_ACCOUNTS_RECEIVABLE and WR_ACCOUNTS_PAYABLE.

Table 2-7 shows how the organizational structure in this example can be expressed in the form of Oracle Label Security groups. Notice that the numeric form assigned to the groups affects display order only. The administrator specifies the hierarchy (that is, the parent/child relationships) separately.

Table 2-7 Group Example

Numeric Form	Long Form	Short Form	Parent Group
1000	WESTERN_REGION	WR	
1100	WR_SALES	WR_SAL	WR
1200	WR_HUMAN_RESOURCES	WR_HR	WR
1300	WR_FINANCE	WR_FIN	WR

Table 2-7 (Cont.) Group Example

Numeric Form	Long Form	Short Form	Parent Group
`1310`	`WR_ACCOUNTS_PAYABLE`	`WR_AP`	`WR_FIN`
`1320`	`WR_ACCOUNTS_RECEIVABLE`	`WR_AR`	`WR_FIN`

Table 2-8 shows different ways of specifying groups.

Table 2-8 Forms of Specifying Groups

Form	Explanation
Numeric form	The numeric form of the group can range from 0 to 9999, and it must be unique for each policy. The numeric form does not indicate any kind of ranking. It does not indicate a parent-child relationship, or greater or less sensitivity. It only controls the display order of the short form group name in the label character string. For example, assume that a label is created that has the level `SENSITIVE`, the compartment `CHEMICAL`, and the groups `WESTERN_REGION` and `WR_HUMAN_RESOURCES` as listed in Table 2-7. When displayed in string format, the label looks like this: `S:CHEM:WR,WR_HR` `WR` is displayed before `WR_HR` because 1000 comes before 1200.
Long form	The long form of the group name can contain up to 80 characters.
Short form	The short form can contain up to 30 characters.

Groups are optional; a label can contain zero or more groups. Oracle Label Security permits defining up to 10,000 groups.

All labels need not have groups. When you analyze the sensitivity of data, you may find that some groups are only used at specific levels. For example, you can specify `HIGHLY_SENSITIVE` and `CONFIDENTIAL` labels with no groups, and a `SENSITIVE` label that does contain groups.

Related Topics

- Releasability Using Inverse Groups
  Oracle Label Security can implement the releasability using inverse groups.

2.3.5 Industry Examples of Levels, Compartments, and Groups

Oracle Label Security levels, compartments, groups are designed to be implemented in various industries.

Table 2-9 illustrates the flexibility of Oracle Label Security levels, compartments, and groups, by listing typical ways in which they can be implemented in various industries.

Table 2-9 Typical Levels, Compartments, and Groups, by Industry

Industry	Levels	Compartments	Groups
Business to Business	TRADE_SECRET	MARKETING	AJAX_CORP
	PROPRIETARY	FINANCIAL	BILTWELL_CO
	COMPANY_CONFIDENTIAL	SALES	ACME_INC
	PUBLIC	PERSONNEL	ERSATZ_LTD
Financial Services	ACQUISITIONS	INSURANCE	CLIENT
	CORPORATE	EQUITIES	TRUSTEE
	CLIENT	TRUSTS	BENEFICIARY
	OPERATIONS	COMMERCIAL_LOANS	MANAGEMENT
		CONSUMER_LOANS	STAFF
Judicial	NATIONAL_SECURITY	CIVIL	ADMINISTRATION
	SENSITIVE	CRIMINAL	DEFENSE
	PUBLIC		PROSECUTION
			COURT
Health Care	PRIMARY_PHYSICIAN	PHARMACEUTICAL	CDC
	PATIENT_CONFIDENTIAL	INFECTIOUS_DISEASES	RESEARCH
	PATIENT_RELEASE		NURSING_STAFF
			HOSPITAL_STAFF
Defense	TOP_SECRET	ALPHA	UK
	SECRET	DELTA	NATO
	CONFIDENTIAL	SIGMA	SPAIN
	UNCLASSIFIED		

2.4 Label Syntax and Type

After label components are defined, you can create data labels by combining particular sets of level, compartments, and groups.

You can use the Oracle Enterprise Manager graphical user interface or a command line procedure. Character string representations of labels use the following syntax:

```
LEVEL:COMPARTMENT1,...,COMPARTMENTn:GROUP1,...,GROUPn
```

The text string specifying the label can have a maximum of 4,000 characters, including alphanumeric characters, spaces, and underscores. The labels are case-insensitive. You can enter them in uppercase, lowercase, or mixed case, but the string is stored in the data dictionary and displayed in uppercase. A colon is used as the delimiter between components. A comma is used as a delimiter between different entries (compartment or group) within a component. It is not necessary to enter trailing delimiters in this syntax.

For example, you can create valid labels such as these:

```
SENSITIVE:FINANCIAL,CHEMICAL:EASTERN_REGION,WESTERN_REGION
CONFIDENTIAL:FINANCIAL:VP_GRP
SENSITIVE
HIGHLY_SENSITIVE:FINANCIAL
SENSITIVE::WESTERN_REGION
```

When a valid data label is created, two additional things occur:

- The label is automatically designated as a valid data label. This functionality limits the labels that can be assigned to data. Most users, however, prefer to create the labels manually in order to limit data label proliferation.
- A numeric label tag is associated with the text string representing the label. It is this label tag, rather than the text string, that is stored in the policy label column of the protected table.

> **Note:**
>
> The dynamic creation of valid data labels uses the `TO_DATA_LABEL` function. Its usage should be tightly controlled.

Related Topics

- Inserting Labels Using TO_DATA_LABEL
  The `TO_DATA_LABEL` function can generate new labels dynamically.
- How Policy Label Column and Label Tags Work
  You should understand how policy label columns in a table or schema are created and filled.
- Label Tags
  You can create label tags, either manually or automatically generating them, that define the label components.

2.5 How Data Labels and User Labels Work Together

A user can access data only within the range of their own label authorizations.

A user has the following:

- Maximum and minimum levels
- A set of authorized compartments
- A set of authorized groups (and, implicitly, authorization for any subgroups)

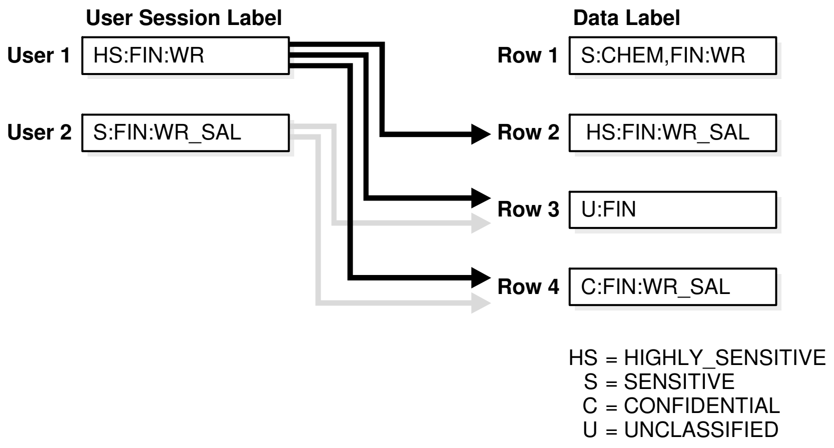
For example, suppose you have the following levels:

- `HIGHLY_SENSITIVE`, with the numeric form 40
- `SENSITIVE`, with the numeric form 30
- `CONFIDENTIAL`, with the numeric form 20
- `PUBLIC`, with the numeric form 10

If a user is assigned a maximum level of `SENSITIVE`, then the user potentially has access to `SENSITIVE`, `CONFIDENTIAL`, and `PUBLIC` data. The user has no access to `HIGHLY_SENSITIVE` data because this level is too high.

Figure 2-4 shows how data labels and user labels work together to provide access control in Oracle Label Security. While data labels are discrete, user labels are inclusive. Depending upon authorized compartments and groups, a user can potentially access data corresponding to all levels within their range.

Figure 2-4   Example: Data Labels and User Labels

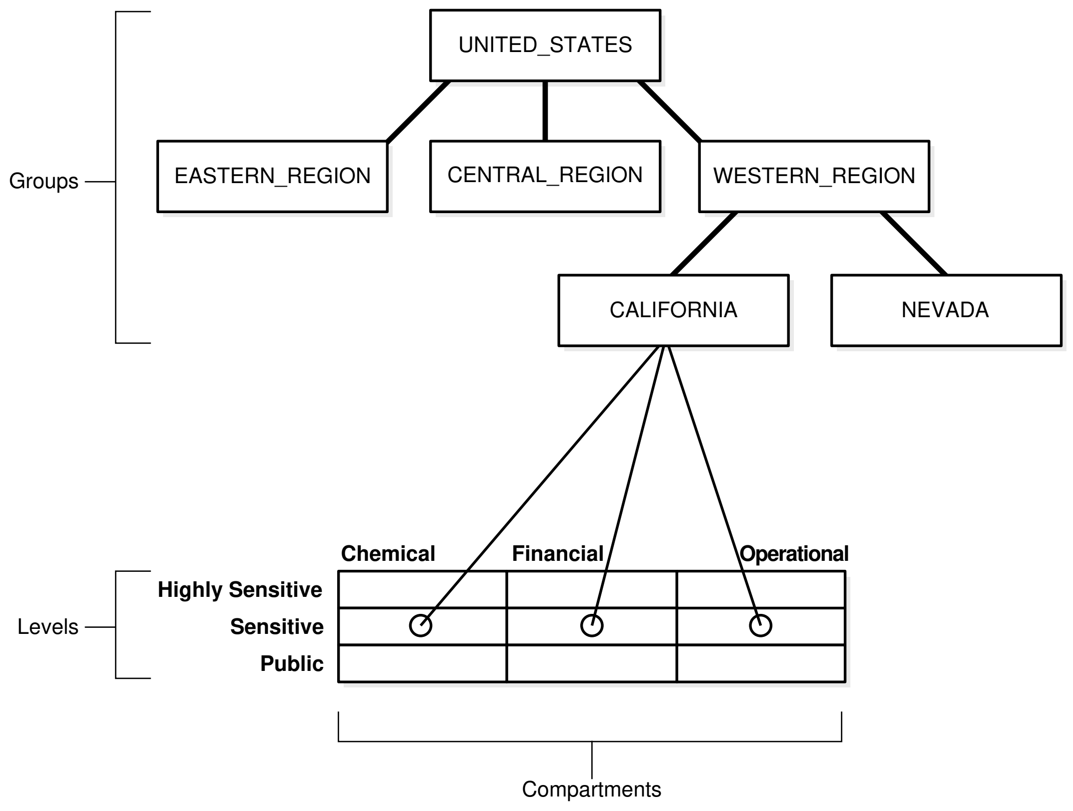


As shown in the figure, User 1 can access the rows 2, 3, and 4 because their maximum level is `HS`. They have access to the `FIN` compartment, and their access to group `WR` hierarchically includes group `WR_SAL`. They cannot access row 1 because they does not have the `CHEM` compartment. (A user must have authorization for *all* compartments in a row's data label to be able to access that row.)

User 2 can access rows 3 and 4. This user's maximum level is `S`, which is less than `HS` in row 2. Although the user has access to the `FIN` compartment, they only have authorization for group `WR_SAL`. So, they cannot access row 1.

Figure 2-5 shows how data pertaining to an organizational hierarchy fits into data levels and compartments.

Figure 2-5 How Label Components Interrelate



For example, the `UNITED_STATES` group includes three subgroups: `EASTERN_REGION`, `CENTRAL_REGION`, and `WESTERN_REGION`. The `WESTERN_REGION` subgroup is further subdivided into `CALIFORNIA` and `NEVADA`. For each group and subgroup, there may be data belonging to some of the valid compartments and levels within the database. So, there may be `SENSITIVE` data that is `FINANCIAL`, within the `CALIFORNIA` subgroup.

Note that data is generally labeled with a single group whereas users' labels form a hierarchy. If users have a particular group, then that group may implicitly include child groups. This way a user associated with the `UNITED_STATES` group has access to all data, but a user associated with `CALIFORNIA` would have access to data pertaining to only that subgroup.

2.6 Administration of Labels

Oracle Label Security provides administrative interfaces to define and manage the labels used in a database.

You define labels in Oracle Database using Oracle Label Security PL/SQL packages or by using Oracle Enterprise Manager. Initially, an administrator must define the levels, compartments, and groups that compose the labels, and then, the user can define the set of valid data labels for the contents of the database.

An administrator can apply a policy to individual tables in the database or to entire application schemas. Finally, the administrator assigns to each database user the label components (and privileges, if needed) required for the user's job function.

3
Access Controls and Privileges

Oracle provides access controls and privileges that determine the *type* of access users can have to labeled rows.

- Access Mediation
  To access data protected by an Oracle Label Security policy, a user must have authorizations based on the labels defined for the policy.
- How the Session Label and Row Label Work
  It is important to understand session labels and row labels.
- How User Authorizations Work
  Oracle Label Security provides authorizations set by the Oracle Label Security administrator and authorizations set by computed session labels.
- Evaluation of Labels for Access Mediation
  Oracle Label Security evaluates labels by comparing the user’s label components to the row’s label components.
- Oracle Label Security Privileges
  Oracle Label Security provides a set of system privileges, object privileges, and policy privileges.
- Working with Multiple Oracle Label Security Policies
  You can use multiple Oracle Label Security policies in both a single database environments and in a distributed environments.

Related Topics

- Understanding Data Labels and User Labels
  You should understand fundamental concepts of data labels and user labels.

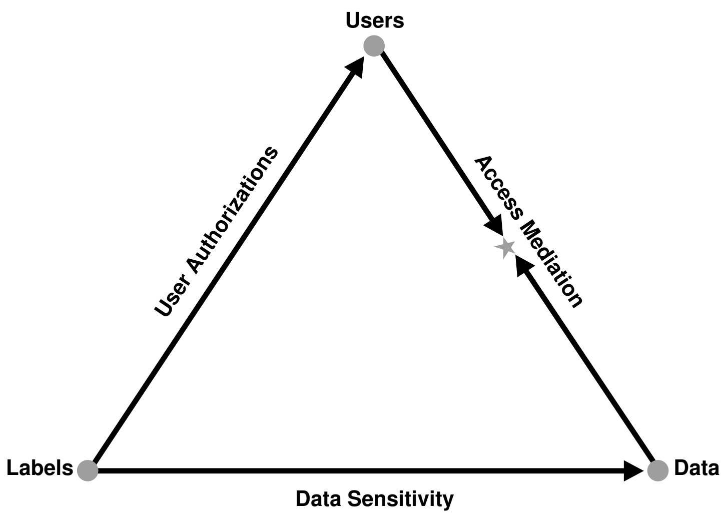
3.1 Access Mediation

To access data protected by an Oracle Label Security policy, a user must have authorizations based on the labels defined for the policy.

The following figure illustrates the relationships between users, data, and labels.

- Data labels specify the sensitivity of data rows.
- User labels provide the appropriate authorizations to users.
- Access mediation between users and rows of data depends on users' labels.

Figure 3-1 Relationships Between Users, Data, and Label



> **Note:**
>
> Oracle Label Security enforcement options affect how access controls apply to tables and schemas. This chapter assumes that all policy enforcement options are in effect.

Related Topics

- Oracle Label Security Policy Enforcement Options
  Oracle Label Security provides a set of policy enforcement options.

3.2 How the Session Label and Row Label Work

It is important to understand session labels and row labels.

- The Session Label
  Each Oracle Label Security user has authorizations that include special components.
- The Row Label
  When a user writes data without specifying its label, a default *row label* is assigned automatically, using the user's session label.
- Session Label Example
  The session label and the row label can fall anywhere within the range of the user's level, compartment, and group authorizations.

3.2.1 The Session Label

Each Oracle Label Security user has authorizations that include special components.

- A maximum and minimum level
- A set of authorized compartments

- A set of authorized groups
- For each compartment and group, a specification of read-only access, or read/write access

The administrator also specifies the user's initial session label when setting up these authorizations for the user. The *session label* is the particular combination of levels, compartments, and groups at which a user works at any given time. The user can change the session label to any combination of components for which the user is authorized.

Related Topics

- SA_SESSION Session Management PL/SQL Package
  The `SA_SESSION` PL/SQL package manages session behavior for user authorizations.

3.2.2 The Row Label

When a user writes data without specifying its label, a default *row label* is assigned automatically, using the user's session label.

However, the user can set the label for the written row, within certain restrictions on the components of the label they specify. The level of this label can be set to any level within the range specified by the administrator. For example, it can be set to the level of the user's current session label down to the user's minimum level. However, the compartments and groups for this row's new label are more restricted. The new label can include only those compartments and groups contained in the current session label and, among those, only the ones for which the user has write access.

When the administrator sets up the user authorizations, they also specify an initial default row label.

Related Topics

- SA_USER_ADMIN PL/SQL Package
  The `SA_USER_ADMIN` PL/SQL package manages user labels by label component.
- SA_SESSION Session Management PL/SQL Package
  The `SA_SESSION` PL/SQL package manages session behavior for user authorizations.

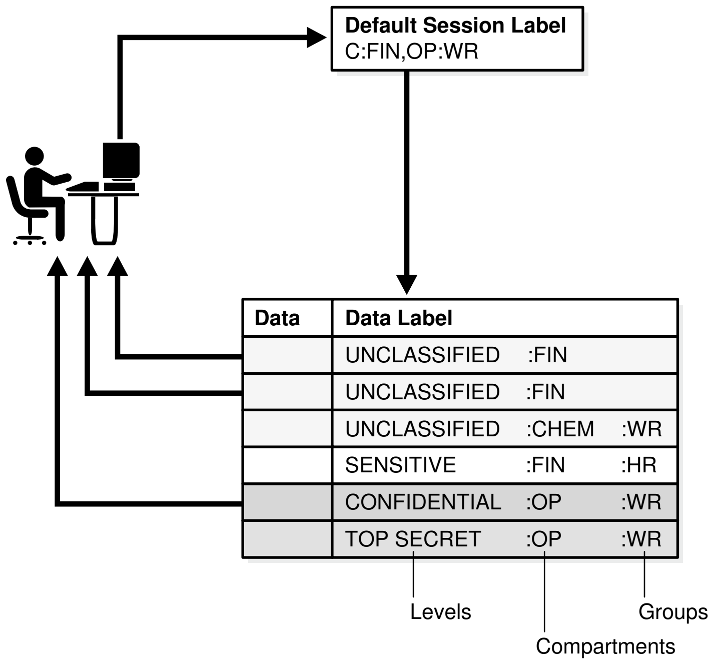
3.2.3 Session Label Example

The session label and the row label can fall anywhere within the range of the user's level, compartment, and group authorizations.

In the following figure, the user's maximum level is `SENSITIVE` and the minimum level is `UNCLASSIFIED`. However, the user's default session label is `C:FIN,OP:WR`. In this example, the administrator has set the user's session label so that the user connects to the database at the `CONFIDENTIAL` level.

Similarly, although the user is authorized for compartments `FIN` and `OP`, and group `WR`, the administrator could set the session label so that the user connects with only compartment `FIN` and group `WR`.

Figure 3-2   User Session Label



Related Topics

- SA_USER_ADMIN.SET_COMPARTMENTS
  The `SA_USER_ADMIN.SET_COMPARTMENTS` procedure assigns compartments to a user and identifies default values for the user's session label and row label.
- SA_USER_ADMIN.ALTER_COMPARTMENTS
  The `SA_USER_ADMIN.ALTER_COMPARTMENTS` procedure changes the write access, default label indicator, and row label indicator for the specified compartments.

3.3 How User Authorizations Work

Oracle Label Security provides authorizations set by the Oracle Label Security administrator and authorizations set by computed session labels.

- Authorizations Set by the Administrator
  The administrator explicitly sets authorizations for levels, compartments, and groups.
- Computed Session Labels
  Oracle Label Security automatically computes a number of labels based on the value of the session label.

3.3.1 Authorizations Set by the Administrator

The administrator explicitly sets authorizations for levels, compartments, and groups.

- Authorized Levels
  The administrator explicitly sets the level authorization for an Oracle Label Security policy.

- Authorized Compartments
  The administrator specifies the list of compartments that a user can place in their session label.
- Authorized Groups
  You must specify a list of groups that a user can place in a session label and grant write access for each group.

3.3.1.1 Authorized Levels

The administrator explicitly sets the level authorization for an Oracle Label Security policy.

Table 3-1 lists authorized levels that the administrator can set.

Table 3-1 Authorized Levels Set by the Administrator

Authorization	Meaning
User Max Level	The maximum ranking of sensitivity that a user can access during read and write operations
User Min Level	The minimum ranking of sensitivity that a user can access during write operations. The User Max Level must be equal to or greater than the User Min Level.
User Default Level	The level that is assumed by default when connecting to Oracle Database
User Default Row Level	The level that is used by default when inserting data into Oracle Database

For example, in Oracle Enterprise Manager, the administrator might set the following level authorizations for user Joe:

Type	Short Name	Long Name	Description
Maximum	`HS`	`HIGHLY_SENSITIVE`	User's highest level
Minimum	`P`	`PUBLIC`	User's lowest level
Default	`C`	`CONFIDENTIAL`	User's default level
Row	`C`	`CONFIDENTIAL`	Row level on `INSERT`

3.3.1.2 Authorized Compartments

The administrator specifies the list of compartments that a user can place in their session label.

The administrator must explicitly give write access to the user for each compartment. A user cannot directly insert, update, or delete a row that contains a compartment that they do not have authorization to write.

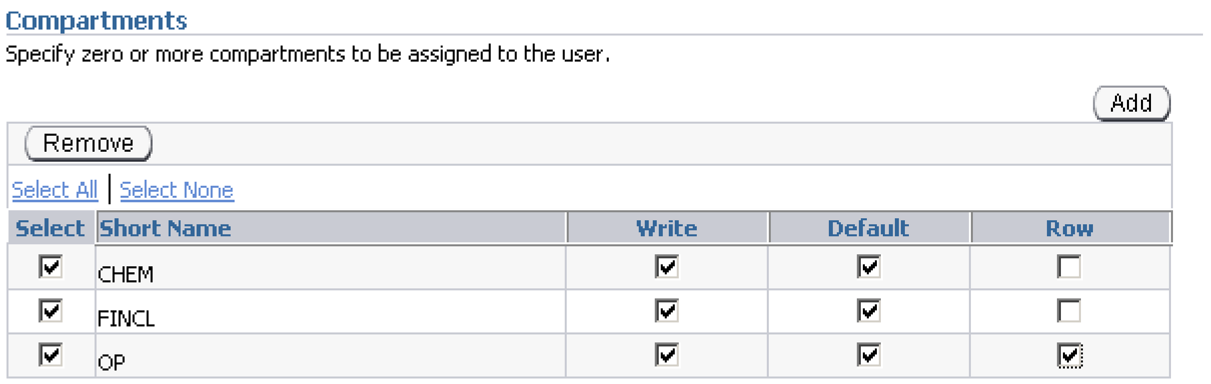
For example, in Oracle Enterprise Manager, the administrator might set the following compartment authorizations for user Joe:

Short Name	Long Name	WRITE	DEFAULT	ROW
`CHEM`	`CHEMICAL`	YES	YES	NO

Short Name	Long Name	WRITE	DEFAULT	ROW
`FINCL`	`FINANCIAL`	YES	YES	NO
`OP`	`OPERATIONAL`	YES	YES	YES

In Figure 3-3, the row designation indicates whether the compartment should be used as part of the default row label for newly inserted data. Note also that the policy option must be in effect for this setting to be valid.

Figure 3-3 Setting Up Authorized Compartments In Enterprise Manager



Compartments

Specify zero or more compartments to be assigned to the user.

Select	Short Name	Write	Default	Row
✓	CHEM	✓	✓	☐
✓	FINCL	✓	✓	☐
✓	OP	✓	✓	✓

3.3.1.3 Authorized Groups

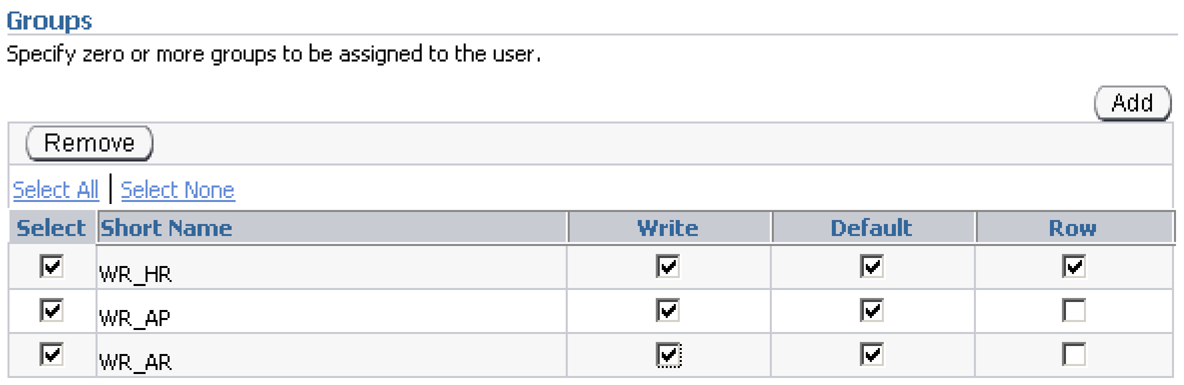
You must specify a list of groups that a user can place in a session label and grant write access for each group.

For example, in Oracle Enterprise Manager, the administrator might set the following group authorizations:

Short Name	Long Name	WRITE	DEFAULT	ROW	Parent
`WR_HR`	`WR_HUMAN_RESOURCES`	YES	YES	YES	`WR`
`WR_AP`	`WR_ACCOUNTS_PAYABLE`	YES	YES	NO	`WR_FIN`
`WR_AR`	`WR_ACCOUNTS_RECEIVABLE`	YES	YES	NO	`WR_FIN`

In Figure 3-4, the row designation indicates whether the group should be used as part of the default row label for newly inserted data. Note also that the `LABEL_DEFAULT` policy option must be in effect for this setting to be valid.

Figure 3-4 Setting Up Authorized Groups in Enterprise Manager



Specify zero or more groups to be assigned to the user.

Select	Short Name	Write	Default	Row
✓	WR_HR	✓	✓	✓
✓	WR_AP	✓	✓	☐
✓	WR_AR	✓	✓	☐

Related Topics

- LABEL_DEFAULT: Using the Session's Default Row Label
  A user can update a row without specifying a label value, because the updated row uses its original label.

3.3.2 Computed Session Labels

Oracle Label Security automatically computes a number of labels based on the value of the session label.

Table 3-2 lists the computed session labels.

Table 3-2 Computed Session Labels

Computed Label	Definition
Maximum Read Label	The user's maximum level combined with any combination of compartments and groups for which the user is authorized.
Maximum Write Label	The user's maximum level combined with the compartments and groups for which the user has been granted write access.
Minimum Write Label	The user's minimum level.
Default Read Label	The single default level combined with compartments and groups that have been designated as default for the user.
Default Write Label	A subset of the default read label, containing the compartments and groups to which the user has been granted write access. The level component is equal to the level default in the read label. This label is automatically derived from the read label based on the user's write authorizations.
Default Row Label	The combination of components between the user's minimum write label and the maximum write label, which has been designated as the default value for the data label for inserted data.

Related Topics

- Computed Labels with Inverse Groups
  Inverse groups affect computed label values.

3.4 Evaluation of Labels for Access Mediation

Oracle Label Security evaluates labels by comparing the user’s label components to the row’s label components.

This way, the Oracle Label Security policy can determine whether the user can access the data. This enables Oracle Label Security to evaluate whether the user is authorized to perform the requested operation on the data in the row.

- About Read and Write Access
  Although data labels are stored in a column within data records, information about user authorizations is stored in relational tables.
- How Oracle Label Security Algorithm for Read Access Works
  The `READ_CONTROL` enforcement determines the ability to read data in a row.
- How the Oracle Label Security Algorithm for Write Access Works
  In the context of Oracle Label Security, `WRITE_CONTROL` enforcement determines the ability to insert, update, or delete data in a row.

3.4.1 About Read and Write Access

Although data labels are stored in a column within data records, information about user authorizations is stored in relational tables.

When a user logs on, the tables are used to dynamically generate user labels for use during the session.

- Difference Between Read and Write Operations
  Two fundamental types of access mediation on Data Manipulation language (DML) operations exist within protected tables: read access and write access.
- Propagation of Read/Write Authorizations on Groups
  When groups are organized hierarchically, a user's assigned groups include all subgroups that are subordinate to the group to which the user belongs.

3.4.1.1 Difference Between Read and Write Operations

Two fundamental types of access mediation on Data Manipulation language (DML) operations exist within protected tables: read access and write access.

The user has a maximum authorization for the data they can read; the user's write authorization is a subset of that. The minimum write level controls the user's ability to disseminate data by lowering its sensitivity. The user cannot write data with a level lower than the minimum level the administrator assigned to this user.

In addition, there are separate lists of compartments and groups for which the user is authorized; that is, for which the user has at least read access. An access flag indicates whether the user can also write to individual compartments or groups.

3.4.1.2 Propagation of Read/Write Authorizations on Groups

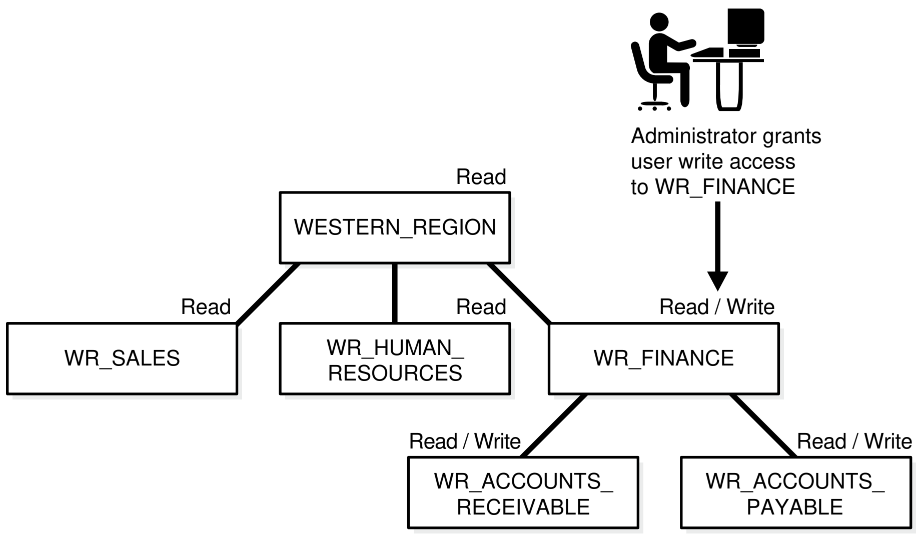
When groups are organized hierarchically, a user's assigned groups include all subgroups that are subordinate to the group to which the user belongs.

In this case, the user's read/write authorizations on a parent group flow down to all the subgroups.

Consider the parent group WESTERN_REGION, with three subgroups as illustrated in Figure 3-5. If the user has read access to WESTERN_REGION, then the read access is also granted to the three subgroups. The administrator can give the user write access to subgroup WR_FINANCE, without granting write access to the WESTERN_REGION parent group (or to the other subgroups). On the other hand, if the user has read/write access on WESTERN_REGION, then read/write access is also granted on all of the subgroups subordinate to it in the tree.

Write authorization on a group does not give a user write authorization on the parent group. If a user has read-only access to WESTERN_REGION and WR_FINANCE, then the administrator can grant write access to WR_ACCOUNTS_RECEIVABLE, without affecting the read-only access to the higher-level groups.

Figure 3-5 Subgroup Inheritance of Read/Write Access



Related Topics

- How Inverse Groups Work
  Inverse groups are implemented in a special way and are organized to suit the needs of Oracle Label Security.

3.4.2 How Oracle Label Security Algorithm for Read Access Works

The READ_CONTROL enforcement determines the ability to read data in a row.

The following rules are used, in the sequence listed, to determine a user's read access to a row of data:

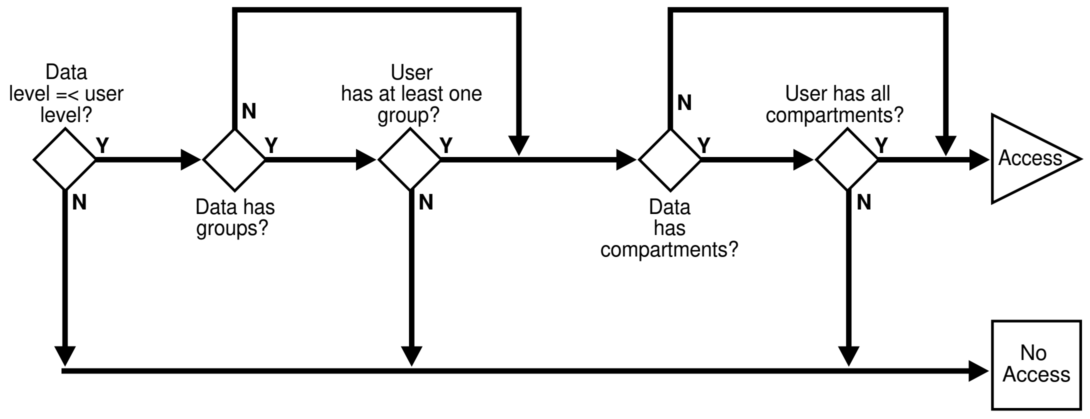
1. The user's level must be *greater than or equal to* the level of the data.
2. The user's label must include *at least one of the groups* that belong to the data (or the parent group of one such subgroup).
3. The user's label must include *all the compartments* that belong to the data.

If the user's label passes these tests, then it is said to dominate the row's label.

Note that there is no notion of read or write access connected with levels. This is because the administrator specifies a range of levels (minimum to maximum) within which a user can potentially read and write. At any time, the user can read all data equal to or less than the current session level. No privileges (other than `FULL`) allow the user to write below the minimum authorized level.

Figure 3-6 illustrates how the label evaluation process proceeds from levels to groups to compartments. Note that if the data label is null or invalid, then the user is denied access.

Figure 3-6   Label Evaluation Process for Read Access



As a read access request comes in, Oracle Label Security evaluates each row to determine the following:

1. Is the user's level equal to, or greater than, the level of the data?
2. If so, does the user have access to at least one of the groups present in the data label?
3. If so, does the user have access to all the compartments present in the data label? (That is, are the data's compartments a subset of the user's compartments?)

If the answer is no at any stage in this evaluation process, then Oracle Label Security denies access to the row and moves on to evaluate the next row of data.

Oracle Label Security policies allow user sessions to read rows at their label and below, which is called *reading down*. Sessions cannot read rows at labels that they do not dominate.

For example, if you are logged in at `SENSITIVE:ALPHA,BETA`, you can read a row labeled `SENSITIVE:ALPHA` because your label dominates that of the row. However, you cannot read a row labeled `SENSITIVE:ALPHA,GAMMA` because your label does not dominate that of the row.

Note that the user can gain access to the rows otherwise denied, if they have special Oracle Label Security privileges.

Related Topics

- Privileges Defined by Oracle Label Security Policies
  Oracle Label Security supports special privileges that allow authorized users to bypass certain parts of the policy.

- How the Access Control Enforcement Options Work
  Access control options limit the rows accessible for `SELECT`, `UPDATE`, `INSERT`, or `DELETE` operations to only those rows whose labels meet established policies.

3.4.3 How the Oracle Label Security Algorithm for Write Access Works

In the context of Oracle Label Security, `WRITE_CONTROL` enforcement determines the ability to insert, update, or delete data in a row.

`WRITE_CONTROL` enables you to control data access with ever finer granularity. Granularity increases when compartments are added to levels. It increases again when groups are added to compartments. Access control becomes even more fine grained when you can manage the user's ability to write the data that they can read.

To determine whether a user can write a particular row of data, Oracle Label Security evaluates the following rules, in the order given:

1. The level in the data label must be greater than or equal to the user's minimum level and less than or equal to the user's session level.
2. When groups are present, the user's label must include *at least one of the groups with write access* that appear in the data label (or the parent of one such subgroup). In addition, the user's label must include *all the compartments* in the data label.
3. When no groups are present, the user's label must have write access on *all of the compartments* in the data label.

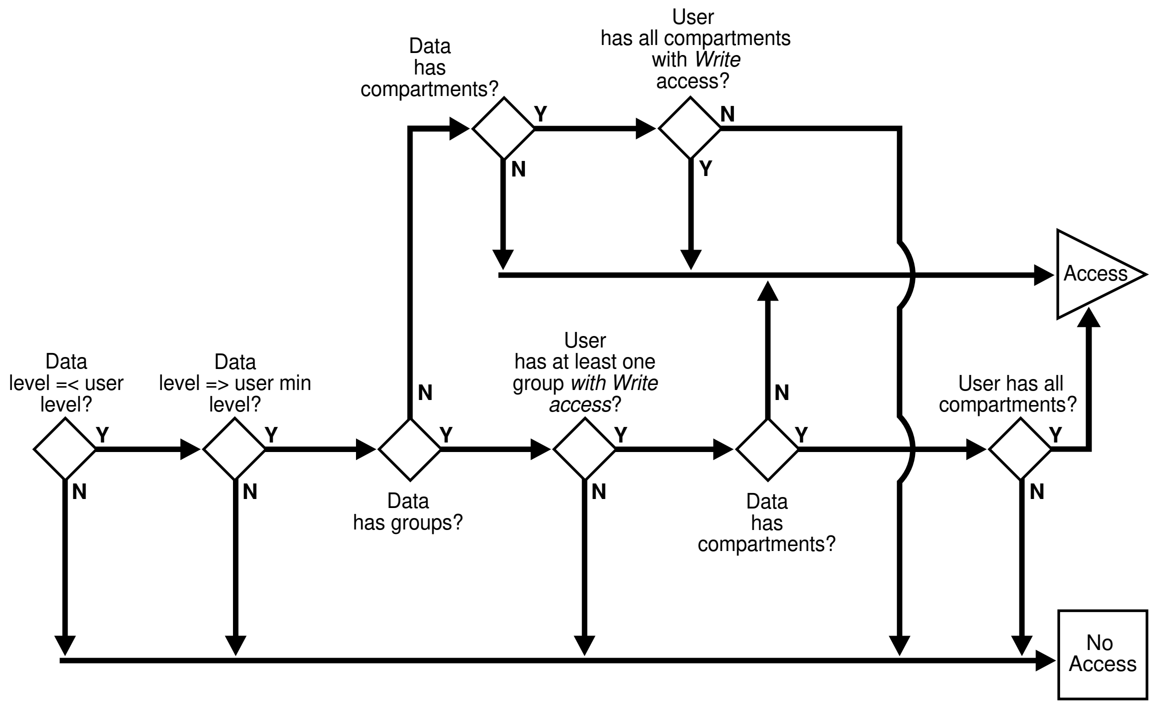
To state tests 2 and 3 another way:

- If the label has *no* groups, then the user must have write access on all the compartments in the label in order to write the data.
- If the label *does* have groups and the user has write access to one of the groups, they only need read access to the compartments in order to write the data.

Just as with read operations, the label evaluation process proceeds from levels to groups to compartments. Note that the user cannot write any data below the authorized minimum level, nor above the current session level. The user can always read below the minimum level.

Figure 3-7 illustrates how the process works with `INSERT`, `UPDATE`, and `DELETE` operations. Note that if the data label is null or invalid, then the user is denied access.

Figure 3-7 Label Evaluation Process for Write Access


Data level =< user level?
Data level => user min level?
Data has groups?
Data has compartments?
User has all compartments with *Write* access?
User has at least one group *with Write access*?
Data has compartments?
User has all compartments?
Y
N
Access
No Access

As an access request comes in, Oracle Label Security evaluates each row to determine the following:

1. Is the data's level equal to, or less than the level of the user session?
2. Is the data's level equal to, or greater than the user's minimum level?
3. If the data's level falls within the foregoing bounds, then does the user have write access to at least one of the groups present in the data label?
4. If so, does the user have access to all the compartments with at least read access that are present in the data label?
5. If there are no groups but there are compartments, then does the user have write access to all of the compartments?

If the answer is no at any stage in this evaluation process, then Oracle Label Security denies access to the row, and moves on to evaluate the next row of data.

Consider a situation in which your session label is `S:ALPHA,BETA` but you have write access to only compartment `ALPHA`. In this case, you can read a row with the label `S:ALPHA,BETA` but you cannot update it.

In summary, write access is enforced on `INSERT`, `UPDATE` and `DELETE` operations upon the data in the row.

In addition, each user may have an associated minimum level below which the user cannot write. The user cannot update or delete any rows labeled with levels below the minimum, and cannot insert a row with a row label containing a level less than the minimum.

Related Topics

- How the Access Control Enforcement Options Work
  Access control options limit the rows accessible for `SELECT`, `UPDATE`, `INSERT`, or `DELETE` operations to only those rows whose labels meet established policies.

3.5 Oracle Label Security Privileges

Oracle Label Security provides a set of system privileges, object privileges, and policy privileges.

- Privileges Defined by Oracle Label Security Policies
  Oracle Label Security supports special privileges that allow authorized users to bypass certain parts of the policy.
- Special Access Privileges
  A user's authorizations can be modified with any of four privileges.
- Special Row Label Privileges
  Once the label on a row has been set, Oracle Label Security privileges are required to modify the label.
- System Privileges, Object Privileges, and Policy Privileges
  Oracle Label Security privileges are different from the standard Oracle Database system and object privileges.
- Access Mediation and Views
  Prior to accessing data through a view, the users must have the appropriate system and object privileges on the view.
- Access Mediation and Program Unit Execution
  The privileges with which procedures that are owned by different users are run differently in Oracle Database and Oracle Label Security.
- Access Mediation and Policy Enforcement Options
  An administrator can choose from among a set of policy enforcement options when applying an Oracle Label Security policy to individual tables.

3.5.1 Privileges Defined by Oracle Label Security Policies

Oracle Label Security supports special privileges that allow authorized users to bypass certain parts of the policy.

Table 3-3 summarizes the full set of privileges that can be granted to users or trusted stored program units. Each privilege is more fully discussed after the table.

Table 3-3 Oracle Label Security Privileges

Security Privilege	Explanation
`READ`	Allows read access to all data protected by the policy
`FULL`	Allows full read and write access to all data protected by the policy
`COMPACCESS`	Allows a session access to data authorized by the row's compartments, independent of the row's groups
`PROFILE_ACCESS`	Allows a session to change its labels and privileges to those of a different user

Table 3-3 (Cont.) Oracle Label Security Privileges

Security Privilege	Explanation
`WRITEUP`	Allows users to set or raise only the level, within a row label, up to the maximum level authorized for the user. (Active only if `LABEL_UPDATE` is active.)
`WRITEDOWN`	Allows users to set or lower the level, within a row label, to any level equal to or greater than the minimum level authorized for the user. (Active only if `LABEL_UPDATE` is active.)
`WRITEACROSS`	Allows a user to set or change groups and compartments of a row label, but does not allow changes to the level. (Active only if `LABEL_UPDATE` is active.)

3.5.2 Special Access Privileges

A user's authorizations can be modified with any of four privileges.

- READ Privilege
  A user with the `READ` privilege can read all data protected by the policy, regardless of the authorizations or session label.
- FULL Privilege
  The `FULL` privilege has the same effect and benefits as the `READ` privilege, with one difference.
- COMPACCESS Privilege
  The `COMPACCESS` privilege allows a user to access data based on the row label's compartments, independent of the row label's groups.
- PROFILE_ACCESS Privilege
  The `PROFILE_ACCESS` privilege allows a session to change its session labels and session privileges to those of a different user.

3.5.2.1 READ Privilege

A user with the `READ` privilege can read all data protected by the policy, regardless of the authorizations or session label.

The user does not even need to have label authorizations.

Note, in addition, that a user with `READ` privilege can *write* to any data rows for which they have write access, based on any label authorizations.

Note:

Access mediation is still enforced on `UPDATE`, `INSERT`, and `DELETE` operations.

This privilege is useful for system administrators who need to export data but who should not be allowed to change data. It is also useful for people who must run reports and compile information but not change data. The `READ` privilege enables optimal performance on SELECT statements, because the system behaves as though the Oracle Label Security policy were not even present.

3.5.2.2 FULL Privilege

The FULL privilege has the same effect and benefits as the READ privilege, with one difference.

A user with the FULL privilege can also *write* to all the data. For a user with the FULL privilege, the READ and WRITE algorithms are not enforced.

Oracle system and object authorizations are still enforced for users who have been granted the FULL privilege. For example, a user must still have the SELECT system privilege on the application table. The FULL authorization turns off the access mediation check at the individual row level.

3.5.2.3 COMPACCESS Privilege

The COMPACCESS privilege allows a user to access data based on the row label's compartments, independent of the row label's groups.

If a row label has no compartments, then access is determined by the group authorizations. However, when compartments do exist and access to them is authorized, then the group authorization is bypassed. This allows a privileged user whose label matches all the compartments of the data to access any data in any particular compartment, independent of what groups may own or otherwise be allowed access to the data.

Figure 3-8 shows the label evaluation process for read access with the COMPACCESS privilege. Note that if the data label is null or invalid, then the user is denied access.

Figure 3-8 Label Evaluation Process for Read Access with COMPACCESS Privilege

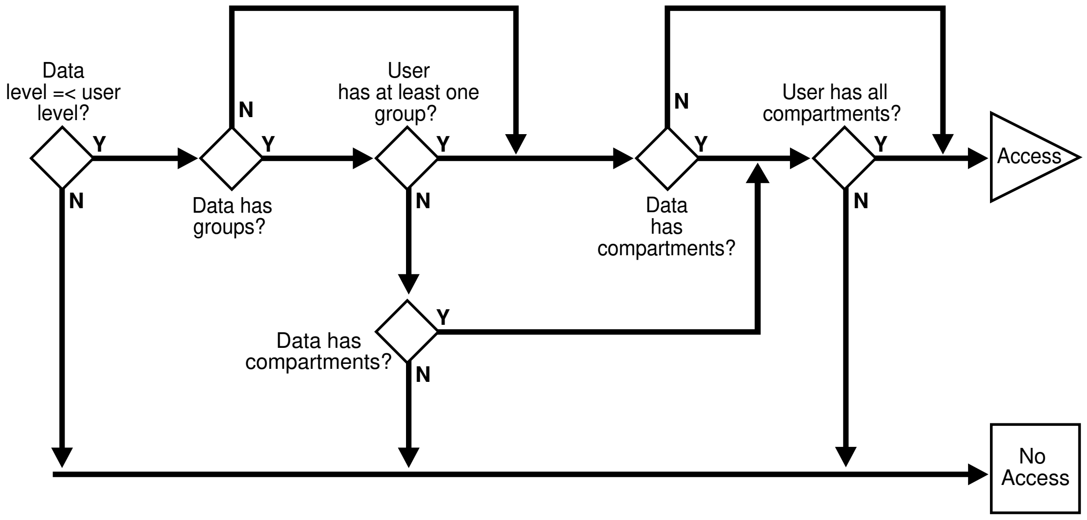
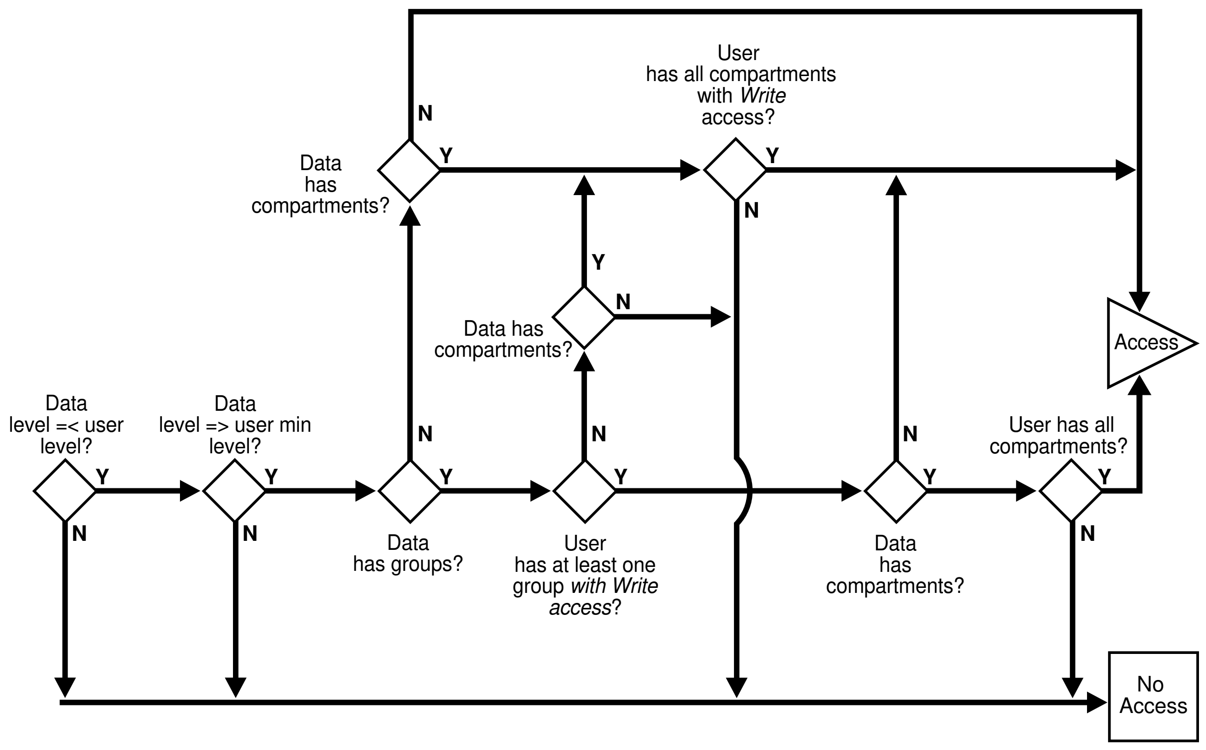


Figure 3-9 shows the label evaluation process for write access with COMPACCESS privilege. Note that if the data label is null or invalid, then the user is denied access.

Figure 3-9 Label Evaluation Process for Write Access with COMPACCESS Privilege


Data level =< user level?
Data level => user min level?
Data has groups?
Data has compartments?
Data has compartments?
User has at least one group *with Write access*?
User has all compartments with *Write* access?
Data has compartments?
User has all compartments?
Access
No Access
Y
N

3.5.2.4 PROFILE_ACCESS Privilege

The `PROFILE_ACCESS` privilege allows a session to change its session labels and session privileges to those of a different user.

This is a very powerful privilege, because the user can potentially become a user with the `FULL` privilege. This privilege cannot be granted to a trusted stored program unit.

3.5.3 Special Row Label Privileges

Once the label on a row has been set, Oracle Label Security privileges are required to modify the label.

Note that the `LABEL_UPDATE` enforcement option must be on for these label modification privileges to be enforced. When a user updates a row label, the new label and old label are compared, and the required privileges are determined.

The special row label privileges include:

- WRITEUP Privilege
  The `WRITEUP` privilege enables the user to raise the level of data within a row, without compromising the compartments or groups.
- WRITEDOWN Privilege
  The `WRITEDOWN` privilege enables the user to lower the level of data within a row, without compromising the compartments or groups.

- WRITEACROSS Privilege
  The WRITEACROSS privilege allows the user to change the compartments and groups of data, without altering its sensitivity level.

3.5.3.1 WRITEUP Privilege

The WRITEUP privilege enables the user to raise the level of data within a row, without compromising the compartments or groups.

This privilege enables a user to raise the level up to their maximum authorized level. You can find the privileges that users have by querying the ALL_SA_USER_PRIVS data dictionary view.

For example, an authorized user can raise the level of a data row that has a level lower than their own minimum level. If a row is UNCLASSIFIED and the user's maximum level is SENSITIVE, then the row's level can be raised to SENSITIVE. It can be raised above the current session level, but it cannot change the compartments.

3.5.3.2 WRITEDOWN Privilege

The WRITEDOWN privilege enables the user to lower the level of data within a row, without compromising the compartments or groups.

The user can lower the level to any level equal to or greater than their minimum authorized level. You can find the privileges that have been granted to a user by querying the ALL_SA_USER_PRIVS data dictionary view.

3.5.3.3 WRITEACROSS Privilege

The WRITEACROSS privilege allows the user to change the compartments and groups of data, without altering its sensitivity level.

This guarantees, for example, that SENSITIVE data remains at the SENSITIVE level, but at the same time enables the data's dissemination to be managed.

It lets the user change compartments and groups to anything that is currently defined as a valid compartment or group within the policy, while maintaining the level. With the WRITEACROSS privilege, a user with read access to one group (or more) can write to a different group without explicitly being given access to it.

You can find the privileges that have been granted to a user by querying the ALL_SA_USER_PRIVS data dictionary view.

3.5.4 System Privileges, Object Privileges, and Policy Privileges

Oracle Label Security privileges are different from the standard Oracle Database system and object privileges.

Table 3-4 Types of Privilege

Source	Privileges	Definition
Oracle Database	System Privileges	The right to run a particular type of SQL statement
Oracle Database	Object Privileges	The right to access another user's object

Table 3-4 (Cont.) Types of Privilege

Source	Privileges	Definition
Oracle Label Security	Policy Privileges	The ability to bypass certain parts of the label security policy

Oracle Database enforces the discretionary access control privileges that a user has been granted. By default, a user has no privileges except those granted to the `PUBLIC` user group. A user must explicitly be granted the appropriate privilege to perform an operation.

For example, to read an object in Oracle Database, you must either be the object's owner, or be granted the `SELECT` privilege on the object, or be granted the `SELECT ANY TABLE` system privilege. Similarly, to update an object, you must either be the object's owner, or be granted the `UPDATE` privilege on the object, or be granted the `UPDATE ANY TABLE` privilege.

Related Topics

- *Oracle Database Security Guide*

3.5.5 Access Mediation and Views

Prior to accessing data through a view, the users must have the appropriate system and object privileges on the view.

If the underlying table (on which the view is based) is protected by Oracle Label Security, then the user of the view must have authorization from Oracle Label Security to access specific rows of labeled data.

3.5.6 Access Mediation and Program Unit Execution

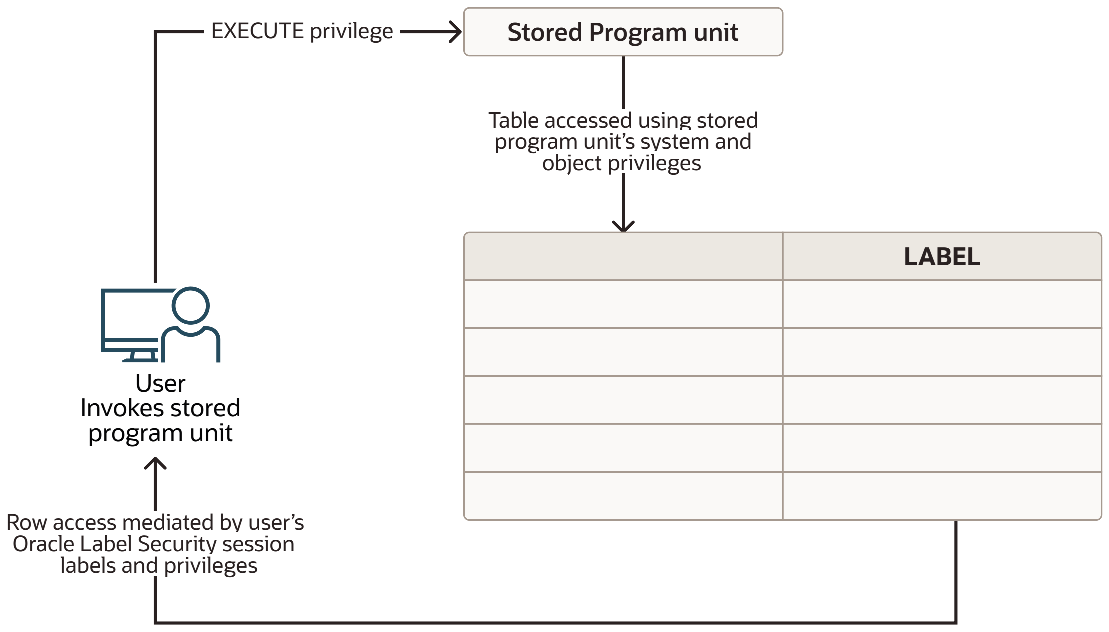
The privileges with which procedures that are owned by different users are run differently in Oracle Database and Oracle Label Security.

For example, in Oracle Database, if `user1` runs a procedure that belongs to `user2`, then the procedure runs with `user2`'s system and object privileges. You can find the privileges that have been granted to a user by querying the `DBA_SYS_PRIVS` data dictionary view. However, any procedure run by `user1` runs with `user1`'s own Oracle Label Security labels and privileges. This is true even when `user1` runs stored program units owned by other users.

Figure 3-10 illustrates this process:

- Stored program units run with the DAC privileges of the procedure's owner (`user2`).
- In addition, stored program units accessing tables protected by Oracle Label Security mediate access to data rows based on the label attached to the row, and the Oracle Label Security labels and privileges of the invoker of the procedure (`user1`).

Figure 3-10 Stored Program Unit Execution


EXECUTE privilege
Stored Program unit
Table accessed using stored program unit's system and object privileges
LABEL
User
Invokes stored program unit
Row access mediated by user's Oracle Label Security session labels and privileges

Stored program units can become *trusted* when an administrator assigns them Oracle Label Security privileges. A stored program unit can be run with its own autonomous Oracle Label Security privileges rather than those of the user who calls it. For example, if you possess no Oracle Label Security privileges in your own right but run a stored program unit that has the `WRITEDOWN` privilege, then you can update labels. In this case, the privileges used are those of the stored program unit, and not your own.

Trusted program units can encapsulate privileged operations in a controlled manner. By using procedures, packages, and functions with assigned privileges, you may be able to access data that your own labels and privileges would not authorize. For example, to perform aggregate functions over all data in a table, not just the data visible to you, you might use a trusted program set up by an administrator. This way program units can thus perform operations on behalf of users, without the need to grant privileges directly to users.

Related Topics

- Administering and Using Trusted Stored Program Units
  You can use trusted stored program units to enhance system security.

3.5.7 Access Mediation and Policy Enforcement Options

An administrator can choose from among a set of policy enforcement options when applying an Oracle Label Security policy to individual tables.

These options enable enforcement to be tailored differently for each database table. In addition to the access controls based on the labels, a SQL predicate can also be associated with each table. The predicate can further define which rows in the table are accessible to the user.

In cases where the label to be associated with a new or updated row should be automatically computed, an administrator can specify a labeling function when applying the policy. That function will thereafter always be invoked to provide the data labels written under that policy, because active labeling functions take precedence over any alternative means of supplying a label.

Except where noted, this guide assumes that all enforcement options are in effect.

Related Topics

- Implementing Policy Enforcement Options and Labeling Functions
  You can customize the enforcement of Oracle Label Security policies and implement labeling functions.
- Labeling Functions
  Labeling functions can compute and return a label using resources such as context variables (for example, date or username) and data values.
- SA_POLICY_ADMIN.APPLY_SCHEMA_POLICY
  The `SA_POLICY_ADMIN.APPLY_SCHEMA_POLICY` procedure applies a policy to all of the tables in a schema and enables the policy for these tables.

3.6 Working with Multiple Oracle Label Security Policies

You can use multiple Oracle Label Security policies in both a single database environments and in a distributed environments.

- Multiple Oracle Label Security Policies in a Single Database
  Several Oracle Label Security policies can protect data in a single database.
- Multiple Oracle Label Security Policies in a Distributed Environment
  In a distributed environment that uses Oracle Label Security, remote connections are controlled by Oracle Label Security.

3.6.1 Multiple Oracle Label Security Policies in a Single Database

Several Oracle Label Security policies can protect data in a single database.

Each defined policy is associated with a set of labels used only by that policy. Data labels are constrained by the set of defined labels for each policy.

Each policy may protect a different table, but multiple policies can also apply to a single table. To access data, you must have label authorizations for all policies protecting that data. To access any particular row, you must be authorized by *all* policies protecting the table containing your desired rows. If you require privileges, then you may need privileges for all of the policies affecting your work.

3.6.2 Multiple Oracle Label Security Policies in a Distributed Environment

In a distributed environment that uses Oracle Label Security, remote connections are controlled by Oracle Label Security.

Related Topics

- Using Oracle Label Security with a Distributed Database
  You should understand the special considerations for using Oracle Label Security in a distributed configuration.

Part II

Using Oracle Label Security Functionality

Part II explains how to work with Oracle Label Security functionality.

- Registering and Logging in to Oracle Label Security
  Before using Oracle Label Security, you must register (configure) it with the database and then you can log in to Oracle Label Security.
- Creating an Oracle Label Security Policy
  An Oracle Label Security policy is a named set of commands that implements Oracle Label Security.
- Working with Labeled Data
  You can manage labeled data, view that data of security attributes for a session, and change the value of session attributes.

4

Registering and Logging in to Oracle Label Security

Before using Oracle Label Security, you must register (configure) it with the database and then you can log in to Oracle Label Security.

- Registering Oracle Label Security with an Oracle Database
  You must register Oracle Label Security with the database in which you plan to use it.
- Security Guideline for Managing the LBACSYS User and the LBAC_DBA Role
  As a good practice, for day-to-day use, grant the `LBAC_DBA` database role to trusted users who will administer Oracle Label Security.
- Logging in to Cloud Control or SQL*Plus for Oracle Label Security
  After you complete the Oracle Label Security registration and enablement process, you can begin using it.

4.1 Registering Oracle Label Security with an Oracle Database

You must register Oracle Label Security with the database in which you plan to use it.

- About Registering Oracle Label Security
  When you install Oracle Database, Oracle Label Security is included in the installation but by default it is not enabled.
- Checking if Oracle Label Security Has Been Registered and Enabled
  You can query the `DBA_OLS_STATUS` data dictionary view to find if Oracle Label Security has already been registered and enabled.
- Registering and Enabling Oracle Label Security from SQL*Plus
  You can both register and enable Oracle Label Security from SQL*Plus.
- Registering and Enabling Oracle Label Security Using DBCA
  You can both register and enable Oracle Label Security using Database Configuration Assistant.
- Recompiling Oracle Label Security
  If invalid objects appear in Oracle Label Security after you have upgraded, then you must recompile Oracle Label Security to remove these invalid objects.

4.1.1 About Registering Oracle Label Security

When you install Oracle Database, Oracle Label Security is included in the installation but by default it is not enabled.

This applies if you install Oracle Database by using the Typical installation method in Database Configuration Assistant (DBCA). If you install using a custom installation, then you can optionally register Oracle Label Security from DBCA as part of the Oracle Database installation process.

If you installed using the Typical installation method, then you must manually register (enable) Oracle Label Security before you can use it. You have the choice of using either SQL*Plus or DBCA to perform the registration.

After Oracle Label Security is registered, you must enable the default Oracle Label Security user account, `LBACSYS`. Afterward, you can disable and re-enable Oracle Label Security when necessary.

Only register Oracle Label Security in the pluggable databases (PDBs) in which you plan to create Oracle Label Security policies. Because Oracle Label Security is not designed to protect data dictionary objects, you cannot create policies in the root.

4.1.2 Checking if Oracle Label Security Has Been Registered and Enabled

You can query the `DBA_OLS_STATUS` data dictionary view to find if Oracle Label Security has already been registered and enabled.

1. Log into 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, query the `DBA_PDBS` data dictionary view. To check the current PDB, run the `show con_name` command.

2. Run the following query:

   ```
   SELECT * FROM DBA_OLS_STATUS;

   NAME                 STATUS  DESCRIPTION
   -------------------- ------- -------------------------------------
   OLS_CONFIGURE_STATUS TRUE    Determines if OLS is configured
   OLS_ENABLE_STATUS    TRUE    Determines if OLS is enabled
   ```

4.1.3 Registering and Enabling Oracle Label Security from SQL*Plus

You can both register and enable Oracle Label Security from SQL*Plus.

1. Log into 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, query the `DBA_PDBS` data dictionary view. To check the current PDB, run the `show con_name` command.

2. Register and enable Oracle Label Security as follows.

   ```
   EXEC LBACSYS.CONFIGURE_OLS; -- This procedure registers Oracle Label
   Security.
   EXEC LBACSYS.OLS_ENFORCEMENT.ENABLE_OLS; -- This procedure enables it.
   ```

   After you invoke `LBACSYS.CONFIGURE_OLS`, all pending transactions are committed and cannot be rolled back, in the event that `LBACSYS.CONFIGURE_OLS` fails. `LBACSYS.CONFIGURE_OLS` can fail for reasons such as a pre-existing `LBAC_TRIGGER` schema.

3. Close and reopen the PDB.

For example:

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

4.1.4 Registering and Enabling Oracle Label Security Using DBCA

You can both register and enable Oracle Label Security using Database Configuration Assistant.

1. Start Database Configuration Assistant (DBCA).
   - **UNIX**: Run the following command:

     ```
     $ORACLE_HOME/bin/dbca
     ```

   - **Windows**: From the **Start** menu, click **All Programs**. Then click **Oracle - *ORACLE_HOME***, then **Configuration and Migration Tools**, and then **Database Configuration Assistant**.

   The Welcome screen appears.
2. Click **Next**.

   The Operations screen appears.
3. Select **Configure Database Options**. Click **Next**.

   The Database screen appears.
4. From the list, select the database where you need to configure and enable OLS. Click **Next**.

   The Database Content screen appears.
5. Select **Oracle Label Security**. Click **Next**.

   The Connection Mode screen appears.
6. Select either **Dedicated Server Mode** or **Shared Server Mode**. Click **Finish**.

   A dialog box is displayed informing you that the operation will require the database to be restarted.
7. Click **OK**.

   A confirmation dialog box is displayed.
8. Click **OK**.

   The DBCA progress screen is displayed.
9. After the operation is complete, you are prompted to perform another operation. Click **No** to exit DBCA.

4.1.5 Recompiling Oracle Label Security

If invalid objects appear in Oracle Label Security after you have upgraded, then you must recompile Oracle Label Security to remove these invalid objects.

Errors such as the following indicate that there are invalid objects: failures with the `catuppst.sql` script, `CATCL` errors, `ORA-06550: identifier must be declared` errors, or `ORA-942 (OLS Is Invalid)` errors.

1. Disable and then enable Oracle Label Security.

2. Recompile the objects within `LBACSYS`.

   ```
   EXEC DBMS_UTILITY.COMPILE_SCHEMA(SCHEMA => 'LBACSYS');
   ```

3. If dependent database objects require re-compilation as well, then recompile all the database objects, as follows:

   ```
   @$ORACLE_HOME/rdbms/admin/utlrp.sql
   ```

4. In the appropriate PDB, use SQL*Plus, to validate Oracle Label Security.

   ```
   EXECUTE SYS.VALIDATE_OLS;
   ```

5. If the `utlrp.sql` script compiles all the `OLS (LDAPSYS)` objects successfully, and there are no errors during compilation, then run the following command to list information about Oracle Label Security components that are loaded into the component registry:

   ```
   EXECUTE DBMS_REGISTRY.VALID('OLS');
   ```

Related Topics

- Disabling and Enabling Oracle Label Security
  You can disable and enable Oracle Label Security as necessary.

4.2 Security Guideline for Managing the LBACSYS User and the LBAC_DBA Role

As a good practice, for day-to-day use, grant the `LBAC_DBA` database role to trusted users who will administer Oracle Label Security.

If you plan to use Enterprise Manager Cloud Control to administer Oracle Label Security, then ensure that any users to whom you have granted the `LBAC_DBA` role also have the `SELECT ANY DICTIONARY` privilege.

Oracle strongly recommends that you maintain two accounts for users who have been granted the `LBAC_DBA` role. One account, the primary user account, will be used on a day-to-day basis and the other account will be used as a backup account in case the password of the primary account is lost and must be reset.

When users create Oracle Label Security policies, Oracle Label Security creates and grants to the user a special role for the policy named in the format of *`policy`*`_DBA`. Only a user who has this role can manage the policy. The `LBAC_DBA` role does not provide privileges to manage the policy.

4.3 Logging in to Cloud Control or SQL*Plus for Oracle Label Security

After you complete the Oracle Label Security registration and enablement process, you can begin using it.

- Logging in to Oracle Label Security from Enterprise Manager Cloud Control
  From Enterprise Manager Cloud Control, you use the Oracle Label Security pages to create and manage Oracle Label Security policies.
- Logging in to Oracle Label Security from SQL*Plus
  You can log in to Oracle Label Security from SQL*Plus if you have been granted the `LBAC_DBA` database role.

4.3.1 Logging in to Oracle Label Security from Enterprise Manager Cloud Control

From Enterprise Manager Cloud Control, you use the Oracle Label Security pages to create and manage Oracle Label Security policies.

1. Ensure that you have configured the Cloud Control target databases that you plan to use with Oracle Label Security.

   See the Oracle Enterprise Manager online help for more information about configuring target databases.
2. Point your browser to the Cloud Control login page.

   For example:

   ```
   https://myserver.example.com:7799/em
   ```
3. Log into Cloud Control as user `SYSMAN`.
4. In the Cloud Control home page, from the **Targets** menu, select **Databases**.
5. In the Databases page, select the link for the database to which you want to connect.

   The Database home page appears.
6. From the **Security** menu, select **Label Security**.

   The Database Login page appears.
7. Enter the following information:
   - **Username:** Enter the Oracle Label Security administrator user name.
   - **Password:** Enter the password.
   - **Role:** Select **NORMAL** from the list.
   - **Save As:** Select this check box if you want these credentials to be automatically filled in for you the next time that this page appears. The credentials are stored in Enterprise Manager in a secured manner. Access to these credentials depends on the user who is currently logged in.

4.3.2 Logging in to Oracle Label Security from SQL*Plus

You can log in to Oracle Label Security from SQL*Plus if you have been granted the `LBAC_DBA` database role.

- To use Oracle Label Security from SQL*Plus, connect as the Oracle Label Security administrator user.

For example:

```
sqlplus ols_admin@pdb_name
Enter password: password
```

To find the available PDBs, query the `DBA_PDBS` data dictionary view. To check the current PDB, run the `show con_name` command.

5

Creating an Oracle Label Security Policy

An Oracle Label Security policy is a named set of commands that implements Oracle Label Security.

- About Creating Oracle Label Security Policies
  When you create an Oracle Label Security policy, you must follow a set of general steps.
- Privileges for Managing Oracle Label Security Policies
  When a user creates an Oracle Label Security policy container, Oracle Label Security creates and grants this user a role that is specific to the policy, named in the format of `policy_DBA`.
- Step 1: Create the Label Security Policy Container
  The Oracle Label Security policy container is a storage place for the policy settings.
- Step 2: Create Data Labels for the Label Security Policy
  After you create a policy container, you are ready to create data labels for each database table row.
- Step 3: Authorize Users for the Label Security Policy
  Before users can have access to data that is protected by an Oracle Label Security policy, they must be authorized.
- Step 4: Grant Privileges to Users and Trusted Stored Program Units
  You can grant privileges to users, such as `READ` so that users can read data protected an Oracle Label Security policy protects.
- Step 5: Apply the Policy to a Database Table or Schema
  After you create grant authorizations and privileges to an Oracle Label Security policy, you can apply it to a database table or schema.
- Step 6: Add Policy Labels to Table Rows
  You must add policy labels to table rows.
- Step 7: Optionally, Configure Auditing
  For new Oracle Label Security policies, if you want to enable auditing, then you must create a unified audit policy.
- Using Oracle Label Security Policies and Oracle Flashback Data Archive
  Oracle Label Security policies do not automatically work with Oracle Flashback Data Archive (FDA).
- Using Enterprise Manager Cloud Control to Create an OLS Policy
  You can create Oracle Label Security policies in Oracle Enterprise Manager Cloud Control. However, you cannot create audit policies using Cloud Control; you must use SQL*Plus to create unified audit policies.

5.1 About Creating Oracle Label Security Policies

When you create an Oracle Label Security policy, you must follow a set of general steps.

1. Create a policy container that defines the policy name, the name of a column that Oracle Label Security will add to the tables to be protected, whether to hide this column, whether to enable the policy, and default enforcement options for the policy.

   See Step 1: Create the Label Security Policy Container for more information.
2. Define the following attributes for the label: level of sensitivity, and optionally, compartments and groups to further filter the label sensitivity. Once you have the attributes defined, create the label itself and then associate these attributes with the label.

   See Step 2: Create Data Labels for the Label Security Policy.
3. Authorize users for the policy.

   See Step 3: Authorize Users for the Label Security Policy for more information.
4. Grant privileges to these users or to trusted program units.

   See Step 4: Grant Privileges to Users and Trusted Stored Program Units for more information.
5. Apply the policy to a database table. Alternatively, you can apply the policy to an entire schema.

   See Step 5: Apply the Policy to a Database Table or Schema for more information.
6. Add the policy labels to the table rows. You must update the table that is being used for the policy.

   See Step 6: Add Policy Labels to Table Rows for more information.
7. Optionally, configure audit settings for users.

   See Step 7: Optionally, Configure Auditing for more information.

5.2 Privileges for Managing Oracle Label Security Policies

When a user creates an Oracle Label Security policy container, Oracle Label Security creates and grants this user a role that is specific to the policy, named in the format of *policy*_DBA.

For example, for a policy named emp_ols_pol, the role name is EMP_OLS_POL_DBA. This role becomes effective only after a new user session begins.

To enable the user to manage the policy, Oracle Label Security performs these checks in the following order:

1. Checks if the user is granted the *policy*_DBA role.
2. Checks if the *policy*_DBA is enabled for this user.
3. In the event of a nested call stack, checks if the top definer's rights procedure owner has been granted the *policy*_DBA role.
4. Checks if the *policy*_DBA role is granted to the LBACSYS user, which is the current user.

5.3 Step 1: Create the Label Security Policy Container

The Oracle Label Security policy container is a storage place for the policy settings.

- About the Label Security Policy Container
  The Oracle Label Security policy container stores metadata that describes how the policy behaves.
- Creating a Label Policy Container
  You can use the `SA_SYSDBA.CREATE_POLICY` procedure to create an Oracle Label Security policy container.

5.3.1 About the Label Security Policy Container

The Oracle Label Security policy container stores metadata that describes how the policy behaves.

This container defines the policy name, the name of a column that Oracle Label Security will add to the tables to be protected, whether to hide this column, and default enforcement options for the policy.

The column that you add to the tables that you want to protect will include data labels (which you create later on) that are assigned to specific rows in a the table, based on values in a specific column.

You can create the policy container in Oracle Enterprise Manager Cloud Control, or use the `SA_SYSDBA.CREATE_POLICY` procedure.

5.3.2 Creating a Label Policy Container

You can use the `SA_SYSDBA.CREATE_POLICY` procedure to create an Oracle Label Security policy container.

- To create the policy, run `SA_SYSDBA.CREATE_POLICY`, specifying the policy name, column name, and default options.

For example:

```
BEGIN
 SA_SYSDBA.CREATE_POLICY (
  policy_name      => 'emp_ols_pol',
  column_name      => 'ols_col',
  default_options  => 'read_control, update_control');
END;
/
```

Related Topics

- SA_SYSDBA.CREATE_POLICY
  The `SA_SYSDBA.CREATE_POLICY` procedure creates a new Oracle Label Security policy, defines a policy-specific column name, and specifies default policy options.

5.4 Step 2: Create Data Labels for the Label Security Policy

After you create a policy container, you are ready to create data labels for each database table row.

- About Data Labels
  A data label indicates the sensitivity of a database table row.
- About Policy Level Sensitivity Components
  A *level* is a ranking that denotes the sensitivity of the information it labels.

- Creating a Policy Level Component
  The `SA_COMPONENTS.CREATE_LEVEL` procedure creates a policy level component.
- About Policy Compartment Components
  Compartments identify areas that describe the sensitivity of the labeled data, providing a finer level of granularity within a level.
- Creating a Policy Compartment Component
  The `SA_COMPONENTS.CREATE_COMPARTMENT` procedure creates an Oracle Label Security compartment.
- About Policy Group Components
  Groups identify organizations owning or accessing the data, such as `EASTERN_REGION`, `WESTERN_REGION`, `WR_SALES`.
- Creating a Policy Data Label Group
  The `SA_COMPONENTS.CREATE_GROUP` procedure creates a data label group.
- About Associating the Policy Components with a Named Data Label
  After defining the data label components, you can create a data label itself by associating it with an existing level.
- Associating the Policy Components with a Named Data Label
  The `SA_LABEL_ADMIN.CREATE_LABEL` procedure creates a data label.

5.4.1 About Data Labels

A data label indicates the sensitivity of a database table row.

Each label is a single attribute with multiple components that control the types of filtering to be used for user access.

Table 5-1 describes the different components of a data label.

Table 5-1 Sensitivity Data Label Components

Component	Description	Examples
Level	A single specification of the sensitivity of labeled data within the ordered ranks established	`CONFIDENTIAL` (1), `SENSITIVE` (2), `HIGHLY_SENSITIVE` (3)
Compartments	Zero or more categories associated with the labeled data	`FINANCIAL`, `STRATEGIC`, `NUCLEAR`
Groups	Zero or more identifiers for organizations owning or accessing the data	`EASTERN_REGION`, `WESTERN_REGION`

All data labels must contain a level component, but the compartment and group components are optional. Compartments and groups are a way of fine tuning access that users will have to the data. Valid characters for specifying all label components include alphanumeric characters, underscores, and spaces. (Leading and trailing spaces are ignored.) You must define the label components before you can create the data label itself.

You can use Cloud Control to create the label and its components for an existing policy. Alternatively, you can use the `SA_COMPONENTS` PL/SQL package to create the components, and the `SA_LABEL_ADMIN` package to create the data label.

Related Topics

- SA_COMPONENTS Label Components PL/SQL Package
  The `SA_COMPONENTS` PL/SQL package manages the component definitions of an Oracle Label Security label.

5.4.2 About Policy Level Sensitivity Components

A *level* is a ranking that denotes the sensitivity of the information it labels.

The more sensitive the information, the higher its level. The less sensitive the information, the lower its level.

Every label must include one level. Oracle Label Security permits up to 10,000 levels in a policy. For each level, you must define a numeric form, a long character form, and the required short character form.

Table 5-2 shows examples of levels.

Table 5-2 Policy Level Example

Numeric Form	Long Form	Short Form
`40`	`HIGHLY_SENSITIVE`	`HS`
`30`	`SENSITIVE`	`S`
`20`	`CONFIDENTIAL`	`C`
`10`	`PUBLIC`	`P`

Table 5-2 explains the numeric form, long form, and short form for levels.

Table 5-3 Forms of Specifying Levels

Form	Explanation
Numeric form, also called "tag"	The numeric form of the level can range from 0 to 9999. Sensitivity is ranked by this numeric value, so you must assign higher numbers to levels that are more sensitive, and lower numbers to levels that are less sensitive. In Table 5-2, 40 (`HIGHLY_SENSITIVE`) is a higher level than 30, 20, and 10. Administrators should avoid using sequential numbers for the numeric form of levels. A good strategy is to use even increments (such as 50 or 100) between levels. You can then insert additional levels between two preexisting levels, at a later date.
Long form	The long form of the level name can contain up to 80 characters.
Short form	The short form can contain up to 30 characters.

Although you define both long and short names for the level (and for each of the other label components), only the short form of the name is displayed upon retrieval. When users manipulate the labels, they use only the short form of the component names.

Examples of levels can be names such as `TOP_SECRET`, `SECRET`, `CONFIDENTIAL`, and `UNCLASSIFIED` or `TRADE_SECRET`, `PROPRIETARY`, `COMPANY_CONFIDENTIAL`, `PUBLIC_DOMAIN`.

If you use only levels, a level 40 user (in this example) can access or alter any data row whose level is 40 or less.

5.4.3 Creating a Policy Level Component

The `SA_COMPONENTS.CREATE_LEVEL` procedure creates a policy level component.

- To create the policy level component, run `SA_COMPONENTS.CREATE_LEVEL`, specifying the policy name and details about the component.

  For example:

  ```
  BEGIN
   SA_COMPONENTS.CREATE_LEVEL (
     policy_name    => 'emp_ols_pol',
     level_num      => 40,
     short_name     => 'HS',
     long_name      => 'HIGHLY_SENSITIVE');
  END;
  /
  ```

Related Topics

- SA_COMPONENTS.CREATE_LEVEL
  The `SA_COMPONENTS.CREATE_LEVEL` procedure creates a level and specify its short name and long name.

5.4.4 About Policy Compartment Components

Compartments identify areas that describe the sensitivity of the labeled data, providing a finer level of granularity within a level.

Compartments associate the data with one or more security areas. All data related to a particular project can be labeled with the same compartment.

Table 5-4 shows an example set of compartments.

Table 5-4 Policy Compartment Example

Numeric Form	Long Form	Short Form
85	FINANCIAL	FINCL
65	CHEMICAL	CHEM
45	OPERATIONAL	OP

Table 5-5 shows different ways to specify compartments.

Table 5-5 Forms of Specifying Compartments

Form	Explanation

Numeric form	The numeric form can range from 0 to 9999. It is unrelated to the numbers used for the levels. The numeric form of the compartment does not indicate greater or less sensitivity. Instead, it controls the display order of the short form compartment name in the label character string. For example, assume a label is created that has all three compartments listed in Table 5-4, and a level of `SENSITIVE`. When this label is displayed in string format, it looks like this:
	`S:OP,CHEM,FINCL`
	meaning `SENSITIVE`: `OPERATIONAL`, `CHEMICAL`, `FINANCIAL`
	The display order follows the order of the numbers assigned to the compartments: 45 is lower than 65, and 65 is lower than 85. By contrast, if the number assigned to the `FINCL` compartment were 5, the character string format of the label would look like this:
	`S:FINCL,OP,CHEM`

Long form	The long form of the compartment name scan have up to 80 characters.
Short form	The short form can contain up to 30 characters.

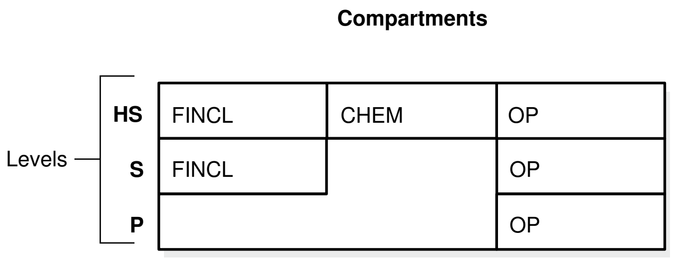
Compartments are optional. You can include up to 10,000 compartments for a label.

Not all labels must have compartments. For example, you can specify `HIGHLY_SENSITIVE` and `CONFIDENTIAL` levels with no compartments, and a `SENSITIVE` level that does contain compartments.

When you analyze the sensitivity of data, you may find that some compartments are only useful at specific levels.

The following figure shows how compartments can be used to categorize data.

Figure 5-1 Compartments in a Label



Here, compartments `FINCL`, `CHEM`, and `OP` are used with the level `HIGHLY_SENSITIVE` (`HS`). The label `HIGHLY_SENSITIVE:FINCL, CHEM` indicates a level of 40 with the two named compartments. Compartment `FINCL` is not more sensitive than `CHEM`, nor is `CHEM` more sensitive than `FINCL`. Note also that some data in the protected table may not belong to any compartment.

If you specify compartments, then a user whose level would normally permit access to a row's data will nevertheless be prevented from such access unless the user's label also contains all the compartments appearing in that row's label. For example, user `hpreston`, who is granted access to the `HS` level, could be granted access only to `FINCL` and `CHEM` but not to `OP`.

5.4.5 Creating a Policy Compartment Component

The `SA_COMPONENTS.CREATE_COMPARTMENT` procedure creates an Oracle Label Security compartment.

- To create the compartment, run the `SA_COMPONENTS.CREATE_COMPARTMENT` procedure to create a compartment, specifying the policy name and details about the compartment.

```
BEGIN
  SA_COMPONENTS.CREATE_COMPARTMENT (
   policy_name     => 'emp_ols_pol',
   comp_num        => '85',
   short_name      => 'FINCL',
   long_name       => 'FINANCIAL');
END;
/
```

Related Topics

- SA_COMPONENTS.CREATE_COMPARTMENT
  The `SA_COMPONENTS.CREATE_COMPARTMENT` procedure creates a compartment and specify its short name and long name.

5.4.6 About Policy Group Components

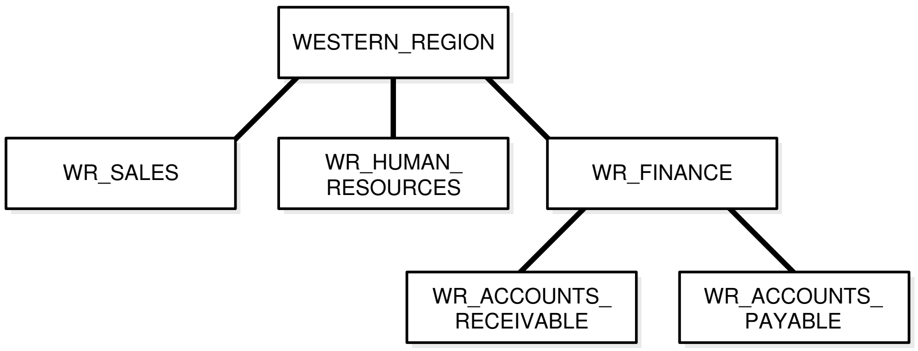
Groups identify organizations owning or accessing the data, such as `EASTERN_REGION`, `WESTERN_REGION`, `WR_SALES`.

All data pertaining to a certain department can have that department's group in the label. Groups are useful for the controlled dissemination of data and for timely reaction to organizational change. When a company reorganizes, data access can change right along with the reorganization.

Groups are hierarchical. You can label data based upon your organizational infrastructure. A group can thus be associated with a parent group.

Figure 5-2 shows how you can define a set of groups corresponding to the following organizational hierarchy.

Figure 5-2 Group Example



The `WESTERN_REGION` group includes three subgroups: `WR_SALES`, `WR_HUMAN_RESOURCES`, and `WR_FINANCE`. The `WR_FINANCE` subgroup is subdivided into `WR_ACCOUNTS_RECEIVABLE` and `WR_ACCOUNTS_PAYABLE`.

Table 5-6 shows how the organizational structure in this example can be expressed in the form of Oracle Label Security groups. The numeric form assigned to the groups affects display order only. You specify the hierarchy (that is, the parent and child relationships) separately. The first group listed, `WESTERN_REGION`, is the parent group of the remaining groups in the table.

Table 5-6 Group Example

Numeric Form	Long Form	Short Form	Parent Group
`1000`	`WESTERN_REGION`	`WR`	
`1100`	`WR_SALES`	`WR_SAL`	`WR`
`1200`	`WR_HUMAN_RESOURCES`	`WR_HR`	`WR`
`1300`	`WR_FINANCE`	`WR_FIN`	`WR`
`1310`	`WR_ACCOUNTS_PAYABLE`	`WR_AP`	`WR_FIN`
`1320`	`WR_ACCOUNTS_RECEIVABLE`	`WR_AR`	`WR_FIN`

Table 5-7 shows the forms that you must use when you specify groups.

Table 5-7 Forms of Specifying Groups

Form	Explanation
Numeric form	The numeric form of the group can range from 0 to 9999, and it must be unique for each policy. The numeric form does not indicate any kind of ranking. It does not indicate a parent-child relationship, or greater or less sensitivity. It only controls the display order of the short form group name in the label character string. For example, assume that a label is created that has the level `SENSITIVE`, the compartment `CHEMICAL`, and the groups `WESTERN_REGION` and `WR_HUMAN_RESOURCES` as listed in Table 5-6. When displayed in string format, the label looks like this: `S:CHEM:WR,WR_HR` `WR` is displayed before `WR_HR` because 1000 comes before 1200.
Long form	The long form of the group name can contain up to 80 characters.
Short form	The short form can contain up to 30 characters.

Groups are optional. A label can contain up to 10,000 groups.

All labels do not need to have groups. When you analyze the sensitivity of data, you may find that some groups are only used at specific levels. For example, you can specify `HIGHLY_SENSITIVE` and `CONFIDENTIAL` labels with no groups, and a `SENSITIVE` label that does contain groups.

5.4.7 Creating a Policy Data Label Group

The `SA_COMPONENTS.CREATE_GROUP` procedure creates a data label group.

- Run the `SA_COMPONENTS.CREATE_GROUP` procedure for each data label group that you need.

  In the following example, the first `CREATE_GROUP` procedure creates the parent group, `WR`, and the second procedure associates a second group with the `WR` group by using the `parent_name` parameter.

  ```
  BEGIN
   SA_COMPONENTS.CREATE_GROUP (
    policy_name     => 'emp_ols_pol',
    group_num       => 1000,
    short_name      => 'WR',
    long_name       => 'WESTERN_REGION');
  END;
  /
  BEGIN
   SA_COMPONENTS.CREATE_GROUP (
    policy_name     => 'emp_ols_pol',
    group_num       => 1100,
    short_name      => 'WR_SAL',
    long_name       => 'WR_SALES',
    parent_name     => 'WR');
  END;
  /
  ```

Related Topics

- SA_COMPONENTS.CREATE_GROUP
  The `SA_COMPONENTS.CREATE_GROUP` procedure creates a group and specify its short name and long name, and optionally a parent group.

5.4.8 About Associating the Policy Components with a Named Data Label

After defining the data label components, you can create a data label itself by associating it with an existing level.

Optionally, you can include compartments and groups in this association.

You can use Oracle Enterprise Manager Cloud Control or the `SA_LABEL_ADMIN.CREATE_LABEL` procedure. Character string representations of labels use the following syntax:

```
level:compartment1,...,compartmentn:group1,...,groupn
```

The text string that specifies the label can have a maximum of 4,000 characters, including alphanumeric characters, spaces, and underscores. The label names are case-insensitive. You can enter them in uppercase, lowercase, or mixed case, but the string is stored in the data dictionary and displayed in uppercase. Separate each set of components with a colon. You do not need to enter trailing delimiters in this syntax.

For example, you can create valid labels such as these:

```
SENSITIVE:FINANCIAL,CHEMICAL:EASTERN_REGION,WESTERN_REGION
CONFIDENTIAL:FINANCIAL:VP_GRP
SENSITIVE
HIGHLY_SENSITIVE:FINANCIAL
SENSITIVE::WESTERN_REGION
```

5.4.9 Associating the Policy Components with a Named Data Label

The `SA_LABEL_ADMIN.CREATE_LABEL` procedure creates a data label.

- Run `SA_LABEL_ADMIN.CREATE_LABEL`, specifying the policy name and details about the policy components.

  For example:

  ```
  BEGIN
   SA_LABEL_ADMIN.CREATE_LABEL  (
    policy_name     => 'emp_ols_pol',
    label_tag       => '1310',
    label_value     => 'S:FINCL,CHEM:ER,WR',
    data_label      => TRUE);
  END;
  /
  ```

  In this example, the `label_value` setting is in the short form, which translates to the following long form:

  ```
  SENSITIVE:FINANCIAL,CHEMICAL:EASTERN_REGION,WESTERN_REGION
  ```

  When you create a data label, two additional actions occur:

  – The label is automatically designated as a valid data label. This functionality limits the labels that can be assigned to data.

– A numeric label tag is associated with the text string representing the label. It is this label tag, rather than the text string, that is stored in the policy label column of the protected table.

> **Note:**
>
> For Oracle Label Security installations that do not use Oracle Internet Directory, dynamic creation of valid data labels uses the `TO_DATA_LABEL` function. Its usage should be tightly controlled.

Related Topics

- Inserting Labels Using TO_DATA_LABEL
  The `TO_DATA_LABEL` function can generate new labels dynamically.
- SA_LABEL_ADMIN.CREATE_LABEL
  The `SA_LABEL_ADMIN.CREATE_LABEL` procedure creates data labels.

5.5 Step 3: Authorize Users for the Label Security Policy

Before users can have access to data that is protected by an Oracle Label Security policy, they must be authorized.

- About Authorizing Users for Label Security Policies
  When you authorize users, you enable them to have access to row data based on how the data labels are defined.
- About Authorizing Levels
  You can explicitly set default, minimum, and maximum authorization levels.
- Authorizing a Level
  The `SA_USER_ADMIN.SET_LEVELS` procedure authorizes users for policy levels components.
- About Authorizing Compartments
  After you authorize the user for a specific level, optionally you can specify compartments to be added to a session label.
- Authorizing a Compartment
  The `SA_USER_ADMIN.SET_COMPARTMENTS` procedure authorizes a user for the compartments component.
- About Authorizing Groups
  You can specify the list of groups that a user can place in session label.
- Authorizing a Group
  The `SA_USER_ADMIN.SET_GROUPS` procedure authorizes users for a policy group.

5.5.1 About Authorizing Users for Label Security Policies

When you authorize users, you enable them to have access to row data based on how the data labels are defined.

First, you set the user's authorization for each level, compartment, and group that is associated with the label. You can find the currently granted privileges for a user by querying the `DBA_SA_USER_PRIVS` data dictionary view.

5.5.2 About Authorizing Levels

You can explicitly set default, minimum, and maximum authorization levels.

Table 5-8 Authorized Levels Set by the Administrator

Authorization	Meaning
User Max Level	The maximum ranking of sensitivity that a user can access during read and write operations
User Min Level	The minimum ranking of sensitivity that a user can access during write operations. The User Max Level must be equal to or greater than the User Min Level.
User Default Level	The level that is assumed by default when connecting to Oracle Database
User Default Row Level	The level that is used by default when inserting data into Oracle Database

For example, you might set the following level authorizations for user `hpreston`:

Type	Short Name	Long Name	Description
Maximum	`HS`	`HIGHLY_SENSITIVE`	User's highest level
Minimum	`P`	`PUBLIC`	User's lowest level
Default	`C`	`CONFIDENTIAL`	User's default level
Row	`C`	`CONFIDENTIAL`	Row level on `INSERT`

5.5.3 Authorizing a Level

The `SA_USER_ADMIN.SET_LEVELS` procedure authorizes users for policy levels components.

Note that when you specify the levels, you must always use the short names, not the long names.

- Run `SA_USER_ADMIN.SET_LEVELS` to authorize the level, specifying the policy name, user name, and levels.

  For example:

```
BEGIN
 SA_USER_ADMIN.SET_LEVELS (
  policy_name   => 'ols_admin_pol',
  user_name     => 'hpreston',
  max_level     => 'HS',
  min_level     => 'P',
  def_level     => 'C',
  row_level     => 'C');
END;
/
```

Related Topics

- SA_USER_ADMIN.SET_LEVELS
  The SA_USER_ADMIN.SET_LEVELS procedure assigns a user minimum and maximum levels and identifies default values for the user's session label and row label.

5.5.4 About Authorizing Compartments

After you authorize the user for a specific level, optionally you can specify compartments to be added to a session label.

Write access must be explicitly given for each compartment. A user cannot directly insert, update, or delete a row that contains a compartment that the user does not have authorization to write.

For example, you could set the following compartment authorizations for user hpreston:

Short Name	Long Name	WRITE	DEFAULT	ROW
CHEM	CHEMICAL	YES	YES	NO
FINCL	FINANCIAL	YES	YES	NO
OP	OPERATIONAL	YES	YES	YES

5.5.5 Authorizing a Compartment

The SA_USER_ADMIN.SET_COMPARTMENTS procedure authorizes a user for the compartments component.

When you specify the compartments, you must use their short names, not their long names.

- Run SA_USER_ADMIN.SET_COMPARTMENTS to authorize a user for a compartment, specifying the policy name, user name, and compartment details.

  For example:

  ```
  BEGIN
   SA_USER_ADMIN.SET_COMPARTMENTS (
    policy_name   => 'ols_admin_pol',
    user_name     => 'hpreston',
    read_comps    => 'FINCL',
    write_comps   => 'FINCL',
    def_comps     => 'FINCL',
    row_comps     => 'FINCL');
  END;
  /
  ```

After you have run this procedure, you can authorize the user for additional compartments by running the SA_USER_ADMIN.ADD_COMPARTMENTS procedure.

Related Topics

- SA_USER_ADMIN.SET_COMPARTMENTS
  The SA_USER_ADMIN.SET_COMPARTMENTS procedure assigns compartments to a user and identifies default values for the user's session label and row label.

- SA_USER_ADMIN.ADD_COMPARTMENTS
  The `SA_USER_ADMIN.ADD_COMPARTMENTS` procedure adds (assigns) compartments to a user's authorizations, indicating if the compartments are authorized for write and read privileges.

5.5.6 About Authorizing Groups

You can specify the list of groups that a user can place in session label.

Write access must be explicitly given for each group listed.

For example, you could set the following group authorizations:

Short Name	Long Name	WRITE	DEFAULT	ROW	Parent
`WR_HR`	`WR_HUMAN_RESOURCES`	YES	YES	YES	`WR`
`WR_AP`	`WR_ACCOUNTS_PAYABLE`	YES	YES	NO	`WR_FIN`
`WR_AR`	`WR_ACCOUNTS_RECEIVABLE`	YES	YES	NO	`WR_FIN`

5.5.7 Authorizing a Group

The `SA_USER_ADMIN.SET_GROUPS` procedure authorizes users for a policy group.

- Run `SA_USER_ADMIN.SET_GROUPS` to authorize the user, specifying the policy name, user name, and authorizations that you want. When you specify the groups, you must use the short name, not the long name.

  For example:

```
BEGIN
 SA_USER_ADMIN.SET_GROUPS (
  policy_name   => 'ols_admin_pol',
  user_name     => 'hpreston',
  read_groups   => 'WR_AP',
  write_groups  => 'WR_AP',
  def_groups    => 'WR_AP',
  row_groups    => 'WR_AP');
END;
/
```

After you have run this procedure, you can authorize the user for additional groups by running the `SA_USER_ADMIN.ADD_GROUPS` procedure.

Related Topics

- SA_USER_ADMIN.SET_GROUPS
  The `SA_USER_ADMIN.SET_GROUPS` procedure assigns groups to a user and identifies default values for the user's session label and row label.

5.6 Step 4: Grant Privileges to Users and Trusted Stored Program Units

You can grant privileges to users, such as `READ` so that users can read data protected an Oracle Label Security policy protects.

- About Granting Privileges to Users and Trusted Program Units for the Policy
  After you have authorized users for policy levels, compartments, and groups, you are ready to grant the user privileges.
- Granting Privileges to a User
  The `SA_USER_ADMIN.SET_USER_PRIVS` procedure grants users privileges.
- Granting Privileges to a Trusted Program Unit
  The `SA_USER_ADMIN.SET_PROG_PRIVS` procedure grants privileges to trusted program units.

5.6.1 About Granting Privileges to Users and Trusted Program Units for the Policy

After you have authorized users for policy levels, compartments, and groups, you are ready to grant the user privileges.

Trusted program units are functions, procedures, or packages that are granted Oracle Label Security privileges. You create a trusted stored program unit in the same way that you create a standard procedure, function, or package, that is by using the `CREATE PROCEDURE`, `CREATE FUNCTION`, or `CREATE PACKAGE` and `CREATE PACKAGE BODY` statements. The program unit becomes trusted when you grant Oracle Label Security privileges to it.

Table 5-9 summarizes the privileges that can be granted to users or trusted stored program units.

Table 5-9 Oracle Label Security Privileges

Security Privilege	Explanation
`READ`	Allows read access to all data protected by the policy
`FULL`	Allows full read and write access to all data protected by the policy
`COMPACCESS`	Allows a session access to data authorized by the row's compartments, independent of the row's groups
`PROFILE_ACCESS`	Allows a session to change its labels and privileges to those of a different user
`WRITEUP`	Allows users to set or raise only the level, within a row label, up to the maximum level authorized for the user. (Active only if `LABEL_UPDATE` is active.)
`WRITEDOWN`	Allows users to set or lower the level, within a row label, to any level equal to or greater than the minimum level authorized for the user. (Active only if `LABEL_UPDATE` is active.)
`WRITEACROSS`	Allows a user to set or change groups and compartments of a row label, but does not allow changes to the level. (Active only if `LABEL_UPDATE` is active.)

5.6.2 Granting Privileges to a User

The `SA_USER_ADMIN.SET_USER_PRIVS` procedure grants users privileges.

- Run `SA_USER_ADMIN.SET_USER_PRIVS`, specifying the policy name, user name, and privileges that you want to grant.

  For example:

  ```
  BEGIN
   SA_USER_ADMIN.SET_USER_PRIVS(
    policy_name   => 'ols_admin_pol',
    user_name     => 'hpreston',
    privileges    => 'WRITEDOWN');
  END;
  /
  ```

Related Topics

- SA_USER_ADMIN.SET_USER_PRIVS
  The `SA_USER_ADMIN.SET_USER_PRIVS` procedure sets policy-specific privileges for users.

5.6.3 Granting Privileges to a Trusted Program Unit

The `SA_USER_ADMIN.SET_PROG_PRIVS` procedure grants privileges to trusted program units.

- Run `SA_USER_ADMIN.SET_PROG_PRIVS` to grant the privileges, specifying the policy name, schema name, program unit name, and privileges that you want to grant.

  For example:

  ```
  BEGIN
   SA_USER_ADMIN.SET_PROG_PRIVS (
    policy_name         => 'oe_ols_pol',
    schema_name         => 'oe',
    program_unit_name   => 'check_order_updates',
    privileges          => 'READ');
  END;
  /
  ```

Related Topics

- SA_USER_ADMIN.SET_PROG_PRIVS
  The `SA_USER_ADMIN.SET_PROG_PRIVS` procedure sets policy-specific privileges for program units.

5.7 Step 5: Apply the Policy to a Database Table or Schema

After you create grant authorizations and privileges to an Oracle Label Security policy, you can apply it to a database table or schema.

- About Applying the Policy to a Database Table or Schema
  When you apply a policy to a table, the policy is automatically enabled.
- Applying a Policy to a Schema
  The `SA_POLICY_ADMIN.APPLY_TABLE_POLICY` applies a policy to a table within a schema and the `SA_POLICY_ADMIN.APPLY_SCHEMA_POLICY` procedure applies a policy to an entire schema.

5.7.1 About Applying the Policy to a Database Table or Schema

When you apply a policy to a table, the policy is automatically enabled.

To disable a policy is to turn off its protections, although it is still applied to the table. To enable a policy is to turn on and enforce its protections for a particular table or schema.

To remove a policy is to take it entirely away from the table or schema. Note, however, that the policy label column and the labels remain in the table unless you explicitly drop them.

You can alter the default policy enforcement options for future tables that may be created in a schema. This does not, however, affect policy enforcement options on existing tables in the schema.

To change the enforcement options on an existing table, you must first *remove* the policy from the table, make the desired changes, and then reapply the policy to the table.

Be aware that you cannot enforce Oracle Label Security policies on external tables.

After you have created the policy components and configured user authorizations, privileges, and auditing for them, you can apply the policy to a database table or to an entire schema.

When you apply the policy to a database table, in addition to the policy name and target schema table, you must specify the following information:

- `table_options`: A comma-delimited list of policy enforcement options to be used for the table. If `NULL`, then the policy's default options are used.
- `label_function`: A string calling a function to return a label value to use as the default. For example, `my_label(:new.dept,:new.status)` computes the label based on the new values of the `DEPT` and `STATUS` columns in the row.
- `predicate`: An additional predicate to combine (using `AND` or `OR`) with the label-based predicate for `READ_CONTROL`

Note the following aspects of using Oracle Label Security policies with schemas:

- If you apply a policy to an empty schema, then every time you create a table within that schema, the policy is applied. Once the policy is applied to the schema, the default options you choose are applied to every table added.
- If you remove the policy from a table so that it is unprotected, and then run `SA_POLICY_ADMIN.ENABLE_SCHEMA_POLICY`, then the table will remain unprotected. If you wish to protect the table once again, then you must apply the policy to the table, or re-apply the policy to the schema.

If you apply a policy to a schema that already contains tables protected by the policy, then all future tables will have the new options that were specified when you applied the policy. The existing tables will retain the options they already had.

5.7.2 Applying a Policy to a Schema

The `SA_POLICY_ADMIN.APPLY_TABLE_POLICY` applies a policy to a table within a schema and the `SA_POLICY_ADMIN.APPLY_SCHEMA_POLICY` procedure applies a policy to an entire schema.

- Run `SA_POLICY_ADMIN.APPLY_TABLE_POLICY` to apply the policy to a table, specifying the policy name, schema name, and necessary options.

  The following example shows how to use the `SA_POLICY_ADMIN.APPLY_TABLE_POLICY` procedure to apply the `ols_admin_pol` policy to the `HR.EMPLOYEES` table.

  ```
  BEGIN
   SA_USER_ADMIN.APPLY_TABLE_POLICY (
    policy_name    => 'ols_admin_pol',
    schema_name    => 'hr',
    table_name     => 'employees',
    table_options  => 'READ_CONTROL,WRITE_CONTROL,CHECK_CONTROL',
    label_function => 'hr.gen_emp_label(:new.department_id,:new.salary',
    predicate      => NULL);
  END;
  /
  ```

  This example shows how to use the `SA_POLICY_ADMIN.APPLY_SCHEMA_POLICY` procedure to apply a policy to an entire schema.

  ```
  BEGIN
   SA_USER_ADMIN.APPLY_SCHEMA_POLICY (
    policy_name      => 'ols_admin_pol',
    schema_name      => 'hr',
    default_options  => NULL);
  END;
  /
  ```

Related Topics

- SA_POLICY_ADMIN.APPLY_TABLE_POLICY
  The `SA_POLICY_ADMIN.APPLY_TABLE_POLICY` procedure adds the specified policy to a table.
- SA_POLICY_ADMIN.APPLY_SCHEMA_POLICY
  The `SA_POLICY_ADMIN.APPLY_SCHEMA_POLICY` procedure applies a policy to all of the tables in a schema and enables the policy for these tables.

5.8 Step 6: Add Policy Labels to Table Rows

You must add policy labels to table rows.

- About Adding Policy Labels to Table Rows
  After you have applied a policy to a table, you must add data labels to the rows in the table.
- Adding a Policy Label to a Table Row
  You must update the table to which you are adding a policy label.

5.8.1 About Adding Policy Labels to Table Rows

After you have applied a policy to a table, you must add data labels to the rows in the table.

These labels are stored in the policy label column that you created earlier in the table. The user updating the table must have the `FULL` security privilege for the policy. This user is normally the owner of the table.

5.8.2 Adding a Policy Label to a Table Row

You must update the table to which you are adding a policy label.

1. To add data labels to a table, in SQL*Plus, enter an `UPDATE` statement using the following syntax:

   ```
   UPDATE table_name
   SET ols_column = CHAR_TO_LABEL('ols_policy','data_label')
   WHERE UPPER(table_column) IN (column_data);
   ```

   For example, suppose `LABCSYS` has created a policy called `ACCESS_LOCATIONS` and wants to add the label `SENS` to the cities Beijing, Tokyo, and Singapore in the `HR.LOCATIONS` table. The policy label column is called `ROW_LABEL`. The `UPDATE` statement is as follows:

   ```
   UPDATE LOCATIONS
   SET ROW_LABEL = CHAR_TO_LABEL('ACCESS_LOCATIONS','SENS')
   WHERE UPPER(city) IN ('BEIJING', 'TOKYO', 'SINGAPORE');
   ```

2. Run the following `SELECT` statement to ensure that the policy was added to the table:

   ```
   SELECT LABEL_TO_CHAR (ROW_LABEL) FROM LOCATIONS;
   ```

5.9 Step 7: Optionally, Configure Auditing

For new Oracle Label Security policies, if you want to enable auditing, then you must create a unified audit policy.

1. Log in to the database instance as a user who has the `LBAC_DBA` database role and the `AUDIT_ADMIN` system privilege.
2. Create the unified audit policy for the Oracle Label Security policy.

   For example:

   ```
   CREATE AUDIT POLICY ols_admin_pol
    ACTIONS UPDATE ON HR.EMPLOYEES, DELETE ON HR.EMPLOYEES
    ACTIONS COMPONENT=OLS DROP POLICY, DISABLE POLICY;
   ```

3. Enable this unified audit policy.

   For example:

   ```
   AUDIT POLICY ols_admin_pol;
   ```

The audit record will be written to the unified audit trail, viewable with the `UNIFIED_AUDIT_TRAIL` data dictionary view. For example:

```
SELECT OLS_PRIVILEGES_USED FROM UNIFIED_AUDIT_TRAIL
WHERE OLS_POLICY_NAME = 'OLS_ADMIN_POL' AND DBUSERNAME = 'PSMITH';
```

Related Topics

- *Oracle Database Security Guide*

5.10 Using Oracle Label Security Policies and Oracle Flashback Data Archive

Oracle Label Security policies do not automatically work with Oracle Flashback Data Archive (FDA).

After you create an Oracle Label Security policy for a table, consider creating an equivalent policy for the Flashback Data Archive (FDA) history table.

1. Find the object ID for the table to which you attached the Oracle Label Security policy.

   For example, suppose you attached an Oracle Label Security policy to the `HR.LOCATIONS` table:

   ```
   SELECT OBJECT_ID FROM DBA_OBJECTS WHERE OBJECT_NAME='LOCATIONS' AND
   OWNER='HR';
   ```

2. Use this object ID to create a policy container for the history table that is associated with the Oracle Label Security policy's associated table.

3.

5.11 Using Enterprise Manager Cloud Control to Create an OLS Policy

You can create Oracle Label Security policies in Oracle Enterprise Manager Cloud Control. However, you cannot create audit policies using Cloud Control; you must use SQL*Plus to create unified audit policies.

- Creating the Label Security Policy Container Using Cloud Control
  You can create the Oracle Label Security policy container in Cloud Control.
- Creating Policy Components Using Cloud Control
  After you create a container for the policy and set enforcement options for it, you can create components for the policy.
- Creating Data Labels for the Policy Using Cloud Control
  You can create data labels for an Oracle Label Security policy in Cloud Control.
- Authorizing and Granting Privileges for a Policy Using Cloud Control
  You can authorize and grant privileges to users for a policy during the user creation process.
- Granting Privileges to Trusted Program Units Using Cloud Control
  You can grant privileges to trusted program units in Cloud Control.
- Applying a Policy to a Database Table with Cloud Control
  You can apply an Oracle Label Security policy to a database table in Cloud Control.
- Applying Policy Labels to Table Rows Using Cloud Control
  You can apply Oracle Label Security policy labels to table rows in Cloud Control.

5.11.1 Creating the Label Security Policy Container Using Cloud Control

You can create the Oracle Label Security policy container in Cloud Control.

1. Log in to Cloud Control as the `SYSTEM` user.
2. To navigate to your database, select **Databases** from the **Targets** menu.
3. Click the database name in the list that appears.

   The database page appears.
4. Under the **Administration** menu, select **Security, Oracle Label Security.** The Label Security Policies page appears.

   You may be required to log in to the database with the credentials of an Oracle Label Security administrator.
5. Click **Create** to start creating a new label security policy. The Create Label Security Policy page appears.
6. Define the policy's name, label column, and the default policy enforcement options.
   - **Name**: Enter a name for the policy, for example, `access_locations`.
   - **Label Column**: (Optional) Enter a name for the label column, for example, `OLS_COLUMN`. If you create an OLS policy without specifying the column name, the column name is auto-generated as *`Pol_name`*`_COL`.Later on, when you apply the policy to a table, the label column is added to that table. By default, the data type of the policy label column is `NUMBER(10)`. You can also specify an existing table column of the `NUMBER(10)` data type as the label column.
   - **Hide Label Column**: Select to hide the column. When you first create the policy, you may want to disable **Hide Label Column** during the development phase of the policy. When the policy is satisfactory and ready for use by users, hide the column so that it is transparent to applications.
   - **Enabled**: Toggle to enable or disable the policy.
   - **Default Policy Enforcement Options**: The default policy enforcement options are used when the policy is applied. Ensure that these meet the needs of the application to which you are applying the policy.

     Select from the following options:
     - **Apply No Policy Enforcements (NO_CONTROL)**
     - **Apply Policy Enforcements**

       For all queries (READ_CONTROL)

       For Insert operations (INSERT_CONTROL)

       For Update Operations (UPDATE_CONTROL)

       Use session's default label for label column update (LABEL_DEFAULT)

       Operations that update the label column (LABEL_UPDATE)

Update and Insert operations so that they are read accessible (CHECK_CONTROL)

7. Click **OK**.

   The new policy appears in the Oracle Label Security Policies page. After you create this policy container, Oracle Label Security creates and grants a role to you named after the policy (for example, `ACCESS_LOCATIONS_DBA`), which enables you to manage this policy during its lifetime.

5.11.2 Creating Policy Components Using Cloud Control

After you create a container for the policy and set enforcement options for it, you can create components for the policy.

1. In the Oracle Label Security Policies page, select the policy you just created. Click **Edit**.
2. In the Edit Label Security Policy page, select the **Label Components** tab.
3. Click **Add 5 Rows** under Levels to add levels for the policy. Enter a Long Name, Short Name, and Numeric Tag for each level that you create. The numeric tag corresponds to the sensitivity of the level. To create more levels, you can click **Add 5 Rows** again. Use the same steps to create compartments and rows. For compartments and groups, the numeric tags do not correspond to sensitivity.

   At a minimum, you must create one level, such as `SECRET`. Creating compartments and groups is optional.

   The level numbers indicate the level of sensitivity for their corresponding labels. A greater number implies greater sensitivity. Select a numeric range that can be expanded later on, in case your security policy needs more levels. For example, if you have created levels `PUBLIC (7000)` and `SENSITIVE (8000)`, and you now want to create an intermediate level called `CONFIDENTIAL`, then you can assign the numeric value 7500 to this level.

   Compartments identify categories associated with data, providing a finer level of granularity within a level. For example, a single table might have data corresponding to different departments that you might like to separate using compartments. Compartments are optional.

   Groups identify organizations owning or accessing the data. Groups are useful for the controlled dissemination of data and for timely reaction to organizational change. Groups are optional.
4. Click **Apply**.

5.11.3 Creating Data Labels for the Policy Using Cloud Control

You can create data labels for an Oracle Label Security policy in Cloud Control.

1. In the Label Security Policies page, select the policy that needs to have labels linked to levels.
2. In the **Actions** box, select Data Labels. Click **Go**.

   The Data Labels page appears.
3. Click **Add**.

   The Create Data Label page appears.
4. Enter the following information:

- **Numeric Tag**: Enter a number that uniquely identifies the label. This number should be unique across all policies.
- **Level**: Select a level from the list.

5. You can optionally select Compartments to add to the label. To add compartments, click **Add** under Compartments. Select the compartments to be added to the label. Click **Select** to add the compartments.
6. Optionally, to add groups, click **Add** under **Groups**. Select the groups to be added to the label. Click **Select** to add the groups.
7. Click **OK** in the Create Data Label page.

   The data label appears in the Data Labels page.
8. Repeat steps 3 to 7 to create more data labels.

Alternatively, you can use the `SA_LABEL_ADMIN` package to define label components for a policy.

Related Topics

- SA_LABEL_ADMIN Label Management PL/SQL Package
  The `SA_LABEL_ADMIN` PL/SQL package provides an administrative interface to manage the labels used by a policy.

5.11.4 Authorizing and Granting Privileges for a Policy Using Cloud Control

You can authorize and grant privileges to users for a policy during the user creation process.

1. In the Label Security Policies page, select the policy that needs authorization.
2. In the **Actions** box, select **Authorization**. Click **Go**.

   The Create User page appears.
3. Add users as follows:
   - Under Database Users, click **Add**. In the Search and Select window, select users that you want and then click **Select**.
   - Under Non Database Users, click **Add 5 Rows**, and then add the user names of the non-database users that you want to add. Most application users are considered non-database users. A non-database user does not exist in the database. This can be any user name that meets the Oracle Database naming standards and can fit into the `VARCHAR2(30)` length field. However, be aware that Oracle Database does not automatically configure the associated security information for the non-database user when the application connects to the database. In this case, the application needs to call an Oracle Label Security function to assume the label authorizations of the specified user who is not a real database user.
4. In the Create User page, select the user that you want to authorize. Click **Next**. If you have multiple users that need the same authorizations, then select all users who need the same authorizations. Click **Next**.

   The Privileges step appears.

5. Next, you can assign privileges to the user you selected in the preceding step. Privileges allow a database user to bypass certain controls enforced by the policy. Select the privileges you want to grant. Click **Next**.

   If you do not want to assign any privileges to the user, then click **Next** without selecting any privileges.

   The Labels, Compartments, and Groups step appears.

6. Next, to create the user label for the user: under Levels, use the flashlight icon to select data to enter for the following fields:

   - **Maximum Level**: Enter the highest level for read and write access for this user.
   - **Minimum Level**: Enter the lowest level for write access.
   - **Default Level**: Enter the default level when the user logs in.

     This value is equal to or greater than the minimum level and equal to or less than the maximum level.
   - **Row Level**: Enter the level given to the row when user writes to the table.

7. Click **Add** under Compartments, to add compartments to the user label. Select the compartments to add. Click **Select**.

8. For each compartment that you add, you can select the following properties:

   - **Write**: Allows the user to write to data that has the compartment as part of its label
   - **Default**: Adds the compartment to the user's default session label
   - **Row**: Adds the compartment to the data label when the user writes to the table

9. Click **Add** under Groups, to add groups to the user label. Select the groups and click **Select**.

10. For each group that you add, you can select the following properties:

    - **Write**: Allows the user to write to data that has the group as part of its label
    - **Default**: Adds the group to the user's default session label
    - **Row**: Adds the group to the data label when the user writes to the table

11. Click **Next**.

    The Audit step appears.

12. Bypass the Audit step by clicking **Next**.

    You cannot configure traditional auditing in a new Oracle Label Security policy. Instead, you must create a unified audit policy using SQL*Plus.

13. You can review the policy authorization settings. Click **Finish** to create the policy authorization. Alternatively, you can click **Back** to modify the authorization settings.

Alternatively, you can use the `SA_USER_ADMIN` package to authorize users.

5.11.5 Granting Privileges to Trusted Program Units Using Cloud Control

You can grant privileges to trusted program units in Cloud Control.

1. In the Label Security Policies page, select the policy that needs authorization.

2. In the **Actions** box, select Authorization. Click **Go**.

   The Authorization page appears.

3. Click the **Trusted Program Units** tab.
4. Click **Add** to add Oracle Label Security privileges for a procedure, function, or package.

   The **Create Program Unit** page appears.
5. Enter the name of the procedure, function, or package, for which the privileges need to be granted, in the **Program Unit** field. You can also use the **Search** icon to search for the procedure, function, or package.
6. Select one or more policy-specific privileges that need to be granted to the program unit. Click **OK**.

   The trusted program unit is added to the Authorizations page.

Alternatively, you can use the **SA_USER_ADMIN** package to authorize trusted program units.

Related Topics

- Administering and Using Trusted Stored Program Units
  You can use trusted stored program units to enhance system security.

5.11.6 Applying a Policy to a Database Table with Cloud Control

You can apply an Oracle Label Security policy to a database table in Cloud Control.

1. In the Label Security Policies page, select the policy that needs to be applied to a table.
2. Select Apply from the **Actions** box. Click **Go**.

   The Apply page appears.
3. Select the **Tables** tab to apply the policy to a table.

   Select the **Schemas** tab if you are applying the policy to a schema.The process is same as applying the policy to a table.
4. Click **Create**.

   The Add Table page appears.
5. Next to the **Table** box, click the flashlight icon.
6. In the Search and Select window, enter the following information under Search:
   - **Schema**: Enter the name of the schema in which the table appears. Leaving this field empty displays tables in all schemas.
   - **Name**: Optionally, enter the name of the table. Leaving this box empty displays all the tables within the schema.

   To narrow the search by using wildcards, use the percent (%) sign. For example, enter `O%` to search for all tables beginning with the letter O.
7. Select the table and click **Select**.

   The Add Table page appears.
8. Enter the following information:
   - **Policy Enforcement Options**: Select enforcement options as needed. These options will apply to the table on top of the enforcement options that you selected when you created the label security policy container.

To make no change from those enforcement options, that is, to use the same enforcement options created earlier, select **Use Default Policy Enforcement**. To add more enforcement options, select from the other options listed.

- **Labeling Function**: Optionally, specify a labeling function to automatically compute the label to be associated with a new or updated row. That function is always invoked thereafter to provide the data labels written under that policy, because active labeling functions take precedence over any alternative means of supplying a label.
- **Predicate**: Optionally, specify an additional predicate to combine (using `AND` or `OR`) with the label-based predicate for `READ_CONTROL`.

9. Click **OK**.

Related Topics

- Step 1: Create the Label Security Policy Container
  The Oracle Label Security policy container is a storage place for the policy settings.

5.11.7 Applying Policy Labels to Table Rows Using Cloud Control

You can apply Oracle Label Security policy labels to table rows in Cloud Control.

1. In the Label Security Policies page, select the policy, for example, `ACCESS_LOCATIONS`.
2. Select Authorization from the **Actions** box. Click **Go**.

   The Authorization page appears.
3. Click **Add**.

   The Create User page appears.
4. Under Database Users, click **Add**.

   The Search and Select window appears.
5. Select the check box corresponding to the user that owns the table. Click **Select**.

   The Create User page lists the user that was added.
6. Click **Next**.

   The Privileges step appears.
7. Select the appropriate privileges for the user, and then click **Next**.

   The Labels, Compartments, and Groups page appears.
8. Click **Next**.

   The Audit step appears.
9. Click **Next**.

   The Review step appears.
10. Click **Finish**.

6 Working with Labeled Data

You can manage labeled data, view that data of security attributes for a session, and change the value of session attributes.

> **Note:**
>
> Many of the examples in this guide use the `HUMAN_RESOURCES` sample policy. Its policy name is `HR` and its policy label column is `HR_LABEL`. Unless otherwise noted, the examples assume that the SQL statements are performed on rows within the user's authorization and with full Oracle Label Security policy enforcement in effect.

- How Policy Label Column and Label Tags Work
  You should understand how policy label columns in a table or schema are created and filled.
- Assignments of Labels to Data Rows
  For existing data rows, labels can be assigned by a labeling function that you create.
- Presenting the Label
  When you retrieve labels, you do not automatically obtain the character string value.
- Filtration of Data Using Labels
  When SQL statements are processed, Oracle Label Security makes calls to the security policies defined in the database by create-and-apply procedures.
- Inserting Labeled Data
  You can insert labeled data in a variety of situations.
- Changing Session and Row Labels
  During a session, a user can change labels based on the authorizations an administrator sets.

6.1 How Policy Label Column and Label Tags Work

You should understand how policy label columns in a table or schema are created and filled.

- The Policy Label Column
  You should understand how to use policy label columns.
- Label Tags
  You can create label tags, either manually or automatically generating them, that define the label components.

6.1.1 The Policy Label Column

You should understand how to use policy label columns.

- About the Policy Label Column
  Each policy that is applied to a table creates a column in the database.
- Hiding the Policy Label Column
  You can choose not to display the column representing a policy.

6.1.1.1 About the Policy Label Column

Each policy that is applied to a table creates a column in the database.

By default, the data type of the `NUMBER`.

Each row's label for that policy is represented by a tag in that column, using the numeric equivalent of the character-string label value. The label tag is automatically generated when the label is created, unless the administrator specifies the tag manually at that time.

The automatic label generation follows the rules established by the administrator while defining the label components.

Note:

The act of creating a policy does not in itself have any effect on tables or schemas. It only applies the policy to a table or schema.

Related Topics

- Understanding Data Labels and User Labels
  You should understand fundamental concepts of data labels and user labels.

6.1.1.2 Hiding the Policy Label Column

You can choose not to display the column representing a policy.

Note:

You cannot hide columns in materialized views.

- To hide the display of a column, apply the `HIDE` option to the table.

  After a policy using `HIDE` is applied to a table, a user running a `SELECT *` or performing a `DESCRIBE` operation will not see the policy label column. If the policy label column is not hidden, then the label tag is displayed as data type `NUMBER`. The following example shows the output of the `EMP` table, with the `HR_LABEL` column showing:

```
DESCRIBE EMP;
 Name                                      Null?    Type
 ----------------------------------------- -------- --------
 EMPNO                                     NOT NULL NUMBER(4)
 ENAME                                              CHAR(10)
 JOB                                                CHAR(9)
 MGR                                                NUMBER(4)
 SAL                                                NUMBER(7,2)
 DEPTNO                                    NOT NULL NUMBER(2)
 HR_LABEL                                           NUMBER(10)
```

Here is how the same table appears with the `HR_LABEL` column hidden:

```
DESCRIBE EMP;
 Name                                      Null?    Type
 ----------------------------------------- -------- --------
 EMPNO                                     NOT NULL NUMBER(4)
 ENAME                                              CHAR(10)
 JOB                                                CHAR(9)
 MGR                                                NUMBER(4)
 SAL                                                NUMBER(7,2)
 DEPTNO                                    NOT NULL NUMBER(2)
```

Related Topics

- How the HIDE Policy Column Option Works
  You can specify the `HIDE` policy configuration option when you add an Oracle Label Security policy column to a table.

6.1.2 Label Tags

You can create label tags, either manually or automatically generating them, that define the label components.

- About Label Tags
  The administrator first defines a set of label components to be used in a policy.
- Manually Defined Label Tags to Order Labels
  By manually defining label tags, you can implement a data manipulation strategy that permits labels to be meaningfully sorted and compared.
- Manually Defined Label Tags to Manipulate Data
  An administratively defined label tag is a convenient way to reference a complete label string (that is, a combination of label components).
- Automatically Generated Label Tags
  Dynamically generated label tags have 10 digits, with no relationship to numbers assigned to any label component.

6.1.2.1 About Label Tags

The administrator first defines a set of label components to be used in a policy.

When creating labels, the administrator specifies the set of valid combinations of components that can make up a label, that is, a level optionally combined with one or more groups or compartments.

Each such valid label within a policy is uniquely identified by an associated numeric tag assigned by the administrator or generated automatically upon its first use. Manual definition has the advantage of allowing the administrator to control the ordering of label values when they are sorted or logically compared.

However, label tags must be unique across all policies in the database. When you use multiple policies in a database, you cannot use the same numeric label tag in different policies. Remember that each label tag uniquely identifies one label, and that numeric tag is what is stored in the data rows, not the label's character-string representation.

6.1.2.2 Manually Defined Label Tags to Order Labels

By manually defining label tags, you can implement a data manipulation strategy that permits labels to be meaningfully sorted and compared.

To do this, you must predefine all of the labels to be associated with protected data, and assigns to each label a meaningful label tag value. Manually assigned label tags can have up to eight digits. The value of a label tag must be greater than zero.

It may be advantageous to implement a strategy in which label tag values are related to the numeric values of label components. In this way, you can use the tags to group data rows in a meaningful way. This approach, however, is not mandatory. It is good practice to set tags for labels of higher sensitivity to a higher numeric value than tags for labels of lower sensitivity.

Table 6-1 illustrates a set of label tags that have been assigned.

Table 6-1 Administratively Defined Label Tags (Example)

Label Tag	Label String
10000	P
20000	C
21000	C:FNCL
21100	C:FNCL,OP
30000	S
31110	S:OP:WR
40000	HS
42000	HS:OP

In this example, labels with a level of `PUBLIC` begin with "1", labels with a level of `CONFIDENTIAL` begin with "2", labels with a level of `SENSITIVE` begin with "3", and labels with a level of `HIGHLY_SENSITIVE` begin with "4".

Labels with the `FINANCIAL` compartment then come in the 1000 range, labels with the compartment `OP` are in the 1100 range, and so on. The tens place is used to indicate the group `WR`, for example.

Another strategy might be completely based on groups, where the tags might be 3110, 3120, 3130, and so on.

Note, however, that label tags identify the *whole* label, independent of the numeric values assigned for the individual label components. The label tag is used as a whole integer, not as a set of individually evaluated numbers.

Related Topics

- Understanding Data Labels and User Labels
  You should understand fundamental concepts of data labels and user labels.

6.1.2.3 Manually Defined Label Tags to Manipulate Data

An administratively defined label tag is a convenient way to reference a complete label string (that is, a combination of label components).

For example, the tag "31110" could stand for the complete label string "S:OP:WR".

Label tags can be used as a convenient way to partition data. For example, all data with labels in the range 1000 - 1999 could be placed in tablespace A, all data with labels in the range 2000 - 2999 could be placed in tablespace B, and so on.

This simplified notation also comes in handy when there is a finite number of labels and you need to perform various operations upon them. Consider a situation in which one company hosts a human resources system for many other companies. Assume that all users from Company Y have the label "C:ALPHA:CY", for which the tag "210" has been set. To determine the total number of application users from Company Y, the host administrator can enter:

```
SELECT * FROM tab1
  WHERE hr_label = 210;
```

6.1.2.4 Automatically Generated Label Tags

Dynamically generated label tags have 10 digits, with no relationship to numbers assigned to any label component.

You cannot group the data by label.

Table 6-2 describes how automatically generated label tags work.

Table 6-2 Generated Label Tags (Example)

Label Tag	Label String
100000020	P
100000052	C
100000503	C:FNCL
100000132	C:FNCL,OP
100000003	S
100000780	S:OP:WR
100000035	HS
100000036	HS:OP

6.2 Assignments of Labels to Data Rows

For existing data rows, labels can be assigned by a labeling function that you create.

In such a function, you specify the exact table and row conditions defining what label to insert. The function can be named in the call to apply a policy to a table or schema, or in an update by the administrator.

Related Topics

- Inserting Labeled Data
  You can insert labeled data in a variety of situations.
- Labeling Functions
  Labeling functions can compute and return a label using resources such as context variables (for example, date or username) and data values.

- SA_POLICY_ADMIN.APPLY_TABLE_POLICY
  The `SA_POLICY_ADMIN.APPLY_TABLE_POLICY` procedure adds the specified policy to a table.
- SA_POLICY_ADMIN.APPLY_SCHEMA_POLICY
  The `SA_POLICY_ADMIN.APPLY_SCHEMA_POLICY` procedure applies a policy to all of the tables in a schema and enables the policy for these tables.

6.3 Presenting the Label

When you retrieve labels, you do not automatically obtain the character string value.

By default, the label tag value is returned. Two label manipulation functions enable you to convert the label tag value to and from its character string representation.

- Converting a Character String to a Label Tag with CHAR_TO_LABEL
  The `CHAR_TO_LABEL` function converts character strings to a label tag, returning the label tag for the specified character string.
- Conversion of a Label Tag to a Character String, with LABEL_TO_CHAR
  You can convert label tags to character strings.

6.3.1 Converting a Character String to a Label Tag with CHAR_TO_LABEL

The `CHAR_TO_LABEL` function converts character strings to a label tag, returning the label tag for the specified character string.

- To convert a character string to a label tag, use the following syntax for the `CHAR_TO_LABEL` function:

  ```
  FUNCTION CHAR_TO_LABEL (
       policy_name     IN VARCHAR2,
       label_string    IN VARCHAR2)
  RETURN NUMBER;
  ```

  For example:

  ```
  INSERT INTO emp (empno,hr_label)
  VALUES (999, CHAR_TO_LABEL('HR','S:A,B:G5');
  ```

6.3.2 Conversion of a Label Tag to a Character String, with LABEL_TO_CHAR

You can convert label tags to character strings.

- Converting a Label Tag to a Character String with LABEL_TO_CHAR
  The `LABEL_TO_CHAR` function returns a `VARCHAR2` string when it converts a label tag to a character string.
- LABEL_TO_CHAR Examples
  Oracle provides examples that illustrate the use of `LABEL_TO_CHAR`.
- Retrieving All Columns from a Table When the Policy Label Column Is Hidden
  If the policy label column is hidden, then it is not automatically returned when you run `SELECT *` on the table.

6.3.2.1 Converting a Label Tag to a Character String with LABEL_TO_CHAR

The `LABEL_TO_CHAR` function returns a `VARCHAR2` string when it converts a label tag to a character string.

When you query a table or view, you automatically retrieve all of the rows in the table or view that satisfy the qualifications of the query and are dominated by your label. If the policy label column is not hidden, then the label tag value for each row is displayed. You must use the `LABEL_TO_CHAR` function to display the character string value of each label. Note that all conversions must be explicit. There is no automatic casting to and from tag and character string representations.

- To convert a label tag to a character string, use the following syntax for the `LABEL_TO_CHAR` function:

  ```
  FUNCTION LABEL_TO_CHAR (
       label IN NUMBER)
  RETURN VARCHAR2;
  ```

6.3.2.2 LABEL_TO_CHAR Examples

Oracle provides examples that illustrate the use of `LABEL_TO_CHAR`.

Example: Retrieving a Row Label from a Table or a View

To retrieve the label of a row from a table or view, specify the policy label column in the `SELECT` statement.

```
SELECT label_to_char (hr_label) AS label, ename FROM tab1
  WHERE ename = 'RWRIGHT';
```

This statement returns the following:

```
LABEL           ENAME
------------    ----------
S:A,B:G1        RWRIGHT
```

Example: Retrieving a Policy Label Column

You can also specify the policy label column in the `WHERE` clause of a `SELECT` statement.

The following statement displays all rows that have the policy label `S:A,B:G1`

```
SELECT label_to_char (hr_label) AS label,ename FROM emp
  WHERE hr_label = char_to_label ('HR', 'S:A,B:G1');
```

This statement returns the following:

```
LABEL            ENAME
-------------    ---------
S:A,B:G1         RWRIGHT
S:A,B:G1         ESTANTON
```

Alternatively, you could use a more flexible statement to look up data that contains the string "S:A,B:G1" anywhere in the text of the `HR_LABEL` column:

```
SELECT label_to_char (hr_label) AS label,ename FROM emp
  WHERE label_to_char (hr_label) like '%S:A,B:G1%';
```

If you do not use the `LABEL_TO_CHAR` function, then you will see the label tag.

Example: Retrieving a Numeric Column Data Type

The following example is with the numeric column data type (`NUMBER`) and dynamically generated label tags, but without using the `LABEL_TO_CHAR` function. If you do not use the `LABEL_TO_CHAR` function, then you will see the label tag.

```
SQL> select empno, hr_label from emp
     where ename='RWRIGHT';

EMPNO      HR_LABEL
---------- ----------
7839       1000000562
```

6.3.2.3 Retrieving All Columns from a Table When the Policy Label Column Is Hidden

If the policy label column is hidden, then it is not automatically returned when you run `SELECT *` on the table.

- To explicitly specify that you want to retrieve a label, use the `LABEL_TO_CHAR` function in the `SELECT` statement.

  For example, to retrieve all columns from the `DEPT` table (including the policy label column in its character representation), enter the following:

  ```
  COLUMN LABEL FORMAT a10
  SELECT LABEL_TO_CHAR (hr_label) AS LABEL, DEPT.* FROM DEPT;
  ```

  Running these SQL statements returns the following data:

  Table 6-3 Data Returned from Sample SQL Statements re Hidden Column

LABEL	DEPTNO	DNAME	LOC
L1	10	ACCOUNTING	NEW YORK
L1	20	RESEARCH	DALLAS
L1	30	SALES	CHICAGO
L1	40	OPERATIONS	BOSTON

  By contrast, if you do not explicitly specify the `HR_LABEL` column, the label is not displayed at all. Note that while the policy column name is on a policy basis, the `HIDE` option is on a table-by-table basis.

Related Topics

- How the HIDE Policy Column Option Works
  You can specify the `HIDE` policy configuration option when you add an Oracle Label Security policy column to a table.

6.4 Filtration of Data Using Labels

When SQL statements are processed, Oracle Label Security makes calls to the security policies defined in the database by create-and-apply procedures.

For SELECT statements, the policy filters the data rows that the user is authorized to see. For INSERT, UPDATE, and DELETE statements, Oracle Label Security permits or denies the requested operation, based on the user's authorizations.

- Use of Numeric Label Tags in WHERE Clauses
  There are different techniques of using numeric label tags in WHERE clauses of SELECT statements.
- Ordering Labeled Data Rows
  The ORDER BY clause of a SELECT statement can be used to order rows by the numeric label tag.
- Ordering by Character Representation of Label
  The LABEL_TO_CHAR function orders data rows by the character representation of the label.
- Determination of the Upper and Lower Bounds of Labels
  Oracle Label Security provides functions that determine the least upper bound or the greatest lower bound of two or more labels.
- Merging Labels with the MERGE_LABEL Function
  The MERGE_LABEL function merges two labels together.

6.4.1 Use of Numeric Label Tags in WHERE Clauses

There are different techniques of using numeric label tags in WHERE clauses of SELECT statements.

When using labels in the NUMBER format, you can set up labels so that a list of your label tags distinguishes the different levels. Comparisons of these numeric label tags can be used for ORDER BY processing, and with the logical operators.

For example, if you have assigned all UNCLASSIFIED labels to the 1000 range, all SENSITIVE labels to the 2000 range, and all HIGHLY_SENSITIVE labels to the 3000 range, then you can list all SENSITIVE records.

```
SELECT * FROM emp
WHERE hr_label BETWEEN 2000 AND 2999;
```

To list all SENSITIVE and UNCLASSIFIED records, you can enter:

```
SELECT * FROM emp
WHERE hr_label <3000;
```

To list all HIGHLY_SENSITIVE records, you can enter:

```
SELECT * FROM emp
WHERE hr_label=3000;
```

Note:

Remember that such queries have meaning only if the administrator has applied a numeric ordering strategy to the label tags that they originally assigned to the labels. In this way, the administrator can provide for convenient dissemination of data. If, however, the label tag values are generated automatically, then there is no intrinsic relationship between the value of the tag and the order of the labels.

Alternatively, you can use dominance relationships to set up an ordering strategy.

Related Topics

- Using Dominance Functions
  Oracle Label Security provides functions to control dominance.

6.4.2 Ordering Labeled Data Rows

The ORDER BY clause of a SELECT statement can be used to order rows by the numeric label tag.

- To perform the ORDER BY operation, use a SELECT statement similar to the following:

  ```
  SELECT * from emp
  ORDER BY hr_label;
  ```

  No functions were necessary in this statement. The statement made use of label tags set up by the administrator.

  > **Note:**
  >
  > Again, such queries have meaning only if the administrator has applied a numeric ordering strategy to the label tags originally assigned to the labels.

6.4.3 Ordering by Character Representation of Label

The LABEL_TO_CHAR function orders data rows by the character representation of the label.

- To order data rows by the character representation of a label, use a statement similar to the following, which returns all rows sorted by the text order of the label:

  ```
  SELECT * FROM emp
  ORDER BY label_to_char (hr_label);
  ```

6.4.4 Determination of the Upper and Lower Bounds of Labels

Oracle Label Security provides functions that determine the least upper bound or the greatest lower bound of two or more labels.

Two single-row functions operate on each row returned by a query. They return one result for each row.

> **Note:**
>
> In all functions that take multiple labels, the labels must all belong to the same policy.

- Finding the Least Upper Bound with OLS_LEAST_UBOUND
  The `OLS_LEAST_UBOUND` (`OLS_LUBD`) function returns a character string label that is the least upper bound of *`label1`* and *`label2`*:.
- Finding Greatest Lower Bound with OLS_GREATEST_LBOUND
  The `OLS_GREATEST_LBOUND` (`OLS_GLBD`) standalone function determines the lowest label of the data that can be involved in an operation, given two different labels.

6.4.4.1 Finding the Least Upper Bound with OLS_LEAST_UBOUND

The `OLS_LEAST_UBOUND` (`OLS_LUBD`) function returns a character string label that is the least upper bound of *`label1`* and *`label2`*:.

That is, the one label that dominates both. The least upper bound is the highest level, the union of the compartments in the labels, and the union of the groups in the labels. For example, the least upper bound of `HIGHLY_SENSITIVE:ALPHA` and `SENSITIVE:BETA` is `HIGHLY_SENSITIVE:ALPHA,BETA`.

- To find the least upper bound, use the following syntax:

```
FUNCTION OLS_LEAST_UBOUND (
     label1                    IN NUMBER,
     label2                    IN NUMBER)
RETURN VARCHAR2;
```

The `OLS_LEAST_UBOUND` function is useful when joining rows with different labels, because it provides a high water mark label for joined rows.

The following query compares each employee's label with the label of their department, and returns the higher label, whether it be in the EMP table or the DEPT table.

```
SELECT ename,dept.deptno,
  OLS_LEAST_UBOUND(emp.hr_label,dept.hr_label) as label
  FROM emp, dept
  WHERE emp.deptno=dept.deptno;
```

This query returns the following data:

Table 6-4 Data Returned from Sample SQL Statements from OLS_LEAST_UBOUND

ENAME	DEPTNO	LABEL
KING	10	L3:M:D10
BLAKE	30	L3:M:D30
CLARK	10	L3:M:D10
JONES	20	L3:M:D20
MARTIN	30	L2:E:D30

6.4.4.2 Finding Greatest Lower Bound with OLS_GREATEST_LBOUND

The `OLS_GREATEST_LBOUND` (`OLS_GLBD`) standalone function determines the lowest label of the data that can be involved in an operation, given two different labels.

This function returns a character string label that is the greatest lower bound of *`label1`* and *`label2`*. The greatest lower bound is the lowest level, the intersection of the compartments in the labels and the groups in the labels. For example, the greatest lower bound of `HIGHLY_SENSITIVE:ALPHA` and `SENSITIVE` is `SENSITIVE`.

- To find the greatest lower bound, use the following syntax:

```
FUNCTION OLS_GREATEST_LBOUND (
     label1                IN NUMBER,
     label2                IN NUMBER)
RETURN VARCHAR2;
```

6.4.5 Merging Labels with the MERGE_LABEL Function

The `MERGE_LABEL` function merges two labels together.

It accepts the numeric form of two labels and the three-character specification of a merge format.

- To merge labels, use the following syntax:

```
FUNCTION merge_label (label1 IN number,
                      label2 IN number,
                      merge_format IN VARCHAR2)
RETURN number;
```

The valid merge format is specified with a three-character string:

`highest_level_or_lowest_level union_or_intersection_of_compartments union_or_intersection_of_groups`

  – The first character indicates whether to merge using the highest level or the lowest level of the two labels.
  – The second character indicates whether to merge using the union or the intersection of the compartments in the two labels.
  – The third character indicates whether to merge using the union or the intersection of the groups in the two labels.

Table 6-5 defines the `MERGE_LABEL` format constants.

Table 6-5 MERGE_LABEL Format Constants

Format Specification	Data Type	Consta nt	Meaning	Positions in Which Format Is Used
`max_lvl_fmt`	`CONSTANT varchar2(1)`	`H`	Maximum level	First (level)
`min_lvl_fmt`	`CONSTANT varchar2(1)`	`L`	Minimum level	First (Level)
`union_fmt`	`CONSTANT varchar2(1)`	`U`	Union of the two labels	Second (compartments) and Third (groups)
`inter_fmt`	`CONSTANT varchar2(1)`	`I`	Intersection of the two labels	Second (compartments) and Third (groups)

Table 6-5 (Cont.) MERGE_LABEL Format Constants

Format Specification	Data Type	Constant	Meaning	Positions in Which Format Is Used
`minus_fmt`	`CONSTANT varchar2(1)`	`M`	Remove second label from first label	Second (compartments) and Third (groups)
`null_fmt`	`CONSTANT varchar2(1)`	`N`	If specified in compartments column, returns no compartments. If specified in groups column, returns no groups.	Second (compartments) and Third (groups)

For example, `HUI` specifies the highest level of the two labels, union of the compartments, intersection of the groups.

The `MERGE_LABEL` function is particularly useful to developers if the `OLS_LEAST_UBOUND` function does not provide the intended result. The `OLS_LEAST_UBOUND` function, when used with two labels containing groups, may result in a less sensitive data label than expected. The `MERGE_LABEL` function enables you to compute an intersection on the groups, instead of the union of groups that is provided by the `OLS_LEAST_UBOUND` function.

For example, if the label of one data record contains the group `UNITED_STATES`, and the label of another data record contains the group `UNITED_KINGDOM`, and the `OLS_LEAST_UBOUND` function is used to compute the least upper bound of these two labels, then the resulting label would be accessible to users authorized for either the `UNITED_STATES` or the `UNITED_KINGDOM`.

If, by contrast, the `MERGE_LABEL` function is used with a format clause of `HUI`, then the resulting label would contain the highest level, the union of the compartments, and no groups. This is because `UNITED_STATES` and `UNITED_KINGDOM` do not intersect.

6.5 Inserting Labeled Data

You can insert labeled data in a variety of situations.

- About Inserting Labeled Data
  When you insert data into a table protected by an Oracle Label Security policy, you must supply a numeric label value tag.
- Inserting Labels Using CHAR_TO_LABEL
  To insert a row label, you can specify the label character string and then transform it into a label using the `CHAR_TO_LABEL` function.
- Inserting Labels Using Numeric Label Tag Values
  You can insert data using the numeric label tag value of a label, rather than using the `CHAR_TO_LABEL` function.
- Inserting Data Without Specifying a Label
  There are two situations in which you do not need to specify a label in `INSERT` statements.

- Inserting Data When the Policy Label Column Is Hidden
  If the label column is hidden, then the existence of the column is transparent to the insertion of data.
- Inserting Labels Using TO_DATA_LABEL
  The `TO_DATA_LABEL` function can generate new labels dynamically.

6.5.1 About Inserting Labeled Data

When you insert data into a table protected by an Oracle Label Security policy, you must supply a numeric label value tag.

Usually, you can insert this value in the `INSERT` statement itself.

To do this, you must explicitly specify the tag for the desired label or explicitly convert the character string representation of the label into the correct tag. Note that this does not mean generating new label tags, but referencing the correct tag.

The only times an `INSERT` statement may omit a label value are:

- If the `LABEL_DEFAULT` enforcement option was specified when the policy was applied, or
- If no enforcement options were specified when the policy was applied and `LABEL_DEFAULT` was specified when the policy was created
- If the statement applying the policy named a labeling function.

In the first two cases, the user's session default row label is used as the inserted row's label. In the third case, the inserted row's label is created by that labeling function.

Related Topics

- Labeling Functions
  Labeling functions can compute and return a label using resources such as context variables (for example, date or username) and data values.
- Implementing Policy Enforcement Options and Labeling Functions
  You can customize the enforcement of Oracle Label Security policies and implement labeling functions.

6.5.2 Inserting Labels Using CHAR_TO_LABEL

To insert a row label, you can specify the label character string and then transform it into a label using the `CHAR_TO_LABEL` function.

The `CHAR_TO_LABEL` function automatically creates a valid data label.

- To insert labels, use an `INSERT INTO` statement.

  Using the definition for table `emp`, the following example shows how to insert data with explicit labels:

  ```
  INSERT INTO emp (ename,empno,hr_label)
  VALUES ('ESTANTON',10,char_to_label ('HR', 'SENSITIVE'));
  ```

6.5.3 Inserting Labels Using Numeric Label Tag Values

You can insert data using the numeric label tag value of a label, rather than using the `CHAR_TO_LABEL` function.

- To insert labels using numeric label tag values, use an `INSERT INTO` statement.

  For example, if the numeric label tag for `SENSITIVE` is 3000, it would appear as follows:

  ```
  INSERT INTO emp (ename, empno, hr_label)
  VALUES ('ESTANTON', 10, 3000);
  ```

6.5.4 Inserting Data Without Specifying a Label

There are two situations in which you do not need to specify a label in `INSERT` statements.

If `LABEL_DEFAULT` is set, or if there is a labeling function applied to the table, then you do not need to specify a label in your `INSERT` statements. The label will be provided automatically.

- To insert data without specifying a label, use an `INSERT INTO` statement.

  For example:

  ```
  INSERT INTO emp (ename, empno)
  VALUES ('ESTANTON', 10);
  ```

  The resulting row label is set according to the default value (or by a labeling function).

6.5.5 Inserting Data When the Policy Label Column Is Hidden

If the label column is hidden, then the existence of the column is transparent to the insertion of data.

`INSERT` statements can be written that do not explicitly list the table columns and do not include a value for the label column. The session's row label is used to label the data, or a labeling function is used if one was specified when the policy was applied to the table or schema. You can insert into a table without explicitly naming the columns, as long as you specify a value for each non-hidden column in the table. The following example shows how to insert a row into a table.

- To insert data when the policy label column is hidden, use the following syntax:

  ```
  INSERT INTO emp
  VALUES ('196','ESTANTON',Technician,RSTOUT,50000,10);
  ```

  Its label will be one of the following three possibilities:

  – The label you specify

  – The label established by the `LABEL_DEFAULT` option of the policy being applied

  – The label created by a labeling function named by the policy being applied

> **Note:**
>
> If the policy label column is *not* hidden, then you must explicitly include a label value (possibly null, indicated by a comma) in the `INSERT` statement.

6.5.6 Inserting Labels Using TO_DATA_LABEL

The `TO_DATA_LABEL` function can generate new labels dynamically.

This approach guarantees that the data labels are valid.

1. Ensure that you have the `EXECUTE` privilege on the `TO_DATA_LABEL` function.
2. Use the `TO_DATA_LABEL` as necessary, for example, in an `INSERT INTO` statement.

   For example:

   ```
   INSERT INTO emp (ename, empno, hr_label)
   VALUES ('ESTANTON', 10, to_data_label ('HR', 'SENSITIVE'));
   ```

> **Note:**
>
> The `TO_DATA_LABEL` function must be explicitly granted to individuals, in order to be used. Its usage should be tightly controlled.

6.6 Changing Session and Row Labels

During a session, a user can change labels based on the authorizations an administrator sets.

Related Topics

- SA_SESSION Session Management PL/SQL Package
  The `SA_SESSION` PL/SQL package manages session behavior for user authorizations.

Part III
Oracle Label Security Tutorials

Part III provides tutorials on how to create Oracle Label Security policies.

- Tutorial: Configuring Levels in Oracle Label Security
  This tutorial demonstrates how to create Oracle Label Security levels.
- Tutorial: Configuring Compartments in Oracle Label Security
  This tutorial demonstrates how to create Oracle Label Security compartments.
- Tutorial: Configuring Groups in Oracle Label Security
  This tutorial demonstrates how to create an Oracle Label Security parent group that has four child groups.

7

Tutorial: Configuring Levels in Oracle Label Security

This tutorial demonstrates how to create Oracle Label Security levels.

- About This Tutorial
  In this tutorial, you will use the `HR` schema to learn how to use Oracle Label Security levels.
- Step 1: Create a Role and User Accounts
  The role that you create will enable any user who is granted it to have the `SELECT` privilege on the `HR.EMPLOYEES` table. The user accounts are for the two Human Resources employees, Susan Mavris and Ida Neau.
- Step 2: Create the Oracle Label Security Policy Container
  As an Oracle Label Security administrator, you must create and then enable the policy container.
- Step 3: Create the Two Level Components for the Oracle Label Security Policy
  After you create the Oracle Label Security policy container, you are ready to create two levels to represent two different levels of sensitivity.
- Step 4: Create the Data Labels for the Levels
  A data label tags data records for use with the Oracle Label Security policy.
- Step 5: Set User Authorizations for the Oracle Label Security Policy
  Setting user authorizations entails associating the user with the policy and the minimum and maximum levels that are associated with the Oracle Label Security policy.
- Step 6: Apply the Oracle Label Security Policy to the HR Schema
  After you apply the policy to the `HR` schema, you must enable the policy association with `HR`.
- Step 7: Add the Policy Labels to the HR.EMPLOYEES Table Data
  Both the Oracle Label Security administrator and the `HR` user will add the policy labels to the `HR.EMPLOYEES` table data in the `EMPLOYEE_ID` column.
- Step 8: Test the Oracle Label Security Policy
  To test the policy, each user will try to query the `HR.EMPLOYEES` table.
- Step 9: Optionally, Remove the Oracle Label Security Policy Components
  You can remove the Oracle Label Security policy, `HR_ROLE` role, and users Ida Neau and Susan Mavris.

7.1 About This Tutorial

In this tutorial, you will use the `HR` schema to learn how to use Oracle Label Security levels.

The Human Resources representative, Susan Mavris, has an assistant, Ida Neau. Susan Mavris must have access to all employee records, including records of employees who have left the company. Ida Neau must have access only to employees who are current.

You will create an Oracle Label Security policy that will use the following levels of sensitivity to govern access to current and former employees:

- `SENSITIVE` enables access to current employees only. User Ida Neau will be assigned this level.
- `HIGHLY_SENSITIVE` enables access to former employees. User Susan Mavris will be assigned this level. This level is a higher level than `SENSITIVE`, which means that it will also provide access to rows protected by `SENSITIVE`. In other words, Susan Mavris will have access to both former and current employee records.

7.2 Step 1: Create a Role and User Accounts

The role that you create will enable any user who is granted it to have the `SELECT` privilege on the `HR.EMPLOYEES` table. The user accounts are for the two Human Resources employees, Susan Mavris and Ida Neau.

1. Log in to a PDB as a user who has privileges to create roles, grant privileges, and create user accounts.

   For example:

   ```
   sqlplus sec_admin@pdb_name
   Enter password: password
   ```

   To find the available PDBs, 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 role as follows:

   ```
   CREATE ROLE HR_ROLE;
   ```

3. Grant the `SELECT` privilege on `HR.EMPLOYEES` to `HR_ROLE`.

   ```
   GRANT SELECT ON HR.EMPLOYEES TO HR_ROLE;
   ```

4. Create the user accounts for Susan Mavris and Ida Neau, and grant them the `HR_ROLE` role.

   ```
   GRANT CONNECT, HR_ROLE TO SMAVRIS IDENTIFIED BY password;
   GRANT CONNECT, HR_ROLE TO INEAU IDENTIFIED BY password;
   ```

7.3 Step 2: Create the Oracle Label Security Policy Container

As an Oracle Label Security administrator, you must create and then enable the policy container.

1. Connect to the PDB as a user who can create and manage Oracle Label Security policies.

   For example:

   ```
   sqlplus psmith_ols@pdb_name
   Enter password: password
   ```

2. Create the policy.

   ```
   BEGIN
    SA_SYSDBA.CREATE_POLICY (
   ```

```
  policy_name      => 'HR_OLS_POL',
  column_name      => 'OLS_COL');
END;
/
```

In this specification, the `default_options` parameter is omitted because you can add it later on in another procedure.

After you create this policy container, Oracle Label Security creates and grants a role to you named `HR_OLS_POL_DBA`, which enables you to manage this policy during its lifetime.

3. Enable the policy.

```
EXEC SA_SYSDBA.ENABLE_POLICY ('HR_OLS_POL');
```

7.4 Step 3: Create the Two Level Components for the Oracle Label Security Policy

After you create the Oracle Label Security policy container, you are ready to create two levels to represent two different levels of sensitivity.

1. If necessary, connect as a user who can create and manage Oracle Label Security policies.

   For example:

```
sqlplus psmith_ols@pdb_name
Enter password: password
```

2. Create the levels as follows:

```
BEGIN
   SA_COMPONENTS.CREATE_LEVEL (
      policy_name => 'HR_OLS_POL',
      level_num   => 3000,
      short_name  => 'HS',
      long_name   => 'HIGHLY_SENSITIVE');

   SA_COMPONENTS.CREATE_LEVEL (
      policy_name => 'HR_OLS_POL',
      level_num   => 2000,
      short_name  => 'S',
      long_name   => 'SENSITIVE');
END;
/
```

   In this specification:

   - `policy_name` associates the levels with the policy container that you just created.
   - `level_num` determines how much access the user can have. Level number `3000` enables a user to have access to this level and any level number below it, in this case, level number `2000`. In other words, a user who is authorized with the `HIGHLY_SENSITIVE` level can also access data assigned to the `SENSITIVE` level.
   - `short_name` is a short-hand name for the `long_name` of the level, and will be used in other procedures to refer to the `long_name` version of the level.

7.5 Step 4: Create the Data Labels for the Levels

A data label tags data records for use with the Oracle Label Security policy.

In this procedure, the data labels will designate the rows that users Susan Mavris and Ida Neau will see in the `HR.EMPLOYEES` table. The rows labeled `HS` will correspond to the `HS` (`HIGHLY_SENSITIVE`) level to be assigned to Susan Mavris, and the rows labeled `S` will correspond with the `S` (`SENSITIVE`) level to be assigned to Ida Neau.

1. If necessary, connect as a user who can create and manage Oracle Label Security policies.

   For example:

   ```
   sqlplus psmith_ols@pdb_name
   Enter password: password
   ```

2. Create the data labels as follows:

   ```
   BEGIN
      SA_LABEL_ADMIN.CREATE_LABEL (
         policy_name  => 'HR_OLS_POL',
         label_tag    => 3100,
         label_value  => 'HS',
         data_label   => TRUE);

      SA_LABEL_ADMIN.CREATE_LABEL (
         policy_name  => 'HR_OLS_POL',
         label_tag    => 2100,
         label_value  => 'S',
         data_label   => TRUE);
   END;
   /
   ```

   In this specification:

   - `label_tag` is used internally by Oracle Label Security to identify the level. Unlike levels, it does not govern any sort of hierarchy with the labels.
   - `data_label` is set to `TRUE` so that the label can be applied to row data.

7.6 Step 5: Set User Authorizations for the Oracle Label Security Policy

Setting user authorizations entails associating the user with the policy and the minimum and maximum levels that are associated with the Oracle Label Security policy.

1. If necessary, connect as a user who can create and manage Oracle Label Security policies.

   For example:

   ```
   sqlplus psmith_ols@pdb_name
   Enter password: password
   ```

2. Authorize the users as follows:

```
BEGIN
   SA_USER_ADMIN.SET_LEVELS (
      policy_name  => 'HR_OLS_POL',
      user_name    => 'SMAVRIS',
      max_level    => 'HS',
      min_level    => 'S');

   SA_USER_ADMIN.SET_LEVELS (
      policy_name  => 'HR_OLS_POL',
      user_name    => 'INEAU',
      max_level    => 'S',
      min_level    => 'S');
END;
/
```

In this specification, the `def_level` (default level) and `row_level` parameters are omitted so that their values can default to the `max_level` parameter setting.

7.7 Step 6: Apply the Oracle Label Security Policy to the HR Schema

After you apply the policy to the `HR` schema, you must enable the policy association with `HR`.

1. If necessary, connect as a user who can create and manage Oracle Label Security policies.

   For example:

   ```
   sqlplus psmith_ols@pdb_name
   Enter password: password
   ```

2. Apply the policy to the `HR` schema.

   ```
   BEGIN
     SA_POLICY_ADMIN.APPLY_TABLE_POLICY (
       policy_name    => 'HR_OLS_POL',
       schema_name    => 'HR',
       table_name     => 'EMPLOYEES',
       table_options  => 'READ_CONTROL');
   END;
   /
   ```

   Earlier, when you created the policy with the `SA_SYSDBA.CREATE_POLICY` procedure, you did not set the `default_options` parameter, which defines the policy enforcement options. Therefore, you must set the policy enforcement here, with the `table_options` parameter of `SA_POLICY_ADMIN.APPLY_TABLE_POLICY`. `READ_CONTROL` enforces the OLS policy during the `SELECT` statement processing that the users will perform later. (It also applies to `UPDATE` and `DELETE` statement processing.)

3. Enable the policy's association with the `HR` schema.

   ```
   BEGIN
      SA_POLICY_ADMIN.ENABLE_TABLE_POLICY (
         policy_name => 'HR_OLS_POL',
         schema_name => 'HR',
         table_name  => 'EMPLOYEES');
   END;
   /
   ```

7.8 Step 7: Add the Policy Labels to the HR.EMPLOYEES Table Data

Both the Oracle Label Security administrator and the `HR` user will add the policy labels to the `HR.EMPLOYEES` table data in the `EMPLOYEE_ID` column.

1. If necessary, connect as a user who can create and manage Oracle Label Security policies.

   For example:

   ```
   sqlplus psmith_ols@pdb_name
   Enter password: password
   ```

2. Grant the `READ` privilege to the `HR_OLS_POL` policy for the `HR` user.

   ```
   BEGIN
      SA_USER_ADMIN.SET_USER_PRIVS (
         policy_name => 'HR_OLS_POL',
         user_name   => 'HR',
         privileges  => 'READ');
   END;
   /
   ```

3. Connect as the `HR` user.

   ```
   connect hr
   Enter password: password
   ```

4. Perform the following `UPDATE` statement to apply the `HIGHLY_SENSITIVE` level to the employee IDs of users who have left the company.

   This `UPDATE` statement controls the access that Susan Mavris will have to the `HR.EMPLOYEES` table because this user is authorized for the `HIGHLY_SENSITIVE` level.

   ```
   UPDATE employees
   SET    ols_col = CHAR_TO_LABEL('HR_OLS_POL','HS')
   WHERE  UPPER(employee_id) IN (200, 101, 102, 176, 201, 122, 114);
   ```

5. Perform the following `UPDATE` statement to apply the `SENSITIVE` level to the employee IDs of current employees in the company.

   This `UPDATE` statement controls the access that Ida Neau will have to the `HR.EMPLOYEES` table because this user is authorized for the `SENSITIVE` level.

   ```
   UPDATE employees
   SET    ols_col = CHAR_TO_LABEL('HR_OLS_POL','S')
   WHERE  UPPER(employee_id) NOT IN (200, 101, 102, 176, 201, 122, 114);
   ```

   This `UPDATE` statement translates to "Apply the `SENSITIVE` label to any employee who is not a former employee."

7.9 Step 8: Test the Oracle Label Security Policy

To test the policy, each user will try to query the `HR.EMPLOYEES` table.

1. Connect as user Ida Neau.

```
connect ineau@pdb_name
Enter password: password
```

2. Set column widths for the table output.

```
column first_name format a25
column last_name format a25
column ols_label format a10
```

3. Run the following query:

```
SELECT FIRST_NAME, LAST_NAME, EMPLOYEE_ID,
LABEL_TO_CHAR(OLS_COL) OLS_LABEL
FROM HR.EMPLOYEES
ORDER BY OLS_COL;
```

The output should be similar to the following:

```
FIRST_NAME                LAST_NAME                 EMPLOYEE_ID OLS_LABEL
------------------------- ------------------------- ----------- ----------
Steven                    King                              100 S
Alexander                 Hunold                            103 S
Bruce                     Ernst                             104 S
David                     Austin                            105 S
Valli                     Pataballa                         106 S
Diana                     Lorentz                           107 S
Nancy                     Greenberg                         108 S
Daniel                    Faviet                            109 S
...
100 rows selected
```

Because Ida Neau was assigned the `SENSITIVE` label, the output in the column `OLS_LABEL` is `S` (for `SENSITIVE`) only. 100 rows are returned.

Note that the Oracle Label Security restriction applies to any `SELECT` query the user makes. For example, if Ida Neau performs a `SELECT COUNT(*) FROM HR.EMPLOYEES;` query, then it would return these 100 rows, not the full 107.

4. Connect as user Susan Mavris.

```
connect smavris
Enter password: password
```

5. Run the same query that Ida Neau ran.

```
SELECT FIRST_NAME, LAST_NAME, EMPLOYEE_ID,
LABEL_TO_CHAR(OLS_COL) OLS_LABEL
FROM HR.EMPLOYEES
ORDER BY OLS_COL;
```

The output should be similar to the following:

```
FIRST_NAME                LAST_NAME                 EMPLOYEE_ID OLS_LABEL
------------------------- ------------------------- ----------- ----------
Steven                    King                              100 S
Alexander                 Hunold                            103 S
...
William                   Gietz                             206 S
Neena                     Kochhar                           101 HS
Lex                       De Haan                           102 HS
Den                       Raphaely                          114 HS
Michael                   Hartstein                         201 HS
Jonathon                  Taylor                            176 HS
Jennifer                  Whalen                            200 HS
```

```
Payam                     Kaufling                          122 HS

107 rows selected
```

Because Susan Mavris was assigned the `HIGHLY_SENSITIVE` label, the output in the column `OLS_LABEL` is `HS` (for `HIGHLY_SENSITIVE`) and `S` (for `SENSITIVE`). 107 rows are returned.

7.10 Step 9: Optionally, Remove the Oracle Label Security Policy Components

You can remove the Oracle Label Security policy, `HR_ROLE` role, and users Ida Neau and Susan Mavris.

However, if you want to try the tutorial on how to create Oracle Label Security compartments, then do not remove these components. The tutorial on compartments builds on this tutorial on levels.

1. Connect as a user who can create and manage Oracle Label Security policies.

   For example:

   ```
   sqlplus psmith_ols@pdb_name
   Enter password: password
   ```

2. Drop the Oracle Label Security policy.

   This procedure also removes the levels and the `OLS_COL` column from the `HR.EMPLOYEES` table.

   ```
   BEGIN
     SA_SYSDBA.DROP_POLICY (
       policy_name  => 'HR_OLS_POL',
       drop_column  => TRUE);
   END;
   /
   ```

3. Connect as user who has privileges to drop roles and user accounts.

   For example:

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

4. Drop the `HR_ROLE` role.

   ```
   DROP ROLE HR_ROLE;
   ```

5. Drop the `ineau` and `smavris` accounts.

   ```
   DROP USER INEAU;
   DROP USER SMAVRIS;
   ```

Related Topics

- Tutorial: Configuring Compartments in Oracle Label Security
  This tutorial demonstrates how to create Oracle Label Security compartments.

8

Tutorial: Configuring Compartments in Oracle Label Security

This tutorial demonstrates how to create Oracle Label Security compartments.

- About This Tutorial
  In this tutorial, you will use the HR schema to learn how to use Oracle Label Security compartments.
- Step 1: Create an Account for Lily Leagull
  Lily Leagull will initially have the same privileges as Susan Mavris and Ida Neau.
- Step 2: Authorize Lily Leagull for the HIGHLY_SENSITIVE Level
  After the lleagull account has been created, you can authorize it to use the HIGHLY_SENSITIVE level.
- Step 3: Create Two Compartments for the Oracle Label Security Policy
  All three users (Susan Mavris, Ida Neau, and Lily Leagull) will use compartments to access their data.
- Step 4: Create the Data Labels for the Compartments
  You will create three data labels for the compartments.
- Step 5: Assign the Labels to the Users
  Assigning the labels to the users will designate the rows to which these users will have access.
- Step 6: Add the Policy Labels to the HR.EMPLOYEES Table Data
  The HR user will add the policy labels to the HR.EMPLOYEES table data in the EMPLOYEE_ID column.
- Step 7: Test the Oracle Label Security Policy
  To test the policy, each user will try to query the HR.EMPLOYEES table.
- Step 8: Optionally, Remove the Oracle Label Security Policy Components
  You can remove the Oracle Label Security policy, HR_ROLE role, and users Ida Neau, Susan Mavris, and Lily Leagull.

8.1 About This Tutorial

In this tutorial, you will use the HR schema to learn how to use Oracle Label Security compartments.

This tutorial builds on the previous tutorial, which demonstrates how to create Oracle Label Security levels to control the access that two users, Susan Mavris and Ida Neau, have to the records in the HR.EMPLOYEES schema. For this tutorial, a third user, Lily Leagull, is an attorney with the company's legal department. Two former employees are suing the company, and Lily must have access to their records. This user must not have access to any other records. The access to the former users is set by the HIGHLY_SENSITIVE level, which you created in the previous tutorial. Access to the records of the two suing former employees will be possible through the use of a compartment within the HIGHLY_SENSITIVE data set, called LEGAL.

Related Topics

- Tutorial: Configuring Levels in Oracle Label Security
  This tutorial demonstrates how to create Oracle Label Security levels.

8.2 Step 1: Create an Account for Lily Leagull

Lily Leagull will initially have the same privileges as Susan Mavris and Ida Neau.

1. Log in to a PDB as a user who has privileges to create user accounts and grant them roles.

   For example:

   ```
   sqlplus sec_admin@pdb_name
   Enter password: password
   ```

   To find the available PDBs, 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 account for Lily Leagull and grant this user the `HR_ROLE` role as follows:

   ```
   GRANT CONNECT, HR_ROLE TO LLEAGULL IDENTIFIED BY password;
   ```

8.3 Step 2: Authorize Lily Leagull for the HIGHLY_SENSITIVE Level

After the `lleagull` account has been created, you can authorize it to use the `HIGHLY_SENSITIVE` level.

1. Connect as a user who can create and manage Oracle Label Security policies.

   For example:

   ```
   sqlplus psmith_ols@pdb_name
   Enter password: password
   ```

2. Authorize `lleagull` to use the `HIGHLY_SENSITIVE` level.

   The short name for `HIGHLY_SENSITIVE` is `HS`.

   ```
   BEGIN
      SA_USER_ADMIN.SET_LEVELS (
         policy_name  => 'HR_OLS_POL',
         user_name    => 'LLEAGULL',
         max_level    => 'HS',
         min_level    => 'S');
   END;
   /
   ```

8.4 Step 3: Create Two Compartments for the Oracle Label Security Policy

All three users (Susan Mavris, Ida Neau, and Lily Leagull) will use compartments to access their data.

The two HR employees, Susan Mavris and Ida Neau, will use the `HR` compartment. The Legal department employee, Lily Leagull, will use the `LEGAL` (`LEG`) compartment.

1. If necessary, connect as a user who can create and manage Oracle Label Security policies.

   For example:

   ```
   sqlplus psmith_ols@pdb_name
   Enter password: password
   ```

2. Create the compartments as follows:

   ```
   BEGIN
     SA_COMPONENTS.CREATE_COMPARTMENT (
       policy_name      => 'HR_OLS_POL',
       long_name        => 'HR',
       short_name       => 'HR',
       comp_num         =>  1000);

     SA_COMPONENTS.CREATE_COMPARTMENT (
       policy_name      => 'HR_OLS_POL',
       long_name        => 'LEGAL',
       short_name       => 'LEG',
       comp_num         =>  2000);
   END;
   /
   ```

   In this specification, the `comp_num` does not denote hierarchy as the `level_num` setting does with levels. It is only used to help identify the compartment.

8.5 Step 4: Create the Data Labels for the Compartments

You will create three data labels for the compartments.

In this procedure, the data labels will designate the rows that users Susan Mavris, Ida Neau, and Lily Leagull will see in the `HR.EMPLOYEES` table.

1. If necessary, connect as a user who can create and manage Oracle Label Security policies.

   For example:

   ```
   sqlplus psmith_ols@pdb_name
   Enter password: password
   ```

2. Create the data labels as follows:

   ```
   BEGIN
      SA_LABEL_ADMIN.CREATE_LABEL (
         policy_name  => 'HR_OLS_POL',
         label_tag    => 1100,
         label_value  => 'S:HR:', -- SENSITIVE level for the HR compartment
   ```

```
      data_label   => TRUE);

   SA_LABEL_ADMIN.CREATE_LABEL (
      policy_name  => 'HR_OLS_POL',
      label_tag    => 1200,
      label_value  => 'HS:HR:', -- HIGHLY_SENSITIVE level for the HR
compartment
      data_label   => TRUE);

   SA_LABEL_ADMIN.CREATE_LABEL (
      policy_name  => 'HR_OLS_POL',
      label_tag    => 1300,
      label_value  => 'HS:LEG:', --HIGHLY_SENSITIVE level for the LEG
compartment
      data_label   => TRUE);

END;
/
```

In this specification:

- `label_value` `S:HR` will be assigned to the records of all current employees.
- `label_value` `HS:HR` will be assigned to the records all current and former employees.
- `label_value` `HS:LEG` will be assigned to the records of former employees who are suing the company.

8.6 Step 5: Assign the Labels to the Users

Assigning the labels to the users will designate the rows to which these users will have access.

1. If necessary, connect as a user who can create and manage Oracle Label Security policies.

   For example:

   ```
   sqlplus psmith_ols@pdb_name
   Enter password: password
   ```

2. Assign the labels to the users as follows:

   ```
   BEGIN
      SA_USER_ADMIN.SET_USER_LABELS (
         policy_name    => 'HR_OLS_POL',
         user_name      => 'ineau',
         max_read_label => 'S:HR:');

      SA_USER_ADMIN.SET_USER_LABELS (
         policy_name    => 'HR_OLS_POL',
         user_name      => 'smavris',
         max_read_label => 'HS:HR,LEG:');

      SA_USER_ADMIN.SET_USER_LABELS (
         policy_name    => 'HR_OLS_POL',
         user_name      => 'lleagull',
         max_read_label => 'HS:LEG:');
   END;
   /
   ```

In this specification:

- User `ineau` (Ida Neau), who is authorized for the `HR` compartment, will continue to have access to all current employees, but not the former or suing employees.
- User `smavris` (Susan Mavris), who is authorized for both the `HR` and `LEG` compartments, will continue to have access to all current and former employees, and former employees who are suing the company.
- User `lleagul` (Lily Leagul), who is authorized for the `LEG` compartment, will have access only to former employees who are suing the company.

8.7 Step 6: Add the Policy Labels to the HR.EMPLOYEES Table Data

The `HR` user will add the policy labels to the `HR.EMPLOYEES` table data in the `EMPLOYEE_ID` column.

1. Connect as the `HR` user.

   ```
   connect hr@pdb_name
   Enter password: password
   ```

2. Perform the following `UPDATE` statement to apply the `SENSITIVE` level and the `HR` compartment to the employee IDs of users who are still currently employed with the company.

   This `UPDATE` statement controls the access that Ida Neau will have to the `HR.EMPLOYEES` table because Ida is authorized for the `SENSITIVE` level and the `HR` compartment.

   ```
   UPDATE employees
   SET    ols_col = CHAR_TO_LABEL('HR_OLS_POL','S:HR')
   WHERE  UPPER(employee_id) NOT IN (200, 101, 102, 176, 201, 122, 114);
   ```

3. Perform the following `UPDATE` statement to apply the `HIGHLY_SENSITIVE` level and `HR` compartment to the employee IDs of current and former employees.

   This `UPDATE` statement controls the access that Susan Mavris will have to the `HR.EMPLOYEES` table because Susan is authorized for the `SENSITIVE` level and the `HR` and `LEG` compartments.

   ```
   UPDATE employees
   SET    ols_col = CHAR_TO_LABEL('HR_OLS_POL','HS:HR,LEG')
   WHERE  UPPER(employee_id) IN (200, 101, 102, 176, 201, 122, 114);
   ```

4. Perform the following `UPDATE` statement to apply the `HIGHLY_SENSITIVE` level and `LEG` compartment to the employee IDs of former employees who are suing the company.

   This `UPDATE` statement controls the access that Lily Leagull will have to the `HR.EMPLOYEES` table because Lily is authorized for the `HIGHLY_SENSITIVE` level and the `LEG` compartment.

   ```
   UPDATE employees
   SET    ols_col = CHAR_TO_LABEL('HR_OLS_POL','HS:LEG')
   WHERE  UPPER(employee_id) IN (200, 101);
   ```

8.8 Step 7: Test the Oracle Label Security Policy

To test the policy, each user will try to query the `HR.EMPLOYEES` table.

1. Connect as user Ida Neau.

   ```
   connect ineau@pdb_name
   Enter password: password
   ```

2. Set column widths for the table output.

   ```
   column first_name format a25
   column last_name format a25
   column ols_label format a10
   ```

3. Run the following query:

   ```
   SELECT FIRST_NAME, LAST_NAME, EMPLOYEE_ID,
   LABEL_TO_CHAR(OLS_COL) OLS_LABEL
   FROM HR.EMPLOYEES
   ORDER BY OLS_COL;
   ```

   The output should be similar to the following:

   ```
   FIRST_NAME                LAST_NAME                 EMPLOYEE_ID OLS_LABEL
   ------------------------- ------------------------- ----------- ----------
   Steven                    King                              100 S:HR
   Alexander                 Hunold                            103 S:HR
   Bruce                     Ernst                             104 S:HR
   David                     Austin                            105 S:HR
   Valli                     Pataballa                         106 S:HR
   Diana                     Lorentz                           107 S:HR
   Nancy                     Greenberg                         108 S:HR
   Daniel                    Faviet                            109 S:HR
   John                      Chen                              110 S:HR
   Ismael                    Sciarra                           111 S:HR
   ...

   100 rows selected
   ```

   Because Ida Neau was assigned the `SENSITIVE` (`S`) label and the `HR` compartment, the output in the column `OLS_LABEL` is `S:HR`. 100 rows are returned.

   Note that the Oracle Label Security policy restriction applies to any `SELECT` query the user makes. For example, if Ida Neau performs a `SELECT COUNT(*) FROM HR.EMPLOYEES;` query, then it would return 100 rows, not the full 107.

4. Connect as user Susan Mavris.

   ```
   connect smavris
   Enter password: password
   ```

5. Run the same query that Ida Neau ran.

   ```
   SELECT FIRST_NAME, LAST_NAME, EMPLOYEE_ID,
   LABEL_TO_CHAR(OLS_COL) OLS_LABEL
   FROM HR.EMPLOYEES
   ORDER BY OLS_COL;
   ```

   The output should be similar to the following:

   ```
   FIRST_NAME                LAST_NAME                 EMPLOYEE_ID OLS_LABEL
   ------------------------- ------------------------- ----------- ----------
   Steven                    King                              100 S:HR
   Alexander                 Hunold                            103 S:HR
   Bruce                     Ernst                             104 S:HR
   David                     Austin                            105 S:HR
   Valli                     Pataballa                         106 S:HR
   ```

```
...
Jennifer                  Whalen                            200 HS:LEG
Neena                     Kochhar                           101 HS:LEG
Michael                   Hartstein                         201 HS:HR,LEG
Jonathon                  Taylor                            176 HS:HR,LEG
Den                       Raphaely                          114 HS:HR,LEG
Lex                       De Haan                           102 HS:HR,LEG
Payam                     Kaufling                          122 HS:HR,LEG

107 rows selected
```

Because Susan Mavris was assigned the `HIGHLY_SENSITIVE` (`HS`) level with the `HR` and `LEG` compartments, the output in the column `OLS_LABEL` is as follows:

- `S:HR` to capture all current employees
- `HS:HR, LEG` to capture all former employees
- `HS:LEG` to capture former employees who are suing.

107 rows are returned.

6. Connect as user Lily Leagull.

```
connect lleagull
Enter password: password
```

7. Run the same query that Susan Mavris ran.

```
SELECT FIRST_NAME, LAST_NAME, EMPLOYEE_ID,
LABEL_TO_CHAR(OLS_COL) OLS_LABEL
FROM HR.EMPLOYEES
ORDER BY OLS_COL;
```

The output should be similar to the following:

```
FIRST_NAME                LAST_NAME                 EMPLOYEE_ID OLS_LABEL
------------------------- ------------------------- ----------- ----------
Jennifer                  Whalen                            200 HS:LEG
Neena                     Kochhar                           101 HS:LEG

107 rows selected
```

Because Lily Leagull was assigned the `HS` level with the `LEG` compartment, only former users who are suing are returned for this query.

8.9 Step 8: Optionally, Remove the Oracle Label Security Policy Components

You can remove the Oracle Label Security policy, `HR_ROLE` role, and users Ida Neau, Susan Mavris, and Lily Leagull.

1. Connect as a user who can create and manage Oracle Label Security policies.

   For example:

```
sqlplus psmith_ols@pdb_name
Enter password: password
```

2. Drop the Oracle Label Security policy.

This procedure also removes the levels, compartments, and from the `HR.EMPLOYEES` table, the `OLS_COL` column.

```
BEGIN
  SA_SYSDBA.DROP_POLICY (
    policy_name  => 'HR_OLS_POL',
    drop_column  => TRUE);
END;
/
```

3. Connect as user who has privileges to drop roles and user accounts.

   For example:

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

4. Drop the `HR_ROLE` role.

   ```
   DROP ROLE HR_ROLE;
   ```

5. Drop the `ineau`, `smavris`, and `lleagull` accounts.

   ```
   DROP USER INEAU;
   DROP USER SMAVRIS;
   DROP USER LLEAGULL;
   ```

9 Tutorial: Configuring Groups in Oracle Label Security

This tutorial demonstrates how to create an Oracle Label Security parent group that has four child groups.

- About This Tutorial
  In this tutorial, you will use the `OE` schema to learn how to use Oracle Label Security groups.
- Step 1: Create a Role and User Accounts
  The role that you create will enable any user who is granted it to have the `SELECT` privilege on the `OE.CUSTOMERS` table. The user accounts are for four sales representatives and the
- Step 2: Create the Oracle Label Security Policy Container
  As an Oracle Label Security administrator, you must create and then enable the policy container.
- Step 3: Create and Authorize a Level Component for the Oracle Label Security Policy
  After you create the Oracle Label Security policy container, you are ready to create and authorize a level component.
- Step 4: Create and Authorize Groups for the Oracle Label Security Policy
  You will create and authorize one parent group and four child groups for this parent group. Each user will be authorized for a group.
- Step 5: Apply and Authorize the Policy to the Table
  You must apply the `OE_OLS_POL` policy to the `OE.CUSTOMERS` table and then authorize the `OE` schema user to have read privileges for the policy.
- Step 6: Add the Policy Labels to the OE.CUSTOMERS Table Data
  The `OE` user will add the policy labels to the `OE.CUSTOMERS` table data in the `ACCOUNT_MGR_ID` column.
- Step 7: Test the Oracle Label Security Policy
  To test the policy, each user will query the `OE.CUSTOMERS` table.
- Step 8: Optionally, Remove the Oracle Label Security Policy Components
  You can remove the Oracle Label Security policy, `OE_CUST` role, and the user accounts.

9.1 About This Tutorial

In this tutorial, you will use the `OE` schema to learn how to use Oracle Label Security groups.

Each sales manager must have access to the records of their customers in the `OE.CUSTOMERS` table. The company president of advertising, Steven King, who each sales manager reports to, must have access to all customer records. The customer records are divided into groups based on the sales managers' territories.

The Oracle Label Security policy that you create will assign each of the sales managers a group, and this group will be used to label the appropriate rows in the `OE.CUSTOMERS` table. The groups will have a parent group, `GLOBAL_SALES`, which will be associated with advertising president Steven King. The child groups of `GLOBAL_SALES` are as follows:

- `EUROPE`, with access by sales manager Alberto Errazuriz
- `ASIA`, with access by sales manager Gerald Cambrault
- `UNITED_STATES_1`, with access by sales manager John Russell
- `UNITED_STATES_2`, with access by sales manager Eleni Zlotkey

By default, the `OE` schema is not installed. You can download this schema from GitHub, as explained in *Oracle Database Sample Schemas*.

Related Topics

- *Oracle Database Sample Schemas*

9.2 Step 1: Create a Role and User Accounts

The role that you create will enable any user who is granted it to have the `SELECT` privilege on the `OE.CUSTOMERS` table. The user accounts are for four sales representatives and the

1. Log in to a PDB as a user who has privileges to create roles, grant privileges, and create user accounts.

   For example:

   ```
   sqlplus sec_admin@pdb_name
   Enter password: password
   ```

   To find the available PDBs, query the `PDB_NAME` column of the `DBA_PDBS` data dictionary view. To check the current container, run the `show con_name` command.

2. Ensure that the `OE` schema has been downloaded from GitHub and installed.

   Oracle Database Sample Schemas explains how to download and install this schema.

3. Create the role as follows:

   ```
   CREATE ROLE OE_CUST;
   ```

4. Grant the `SELECT` privilege on `OE.CUSTOMERS` to `OE_CUST`.

   ```
   GRANT SELECT ON OE.CUSTOMERS TO OE_CUST;
   ```

5. Create the user accounts and grant them the `OE_CUST` role.

   ```
   GRANT CONNECT, OE_CUST TO SKING IDENTIFIED BY password; --For Steven King,
   president
   GRANT CONNECT, OE_CUST TO AERRAZURIZ IDENTIFIED BY password; --For Alberto
   Errazuriz, sales manager
   GRANT CONNECT, OE_CUST TO GCAMBRAULT IDENTIFIED BY password; --For Gerald
   Cambrault, sales manager
   GRANT CONNECT, OE_CUST TO JRUSSELL IDENTIFIED BY password; --For John
   Russell, sales manager
   GRANT CONNECT, OE_CUST TO EZLOTKEY IDENTIFIED BY password; --For Eleni
   Zlotkey, sales manager
   ```

Related Topics

- *Oracle Database Sample Schemas*

9.3 Step 2: Create the Oracle Label Security Policy Container

As an Oracle Label Security administrator, you must create and then enable the policy container.

1. Connect as a user who can create and manage Oracle Label Security policies.

   For example:

   ```
   sqlplus psmith_ols@pdb_name
   Enter password: password
   ```

2. Create the policy.

   ```
   BEGIN
    SA_SYSDBA.CREATE_POLICY (
     policy_name      => 'OE_OLS_POL',
     column_name      => 'OLS_COL');
   END;
   /
   ```

   In this specification, the `default_options` parameter is omitted because you can add it later on in another procedure.

   After you create this policy container, Oracle Label Security creates and grants a role to you named `OE_OLS_POL_DBA`, which enables you to manage this policy during its lifetime.

3. Enable the policy.

   ```
   EXEC SA_SYSDBA.ENABLE_POLICY ('OE_OLS_POL');
   ```

9.4 Step 3: Create and Authorize a Level Component for the Oracle Label Security Policy

After you create the Oracle Label Security policy container, you are ready to create and authorize a level component.

Levels are used for this policy but you do need to have a default level in order for the data labels that will be created later on to work.

1. If necessary, connect as a user who can create and manage Oracle Label Security policies.

   For example:

   ```
   sqlplus psmith_ols@pdb_name
   Enter password: password
   ```

2. Create the levels as follows:

   ```
   BEGIN
      SA_COMPONENTS.CREATE_LEVEL (
         policy_name => 'OE_OLS_POL',
         level_num   => 50,
         short_name  => 'D',
         long_name   => 'DEFAULT');
   ```

```
END;
/
```

In this specification:

- `policy_name` associates the levels with the policy container that you just created.
- `level_num` determines how much access a user can have. Level number `50` enables a user to have access to this level and any level number below it. However, this tutorial only uses one level.
- `short_name` is a short-hand name for the `long_name` of the level, and will be used in other procedures to refer to the `long_name` version of the level.

3. Authorize the level for the five users who will access the `OE.EMPLOYEES` table.

```
BEGIN
   SA_USER_ADMIN.SET_LEVELS (
      policy_name  => 'OE_OLS_POL',
      user_name    => 'SKING',
      max_level    => 'D');

   SA_USER_ADMIN.SET_LEVELS (
      policy_name  => 'OE_OLS_POL',
      user_name    => 'AERRAZURIZ',
      max_level    => 'D');

   SA_USER_ADMIN.SET_LEVELS (
      policy_name  => 'OE_OLS_POL',
      user_name    => 'GCAMBRAULT',
      max_level    => 'D');

   SA_USER_ADMIN.SET_LEVELS (
      policy_name  => 'OE_OLS_POL',
      user_name    => 'JRUSSELL',
      max_level    => 'D');

   SA_USER_ADMIN.SET_LEVELS (
      policy_name  => 'OE_OLS_POL',
      user_name    => 'EZLOTKEY',
      max_level    => 'D');
END;
/
```

9.5 Step 4: Create and Authorize Groups for the Oracle Label Security Policy

You will create and authorize one parent group and four child groups for this parent group. Each user will be authorized for a group.

1. If necessary, connect as a user who can create and manage Oracle Label Security policies.

   For example:

```
sqlplus psmith_ols@pdb_name
Enter password: password
```

2. Create the `GLOBAL_SALES` parent group.

```
BEGIN
  SA_COMPONENTS.CREATE_GROUP (
   policy_name     => 'OE_OLS_POL',
   group_num       => 2000,
   short_name      => 'GS',
   long_name       => 'GLOBAL_SALES');
END;
/
```

In this specification, `group_num` is used for identification purposes only. It does not control any hierarchy in this set of groups.

3. Create the child groups.

   Any user who is authorized for the parent group, `GLOBAL_SALES` (`GS`), will have authorization for these child groups as well.

```
BEGIN
  SA_COMPONENTS.CREATE_GROUP (
   policy_name     => 'OE_OLS_POL',
   group_num       => 2100,
   short_name      => 'EU',
   long_name       => 'EUROPE',
   parent_name     => 'GS');

  SA_COMPONENTS.CREATE_GROUP (
   policy_name     => 'OE_OLS_POL',
   group_num       => 2200,
   short_name      => 'AS',
   long_name       => 'ASIA',
   parent_name     => 'GS');

  SA_COMPONENTS.CREATE_GROUP (
   policy_name     => 'OE_OLS_POL',
   group_num       => 2300,
   short_name      => 'US1',
   long_name       => 'UNITED_STATES_1',
   parent_name     => 'GS');

  SA_COMPONENTS.CREATE_GROUP (
   policy_name     => 'OE_OLS_POL',
   group_num       => 2400,
   short_name      => 'US2',
   long_name       => 'UNITED_STATES_2',
   parent_name     => 'GS');
END;
/
```

   In this specification, the `parent_name` parameter designates the parent group, `GS`.

4. Authorize the users that you created earlier for these groups.

```
BEGIN
 SA_USER_ADMIN.SET_GROUPS (
  policy_name    => 'OE_OLS_POL',
  user_name      => 'SKING',
  read_groups    => 'GS');

 SA_USER_ADMIN.SET_GROUPS (
  policy_name    => 'OE_OLS_POL',
  user_name      => 'AERRAZURIZ',
  read_groups    => 'EU');
 SA_USER_ADMIN.SET_GROUPS (
  policy_name    => 'OE_OLS_POL',
  user_name      => 'GCAMBRAULT',
  read_groups    => 'AS');

 SA_USER_ADMIN.SET_GROUPS (
  policy_name    => 'OE_OLS_POL',
  user_name      => 'JRUSSELL',
  read_groups    => 'US1');

 SA_USER_ADMIN.SET_GROUPS (
  policy_name    => 'OE_OLS_POL',
  user_name      => 'EZLOTKEY',
  read_groups    => 'US2');
END;
/
```

In this specification, user `SKING` is authorized for the parent group, `GS`, and the remaining users, who are all sales managers, are authorized for the groups that represent their sales territories.

After you set the authorization for these groups, and because you have not yet created data labels, Oracle Label Security automatically creates the data labels for you. In earlier tutorials, you learned how to manually create the data labels, but for this tutorial, you get to allow Oracle Label Security to create them for you. You can see the labels by querying the `DBA_SA_LABELS` data dictionary view. For example:

```
SELECT POLICY_NAME, LABEL, LABEL_TAG FROM DBA_SA_LABELS ORDER BY LABEL_TAG;
```

Output similar to the following appears:

```
POLICY_NAME      LABEL      LABEL_TAG
------------     ------     -----------
OE_OLS_POL       D          1000000085
OE_OLS_POL       D::GS      1000000086
OE_OLS_POL       D::EU      1000000087
OE_OLS_POL       D::AS      1000000088
OE_OLS_POL       D::US1     1000000089
OE_OLS_POL       D::US2     1000000090
```

9.6 Step 5: Apply and Authorize the Policy to the Table

You must apply the `OE_OLS_POL` policy to the `OE.CUSTOMERS` table and then authorize the `OE` schema user to have read privileges for the policy.

1. If necessary, connect as a user who can create and manage Oracle Label Security policies.

   For example:

   ```
   sqlplus psmith_ols@pdb_name
   Enter password: password
   ```

2. Apply the `OE_OLS_POL` policy to the `OE.CUSTOMERS` table.

   ```
   BEGIN
     SA_POLICY_ADMIN.APPLY_TABLE_POLICY (
       policy_name    => 'OE_OLS_POL',
       schema_name    => 'OE',
       table_name     => 'CUSTOMERS',
   ```

```
    table_options  => 'READ_CONTROL');
END;
/
```

Earlier, when you created the policy with the `SA_SYSDBA.CREATE_POLICY` procedure, you did not set the `default_options` parameter, which defines the policy enforcement options. Therefore, you must set the policy enforcement here, with the `table_options` parameter of `SA_POLICY_ADMIN.APPLY_TABLE_POLICY`. `READ_CONTROL` enforces the OLS policy during the `SELECT` statement processing that the users will perform later. (It also applies to `UPDATE` and `DELETE` statement processing.)

3. Enable the `OE_OLS_POL` policy for `OE.CUSTOMERS`.

```
BEGIN
   SA_POLICY_ADMIN.ENABLE_TABLE_POLICY (
      policy_name => 'OE_OLS_POL',
      schema_name => 'OE',
      table_name  => 'CUSTOMERS');
END;
/
```

4. Set user privileges for `OE` so that `OE` can apply the labels to the `OE.CUSTOMERS` data rows.

```
BEGIN
   SA_USER_ADMIN.SET_USER_PRIVS (
      policy_name => 'OE_OLS_POL',
      user_name   => 'OE',
      privileges  => 'READ');
END;
/
```

9.7 Step 6: Add the Policy Labels to the OE.CUSTOMERS Table Data

The `OE` user will add the policy labels to the `OE.CUSTOMERS` table data in the `ACCOUNT_MGR_ID` column.

1. Connect as the `OE` user.

```
connect OE@pdb_name
Enter password: password
```

2. Perform the following `UPDATE` statement to apply the `GLOBAL_SALES` (`GS`) group to `OE.CUSTOMERS`.

```
UPDATE customers
SET    ols_col = CHAR_TO_LABEL('OE_OLS_POL','D::GS')
WHERE  UPPER(account_mgr_id) IN (145, 147, 148, 149);
```

In this specification, the user who is authorized for the label identifier `D::GS` (Steven King) will have access to the rows that are available to users whose `account_mgr_id` IDs are `145`, `147`, `148`, and `149`.

3. Perform the following `UPDATE` statements for the sales managers.

For European sales manager Alberto Errazuriz, whose ID is `147`:

```
UPDATE customers
SET    ols_col = CHAR_TO_LABEL('OE_OLS_POL','D::EU')
WHERE  UPPER(account_mgr_id) = 147;
```

For Asian sales manager Gerald Cambrault, whose ID is `148`:

```
UPDATE customers
SET    ols_col = CHAR_TO_LABEL('OE_OLS_POL','D::AS')
WHERE  UPPER(account_mgr_id) = 148;
```

For US sales manager John Russell, whose ID is `145`:

```
UPDATE customers
SET    ols_col = CHAR_TO_LABEL('OE_OLS_POL','D::US1')
WHERE  UPPER(account_mgr_id) = 145;
```

For US sales manager Elena Zlotkey, whose ID is `149`:

```
UPDATE customers
SET    ols_col = CHAR_TO_LABEL('OE_OLS_POL','D::US2')
WHERE  UPPER(account_mgr_id) = 149;
```

9.8 Step 7: Test the Oracle Label Security Policy

To test the policy, each user will query the `OE.CUSTOMERS` table.

1. Connect as user Alberto Errazuriz.

   ```
   connect aerrazuriz@pdb_name
   Enter password: password
   ```

2. Set column widths for the table output.

   ```
   column cust_first_name format a25
   column cust_last_name format a25
   column ols_label format a10
   ```

3. Run the following query:

   ```
   SELECT CUST_FIRST_NAME, CUST_LAST_NAME, ACCOUNT_MGR_ID,
   LABEL_TO_CHAR(OLS_COL) OLS_LABEL
   FROM OE.CUSTOMERS
   ORDER BY OLS_COL;
   ```

   The output should be similar to the following:

   ```
   CUST_FIRST_NAME           CUST_LAST_NAME            ACCOUNT_MGR_ID OLS_LABEL
   ------------------------- ------------------------- -------------- ----------
   Hal                       Olin                                 147 D::EU
   Hannah                    Kanth                                147 D::EU
   Hannah                    Field                                147 D::EU
   Margret                   Powell                               147 D::EU
   Harry Mean                Taylor                               147 D::EU
   Margrit                   Garner                               147 D::EU
   Maria                     Warden                               147 D::EU
   Marilou                   Landis                               147 D::EU
   ...
   76 rows selected.
   ```

   Because Alberto Errazuriz is assigned the `D` level with the `EU` group, the output is `D:EU`.

4. Repeat this query for the other sales managers:

   - Asian sales manager Gerald Cambrault (`gcambrault`), whose output in the `OLS_LABEL` column should be `D:AS`, with 58 rows returned.

- US sales manager John Russell (`jrussell`), whose output in the `OLS_LABEL` column should be `D:US1`, with 111 rows returned.
- Elena Zlotkey (`ezlotkey`), whose output in the `OLS_LABEL` column should be `D:US2`, with 74 rows returned.

5. Connect as president Steven King.

```
connect sking@pdb_name
Enter password: password
```

6. Run the query.

```
SELECT CUST_FIRST_NAME, CUST_LAST_NAME, ACCOUNT_MGR_ID,
LABEL_TO_CHAR(OLS_COL) OLS_LABEL
FROM OE.CUSTOMERS
ORDER BY OLS_COL;
```

The output should be similar to the following:

```
CUST_FIRST_NAME           CUST_LAST_NAME            ACCOUNT_MGR_ID OLS_LABEL
------------------------- ------------------------- -------------- ----------
Kelly                     Lange                                147 D::EU
Kenneth                   Redford                              147 D::EU
Rick                      Lyon                                 147 D::EU
Mammutti                  Sutherland                           147 D::EU
Margaret                  Ustinov                              147 D::EU
Kevin                     Cleveland                            147 D::EU
Klaus Maria               Russell                              147 D::EU
Kris                      de Niro                              147 D::EU
Alain                     Barkin                               147 D::EU
Albert                    Dutt                                 147 D::EU
Amanda                    Finney                               147 D::EU

...

Dom                       McQueen                              149 D::US2
Dom                       Hoskins                              149 D::US2
Don                       Siegel                               149 D::US2
Gvtz                      Bradford                             149 D::US2
Holly                     Kurosawa                             149 D::US2
Rob                       MacLaine                             149 D::US2
Don                       Barkin                               149 D::US2
Meg                       Sen                                  149 D::US2
...
319 rows selected.
```

Because Steven King is assigned the `GS` parent group, the output in the `OLS_LABEL` column is includes all four child groups: `D::EU`, `D::AS`, `D::US1`, and `D::US2`.

9.9 Step 8: Optionally, Remove the Oracle Label Security Policy Components

You can remove the Oracle Label Security policy, `OE_CUST` role, and the user accounts.

1. Connect as a user who can create and manage Oracle Label Security policies.

   For example:

```
sqlplus psmith_ols@pdb_name
Enter password: password
```

2. Drop the Oracle Label Security policy.

   This procedure also removes the levels, groups, and from the `OE.CUSTOMERS` table, the `OLS_COL` column.

   ```
   BEGIN
     SA_SYSDBA.DROP_POLICY (
       policy_name  => 'OE_OLS_POL',
       drop_column  => TRUE);
   END;
   /
   ```

3. Connect as user who has privileges to drop roles and user accounts.

   For example:

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

4. Drop the `OE_CUST` role.

   ```
   DROP ROLE OE_CUST;
   ```

5. Drop the user accounts.

   ```
   DROP USER SKING;
   DROP USER AERRAZURIZ;
   DROP USER GCAMBRAULT;
   DROP USER JRUSSELL;
   DROP USER EZLOTKEY;
   ```

Part IV
Administering an Oracle Label Security Application

Part IV describes how to administer an Oracle Label Security application.

- Implementing Policy Enforcement Options and Labeling Functions
  You can customize the enforcement of Oracle Label Security policies and implement labeling functions.
- Administering and Using Trusted Stored Program Units
  You can use trusted stored program units to enhance system security.
- Using Oracle Label Security with a Distributed Database
  You should understand the special considerations for using Oracle Label Security in a distributed configuration.
- Performing DBA Functions Under Oracle Label Security
  Oracle Label Security supports the standard Oracle Database utilities, but certain restrictions apply, which may require extra steps to get the expected results.
- Releasability Using Inverse Groups
  Oracle Label Security can implement the releasability using inverse groups.
- Auditing Oracle Label Security
  All activities in Oracle Label Security can be audited, including Oracle Label Security administrator activities.

10 Implementing Policy Enforcement Options and Labeling Functions

You can customize the enforcement of Oracle Label Security policies and implement labeling functions.

- Using the LBAC_TRIGGER Schema
  Oracle Label Security stores Oracle Label Security triggers in the `LBAC_TRIGGER` schema.
- Oracle Label Security Policy Enforcement Options
  Oracle Label Security provides a set of policy enforcement options.
- Labeling Functions
  Labeling functions can compute and return a label using resources such as context variables (for example, date or username) and data values.
- Inserting Labeled Data Using Policy Options and Labeling Functions
  It is important to understand how enforcement options and labeling functions affect the insertion of labeled data.
- Inserts of Rows into Foreign Key Tables That Do Not Exist Yet in Referential Tables
  If you insert a row into a foreign key table that does not yet exist in the referenced table with a primary key, then an `ORA-28117: integrity constraint violated - parent record not found` error occurs when you run the statement to perform the insert, not at `COMMIT` time.
- Updating Labeled Data Using Policy Options and Labeling Functions
  Users must be authorized to change rows that are protected by Oracle Label Security.
- Deletion of Labeled Data Using Policy Options and Labeling Functions
  You can delete labeled data.
- SQL Predicates with an Oracle Label Security Policy
  You can use a SQL predicate to provide extensibility for selective enforcement of data access rules.

10.1 Using the LBAC_TRIGGER Schema

Oracle Label Security stores Oracle Label Security triggers in the `LBAC_TRIGGER` schema.

- About Using the LBAC_TRIGGER Schema
  The `LBAC_TRIGGER` schema stores internal triggers that were previously in the `LBACSYS` schema.
- Migrating Existing LBACSYS Triggers to the LBAC_TRIGGER Schema
  If there are DML triggers in the `LBACSYS` schema, then you must migrate them to the `LBAC_TRIGGER` schema
- Downgrading to an Oracle Database Release Earlier than Release 23ai
  If you must downgrade to a pre-Oracle Database 23ai release, then the downgrade will fail if the `LBAC_TRIGGER` schema has triggers.

10.1.1 About Using the LBAC_TRIGGER Schema

The `LBAC_TRIGGER` schema stores internal triggers that were previously in the `LBACSYS` schema.

When you create a custom label function, Oracle Label Security creates internal triggers and invokes these triggers before or after an insert or an update of a row. This action generates the new label that is written into the label column of the table. In previous releases, these internal triggers were created in the common `LBACSYS` schema. Starting in Oracle Database release 23ai, these triggers are created in the local (PDB) `LBAC_TRIGGER` schema and are also dictionary protected. Therefore, the `LBAC_TRIGGER` schema stores the DML triggers for the following operations: before `INSERT`, after `INSERT`, before `UPDATE`, and after `UPDATE`.

When you upgrade to Oracle Database release 23ai or later, the internal triggers in the `LBACSYS` from previous releases remain there until you migrate them to the `LBAC_TRIGGER` schema. Be aware that if you must downgrade the Oracle database to a release earlier than release 23ai, then you must move any triggers that are in the `LBAC_TRIGGER` schema to another schema, such as `LBACSYS`.

If you do not migrate the triggers, then they will continue to work in the `LBACSYS` schema. However, Oracle strongly recommends that you migrate them to the `LBAC_TRIGGER` schema so that your Oracle Label Security policies will benefit by having stronger security and meeting future compatibility requirements.

10.1.2 Migrating Existing LBACSYS Triggers to the LBAC_TRIGGER Schema

If there are DML triggers in the `LBACSYS` schema, then you must migrate them to the `LBAC_TRIGGER` schema

1. Log in to the PDB as a user who can create and manage Oracle Label Security policies.
2. Query the `ALL_SA_TABLE_POLICIES` data dictionary view.

   `ALL_SA_TABLE_POLICIES` lists the policy, schema, table, and function that are associated with the triggers that you need to migrate. You will need this information for later steps in this procedure.
3. Disable or remove the table policy for the affected tables.

   For example, to disable a table policy:

   ```
   BEGIN
    SA_POLICY_ADMIN.DISABLE_SCHEMA_POLICY(
     policy_name    => 'hr_ols_pol',
     schema_name    => 'HR');
   END;
   /
   ```

To remove a table policy:

```
BEGIN
 SA_POLICY_ADMIN.REMOVE_SCHEMA_POLICY(
  policy_name    => 'ols_admin_pol',
  schema_name    => 'HR',
  drop_column    => TRUE);
END;
/
```

4. Revoke the `EXECUTE` privilege for label functions that are used on the `LBACSYS` schema.

   For example:

   ```
   REVOKE EXECUTE ON hs FROM LBACSYS;
   ```

5. Grant the `EXECUTE` privilege to the label functions that are used for the `LBAC_TRIGGER` schema.

   For example:

   ```
   GRANT EXECUTE ON hs TO LBAC_TRIGGER;
   ```

6. Enable or apply the table policy for the affected tables.

   For example, to enable a table policy:

   ```
   BEGIN
    SA_POLICY_ADMIN.ENABLE_TABLE_POLICY(
     policy_name    => 'hr_ols_pol',
     schema_name    => 'HR',
     table_name     => 'EMPLOYEES');
   END;
   /
   ```

   To apply a table policy:

   ```
   BEGIN
    SA_POLICY_ADMIN.APPLY_TABLE_POLICY(
     policy_name    => 'ols_admin_pol',
     schema_name    => 'HR',
     table_name     => 'EMPLOYEES',
     table_options  => NULL,
     label_function => 'hs(:new.dept,:new.status)',
     predicate      => 'no_control');
   END;
   /
   ```

10.1.3 Downgrading to an Oracle Database Release Earlier than Release 23ai

If you must downgrade to a pre-Oracle Database 23ai release, then the downgrade will fail if the `LBAC_TRIGGER` schema has triggers.

1. Log in to the PDB as a user who can create and manage Oracle Label Security policies.
2. Query the `ALL_SA_TABLE_POLICIES` data dictionary view.

   `ALL_SA_TABLE_POLICIES` lists the policy, schema, table, and function that are associated with the triggers that you need to migrate. You will need this information for later steps in this procedure.
3. Disable or remove the table policy for the affected tables.

   For example, to disable a table policy:

   ```
   BEGIN
    SA_POLICY_ADMIN.DISABLE_SCHEMA_POLICY(
     policy_name     => 'hr_ols_pol',
     schema_name     => 'HR');
   END;
   /
   ```

   To remove a table policy:

   ```
   BEGIN
    SA_POLICY_ADMIN.REMOVE_SCHEMA_POLICY(
     policy_name     => 'ols_admin_pol',
     schema_name     => 'HR',
     drop_column     => TRUE);
   END;
   /
   ```
4. Revoke the `EXECUTE` privilege for label functions that are used on the `LBAC_TRIGGER` schema.

   For example:

   ```
   REVOKE EXECUTE ON hs FROM LBAC_TRIGGER;
   ```
5. Perform the downgrade from Oracle Database release 23ai or later.

   After the downgrade is complete, applying the table policy will create the triggers within `LBACSYS` schema.
6. Grant the `EXECUTE` privilege to the label functions that are used for the `LBACSYS` schema.

   For example:

   ```
   GRANT EXECUTE ON hs TO LBACSYS;
   ```
7. Enable or apply the table policy for the affected tables.

   For example, to enable a table policy:

   ```
   BEGIN
    SA_POLICY_ADMIN.ENABLE_TABLE_POLICY(
     policy_name     => 'hr_ols_pol',
     schema_name     => 'HR',
     table_name      => 'EMPLOYEES');
   ```

```
END;
/
```

To apply a table policy:

```
BEGIN
 SA_POLICY_ADMIN.APPLY_TABLE_POLICY(
  policy_name    => 'ols_admin_pol',
  schema_name    => 'HR',
  table_name     => 'EMPLOYEES',
  table_options  => NULL,
  label_function => 'hs(:new.dept,:new.status)',
  predicate      => 'no_control');
END;
/
```

10.2 Oracle Label Security Policy Enforcement Options

Oracle Label Security provides a set of policy enforcement options.

- About Policy Enforcement Options
  Of all the enforcement controls that Oracle Label Security permits, the administrator must choose those that meet the needs of the given application.
- Levels of Policy Enforcement Options
  You can set policy, schema, and table levels of policy enforcement.
- Categories of Policy Enforcement Options
  Oracle Label Security enforces policies using three categories: label management options, access control options, and overriding options.
- Relationships of Policy Enforcement Options
  Oracle Label Security has a set of policy enforcement options.
- How the HIDE Policy Column Option Works
  You can specify the `HIDE` policy configuration option when you add an Oracle Label Security policy column to a table.
- How the Label Management Enforcement Options Work
  The three label enforcement options control the data label written when a row is inserted or updated.
- How the Access Control Enforcement Options Work
  Access control options limit the rows accessible for `SELECT`, `UPDATE`, `INSERT`, or `DELETE` operations to only those rows whose labels meet established policies.
- How the Overriding Enforcement Options Work
  Whereas `ALL_CONTROL` applies all of the label management and access control enforcement options, `NO_CONTROL` applies none of them.
- Guidelines for Using the Policy Enforcement Options
  You can customize policy enforcement for a schema or table through the Oracle Enterprise Manager.
- Exemptions from Oracle Label Security Policy Enforcement
  Oracle Label Security has several exceptions from OLS policy enforcement.

- Data Dictionary Views for Viewing Policy Options on Tables and Schemas
  Oracle Label Security provides data dictionary views that describe the policy enforcement options currently applied to tables and schemas.

10.2.1 About Policy Enforcement Options

Of all the enforcement controls that Oracle Label Security permits, the administrator must choose those that meet the needs of the given application.

This means identifying levels of data sensitivity to exposure, alteration, or misuse, as well as identifying which users have the need or the right to access or alter such data. The policy enforcement options enable administrators to fine-tune users' abilities to read or write data or labels.

10.2.2 Levels of Policy Enforcement Options

You can set policy, schema, and table levels of policy enforcement.

Table 10-1 lists the levels on which policy enforcement options can operate.

Table 10-1 When Policy Enforcement Options Take Effect

Level at which option set	Options set at this level affect user operations ...
Policy lvel	... only when the policy has been applied to the table or schema
Schema lvel	... whenever a user acts in this schema
Table lvel	... whenever a user acts in this table

When you apply a policy to a table or schema, you can specify the enforcement options that are to constrain use of that table or schema. If you do not specify enforcement options at that time, then the default enforcement options you specified when you created that policy are used automatically.

These options customize your policy enforcement to meet your security requirements as to `READ` access, `WRITE` access, and label changes. You can also specify whether the label column should be displayed or hidden. You can choose to enforce some or all of the policy options for any protected table by specifying only those you want.

Optionally, you can assign each table a labeling function, which determines the label of any row inserted or updated in that table. You can also specify, optionally, a *SQL* predicate for a table, to control which rows are accessible to users, based on their labels.

When Oracle Label Security policy enforcement options are applied, they control which rows are accessible to view or to insert, update, or delete.

Related Topics

- Labeling Functions
  Labeling functions can compute and return a label using resources such as context variables (for example, date or username) and data values.
- SQL Predicates with an Oracle Label Security Policy
  You can use a SQL predicate to provide extensibility for selective enforcement of data access rules.

10.2.3 Categories of Policy Enforcement Options

Oracle Label Security enforces policies using three categories: label management options, access control options, and overriding options.

Table 10-2 lists the categories of policy enforcement options.

- Label management options ensure that data labels written for inserted or updated rows do not violate policies set for such labels
- Access control options ensure that only rows whose labels meet established policies are accessible for `SELECT`, `UPDATE`, `INSERT`, or `DELETE` operations.
- Overriding options can suspend or apply all other enforcement options.

Table 10-2 Policy Enforcement Options

Type of Enforcement	Option	Description
How the Label Management Enforcement Options Work	`LABEL_DEFAULT`	Uses the session's default row label value unless the user explicitly specifies a label on `INSERT`.
-	`LABEL_UPDATE`	Applies policy enforcement to `UPDATE` operations that set or change the value of a label attached to a row. The `WRITEUP`, `WRITEDOWN`, and `WRITEACROSS` privileges are enforced only if the `LABEL_UPDATE` option is active.
-	`CHECK_CONTROL`	Applies `READ_CONTROL` policy enforcement to `INSERT` and `UPDATE` statements to assure that the new row label is read-accessible.
How the Access Control Enforcement Options Work	`READ_CONTROL`	Applies policy enforcement to all queries. Only authorized rows are accessible for `SELECT`, `UPDATE`, and `DELETE` operations. See INSERT_CONTROL, UPDATE_CONTROL, and DELETE_CONTROL.
-	`WRITE_CONTROL`	Determines the ability to `INSERT`, `UPDATE`, and `DELETE` data in a row. If this option is active, it enforces `INSERT_CONTROL`, `UPDATE_CONTROL`, and `DELETE_CONTROL`.
-	`INSERT_CONTROL`	Applies policy enforcement to `INSERT` operations, according to the algorithm for write access described in the figure in How Oracle Label Security Algorithm for Read Access Works.
-	`DELETE_CONTROL`	Applies policy enforcement to `DELETE` operations, according to the algorithm for write access described in the figure in How Oracle Label Security Algorithm for Read Access Works.
-	`UPDATE_CONTROL`	Applies policy enforcement to `UPDATE` operations on the data columns within a row, according to the algorithm for write access described in the figure in How Oracle Label Security Algorithm for Read Access Works.
How the Overriding Enforcement Options Work	`ALL_CONTROL`	Applies all enforcement options.

Table 10-2 (Cont.) Policy Enforcement Options

Type of Enforcement	Option	Description
-	`NO_CONTROL`	Applies no enforcement options. A labeling function or a SQL predicate can nonetheless be applied.

Remember that even when Oracle Label Security is applicable to a table, some DML operations may not be covered by the policies being applied. The policy enforcement options set by the administrator determine both the SQL processing behavior and what an authorized user can actually see in response to a query on a protected table. Except where noted, this chapter assumes that `ALL_CONTROL` is active, meaning that all enforcement options are in effect. If users attempt to perform an operation for which they are not authorized, then an error message is raised and the SQL statement fails.

Understanding the relationships among these policy enforcement options, and what SQL statements they control, is essential to their effective use in designing and implementing your Oracle Label Security policies.

Related Topics

- Implementation of Inverse Groups with INVERSE_GROUP Enforcement
  When creating an Oracle Label Security policy, you can specify whether the policy can use inverse group functionality to implement releasability.

10.2.4 Relationships of Policy Enforcement Options

Oracle Label Security has a set of policy enforcement options.

Table 10-3 describes the relationships between policy enforcement options.

Table 10-3 What Policy Enforcement Options Control

Specifying This Option in a Policy	Controls These SQL Operations	Using These Criteria and with These Effects
`READ_CONTROL`	`SELECT`, `UPDATE`, and `DELETE`	Only authorized rows (*) are accessible.
`WRITE_CONTROL`	`INSERT`, `UPDATE`, and `DELETE`	(a) Only authorized rows (**) are accessible (b) Data labels writable unless `LABEL_UPDATE` is active.
`WRITE_CONTROL` (necessary for `INSERT_CONTROL`, `UPDATE_CONTROL`, and `DELETE_CONTROL`)	-	-
`INSERT_CONTROL`	`INSERT`	-
`UPDATE_CONTROL`	`UPDATE`	-
`DELETE_CONTROL`	`DELETE`	-
`CHECK_CONTROL`	-	Applies `READ_CONTROL` policy enforcement to `INSERT` and `UPDATE` statements to assure that the new row label is read-accessible.

Table 10-3 (Cont.) What Policy Enforcement Options Control

Specifying This Option in a Policy	Controls These SQL Operations	Using These Criteria and with These Effects
How the Access Control Enforcement Options Work	-	Applies policy enforcement to all queries. Only authorized rows are accessible for operations.
`INSERT_CONTROL`	`INSERT_CONTROL`	Applies policy enforcement to `INSERT` operations.
`DELETE_CONTROL`	`DELETE_CONTROL`	Applies policy enforcement to `DELETE` operations.
`UPDATE_CONTROL`	`UPDATE_CONTROL`	Applies policy enforcement to `UPDATE` operations on the data columns within a row.
How the Overriding Enforcement Options Work	`ALL_CONTROL`	Applies all enforcement options.
`NO_CONTROL`	`NO_CONTROL`	Applies no enforcement options. A labeling function or a SQL predicate can nonetheless be applied.

(*) A row is authorized for `READ` access if the following three criteria are all met:(user-minimum-level) < = (data-row-level) < = (session-level)(any-data-group) is a child of (any-user-group-or-childgroup) (every-data-compartment) is also in (the user's compartments). Refer to the figure in How Oracle Label Security Algorithm for Read Access Works

(**) A row is authorized for READ access if the following three criteria are all met:(user-minimum-level) < = (data-row-level) < = (session-level)(any-data-group) is a child of (any-user-group-or-childgroup) (every-data-compartment) is also in (the user's compartments). Refer to the figure in How Oracle Label Security Algorithm for Read Access Works.

10.2.5 How the HIDE Policy Column Option Works

You can specify the `HIDE` policy configuration option when you add an Oracle Label Security policy column to a table.

This prevents display of the column containing the policy's labels.

Once the policy has been applied, the hidden (or not hidden) status of the column cannot be changed unless the policy is removed with the `DROP_COLUMN` parameter set to `TRUE`. Then, the policy can be reapplied with a new hidden status.

`INSERT` statements doing all-column inserts do not require the values for hidden label columns.

`SELECT` statements do not automatically return the values of hidden label columns. Such values must be explicitly retrieved.

A `DESCRIBE` on a table may or may not display the label column. If the administrator sets the `HIDE` option, then the label column will not be displayed. If `HIDE` is not specified for a policy, then the label column is displayed in response to a `SELECT`.

Related Topics

- SA_POLICY_ADMIN.APPLY_TABLE_POLICY
  The `SA_POLICY_ADMIN.APPLY_TABLE_POLICY` procedure adds the specified policy to a table.

- Retrieving All Columns from a Table When the Policy Label Column Is Hidden
  If the policy label column is hidden, then it is not automatically returned when you run `SELECT *` on the table.

10.2.6 How the Label Management Enforcement Options Work

The three label enforcement options control the data label written when a row is inserted or updated.

- About the Label Management Enforcement Options
  When a policy specifies the options and is applied to a table or schema, these options apply to special situations.
- LABEL_DEFAULT: Using the Session's Default Row Label
  A user can update a row without specifying a label value, because the updated row uses its original label.
- LABEL_UPDATE: Changing Data Labels
  A user updating a row can normally change its label to any label within their authorized label range.
- CHECK_CONTROL: Checking Data Labels
  If an inserted or updated row gets its label from a labeling function, the label could be outside the user's authorizations.

10.2.6.1 About the Label Management Enforcement Options

When a policy specifies the options and is applied to a table or schema, these options apply to special situations.

A user inserting a row can specify any data label within the range of the user's label authorizations. If the user does not specify a label for the row being written, `LABEL_DEFAULT` can do so. Updates can be restricted by `LABEL_UPDATE`. Inserts or updates that use a labeling function need `CHECK_CONTROL` to prevent assigning a data label outside the user's authorizations. Such a label would prevent the user from accessing the row just written, and could enable the user to make data available inappropriately.

Any labeling function in force on a table overrides these options. Such a function can be named in the call that applies the policy to the table. If the administrator named such a function when applying a policy, but then disables or removes that policy, then that function is no longer applied.

Related Topics

- SA_SYSDBA.DISABLE_POLICY
  The `SA_SYSDBA.DISABLE_POLICY` procedure turns off enforcement of a policy, without removing it from the database.

10.2.6.2 LABEL_DEFAULT: Using the Session's Default Row Label

A user can update a row without specifying a label value, because the updated row uses its original label.

However, to insert a new row, the user must supply a valid label unless a labeling function is in force or `LABEL_DEFAULT` applies for the table. `LABEL_DEFAULT` causes the user's session default row label to be used as the new row label.

If neither `LABEL_DEFAULT` nor a labeling function is in force and the user attempts to `INSERT` a row, then an error occurs.

Note that any labeling function in force on a table overrides the `LABEL_DEFAULT` option.

10.2.6.3 LABEL_UPDATE: Changing Data Labels

A user updating a row can normally change its label to any label within their authorized label range.

However, if `LABEL_UPDATE` applies, then to modify a label, the user must have one or more of these privileges: `WRITEUP`, `WRITEDOWN`, and `WRITEACROSS`.

The `LABEL_UPDATE` option uses an Oracle after-row trigger which is called only on an update operation affecting the label. Note that any labeling function in force on a table overrides the `LABEL_UPDATE` option.

Related Topics

- Special Row Label Privileges
  Once the label on a row has been set, Oracle Label Security privileges are required to modify the label.

10.2.6.4 CHECK_CONTROL: Checking Data Labels

If an inserted or updated row gets its label from a labeling function, the label could be outside the user's authorizations.

This prevents this user from being able to read or update the row. To prevent this problem, use the `CHECK_CONTROL` setting to allow `READ_CONTROL` to apply to the new label. This ensures that this user will be authorized to read the inserted or updated row after the operation. If not, then the insert or update operation is canceled and has no effect.

In other words, if `CHECK_CONTROL` is included as an option in a policy being enforced on a row, then the user modifying that row must still be able to access it after the operation. `CHECK_CONTROL` prevents a user or a labeling function from modifying a row's label to include a level, group, or compartment that the modifying user would be prevented from accessing.

Note that `CHECK_CONTROL` overrides any labeling function in force on a table.

10.2.7 How the Access Control Enforcement Options Work

Access control options limit the rows accessible for `SELECT`, `UPDATE`, `INSERT`, or `DELETE` operations to only those rows whose labels meet established policies.

- READ_CONTROL: Reading Data
  `READ_CONTROL` limits the set of records accessible to a session for `SELECT`, `UPDATE` and `DELETE` operations.
- WRITE_CONTROL: Writing Data
  When an Oracle Label Security policy specifying the `WRITE_CONTROL` option is applied to a table, triggers are generated and the algorithm is enforced.
- INSERT_CONTROL, UPDATE_CONTROL, and DELETE_CONTROL
  The `INSERT_CONTROL`, `UPDATE_CONTROL`, and `DELETE_CONTROL` options control policy enforcement during the corresponding operations on the data columns in a row.

10.2.7.1 READ_CONTROL: Reading Data

`READ_CONTROL` limits the set of records accessible to a session for `SELECT`, `UPDATE` and `DELETE` operations.

If `READ_CONTROL` is not active, then even rows in the table protected by the policy are accessible to all users.

`READ_CONTROL` uses Oracle virtual private database (VPD) technology to enforce the read access mediation algorithm illustrated in Figure 3-6.

10.2.7.2 WRITE_CONTROL: Writing Data

When an Oracle Label Security policy specifying the `WRITE_CONTROL` option is applied to a table, triggers are generated and the algorithm is enforced.

`WRITE_CONTROL` uses Oracle after-row triggers to enforce the write access mediation algorithm illustrated in Figure 3-7.

Note:

The protection implementation for `WRITE_CONTROL` is the same for all write operations, but you need not apply all write options across the board. You can apply `WRITE_CONTROL` selectively for `INSERT`, `UPDATE`, and `DELETE` operations by using the corresponding policy enforcement option (`INSERT_CONTROL`, `UPDATE_CONTROL`, and `DELETE_CONTROL`) instead of `WRITE_CONTROL`.

If `WRITE_CONTROL` is on but `LABEL_UPDATE` is not specified, then the user can change both data and labels. If you want to control updating the row labels, then specify the `LABEL_UPDATE` option in addition to `WRITE_CONTROL` when creating your policies.

10.2.7.3 INSERT_CONTROL, UPDATE_CONTROL, and DELETE_CONTROL

The `INSERT_CONTROL`, `UPDATE_CONTROL`, and `DELETE_CONTROL` options control policy enforcement during the corresponding operations on the data columns in a row.

These options apply according to the algorithm for write access described in Figure 3-7.

Specifying `WRITE_CONTROL` limits all `INSERT`, `UPDATE`, and `DELETE` operations. However,

- Specifying `INSERT_CONTROL` limits insertions but not updates or deletes.
- Specifying `UPDATE_CONTROL` limits updates but not insertions or deletes.
- Specifying `DELETE_CONTROL` limits deletes but not insertions or updates.

Related Topics

- Inserting Labeled Data Using Policy Options and Labeling Functions
  It is important to understand how enforcement options and labeling functions affect the insertion of labeled data.

- Updating Labeled Data Using Policy Options and Labeling Functions
  Users must be authorized to change rows that are protected by Oracle Label Security.
- Deletion of Labeled Data Using Policy Options and Labeling Functions
  You can delete labeled data.

10.2.8 How the Overriding Enforcement Options Work

Whereas `ALL_CONTROL` applies all of the label management and access control enforcement options, `NO_CONTROL` applies none of them.

In either case, labeling functions and SQL predicates can be applied. Note that the `ALL_CONTROL` option can be used only on the command line. If you apply a policy with `NO_CONTROL` specified, then a policy label column is added to the table, but the label values are `NULL`. Because no access controls are operating on the table, you can proceed to enter labels as desired. You can then set the policy enforcement options as you want. `NO_CONTROL` can be a useful option if you have a labeling function in force to label the data correctly, but want to let all users access all the data.

10.2.9 Guidelines for Using the Policy Enforcement Options

You can customize policy enforcement for a schema or table through the Oracle Enterprise Manager.

This functionality is described in Creating an Oracle Label Security Policy or you can use the `SA_POLICY_ADMIN` package as described in SA_POLICY_ADMIN Policy Administration PL/SQL Package.

This section documents the supported keywords.

Note that when you create a policy, you can specify a string of default options to be used whenever the policy is applied without schema or table options being specified.

If a policy is first applied to a table, and then also applied to the schema containing that table, then the options on the table are not affected by the schema policy. The options of the policy originally applied to the table remain in force.

In general, administrators use the `LABEL_DEFAULT` policy option, causing data written by a user to be labeled with that user's row label. Alternatively, a labeling function can be used to label the data. If neither of these two choices is used, then a label must be specified in every `INSERT` statement. (Updates retain the row's original label.)

The following table suggests that certain combinations of policy enforcement options are useful when implementing an Oracle Label Security policy. As the table indicates, you might typically enforce `READ_CONTROL` and `WRITE_CONTROL`, choosing from among several possible combinations for setting the data label on writes.

Table 10-4 Suggested Policy Enforcement Option Combinations

Options	Access Enforcement
`READ_CONTROL`, `WRITE_CONTROL`, `LABEL_DEFAULT`	Read and write access based on session label. Default label provided; users can insert/update both data and labels.

Table 10-4 (Cont.) Suggested Policy Enforcement Option Combinations

Options	Access Enforcement
`READ_CONTROL`, `WRITE_CONTROL`, Labeling Function	Read and write access based on session label. Users can set/change only row data; all row labels are set explicitly by the labeling function. Add `CHECK_CONTROL` to restrict new labels (on insert or update) to visible range of labels.
`READ_CONTROL`, `WRITE_CONTROL`, `LABEL_UPDATE`	Read and write access based on session label. Users cannot change labels without privileges. Add `CHECK_CONTROL` to restrict new labels (on insert or update) to visible range.

Related Topics

- Authorized Levels
  The administrator explicitly sets the level authorization for an Oracle Label Security policy.

10.2.10 Exemptions from Oracle Label Security Policy Enforcement

Oracle Label Security has several exceptions from OLS policy enforcement.

These exemptions are as follows:

- Oracle Label Security is not enforced during `DIRECT` path export.
- By design, Oracle Label Security policies cannot be applied to objects in schema `SYS`. As a consequence, the `SYS` user, and users making a DBA-privileged connection to the database (such as `CONNECT AS SYSDBA`) do not have Oracle Label Security policies applied to their actions. DBAs need to be able to administer the database. It would make no sense, for example, to export part of a table due to an Oracle Label Security policy being applied. The database user `SYS` is thus always exempt from Oracle Label Security enforcement, regardless of the export mode, application, or utility used to extract data from the database.
- Similarly, database users granted the `EXEMPT ACCESS POLICY` privilege, either directly or through a database role, are exempted from some Oracle Label Security policy enforcement controls such as `READ_CONTROL` and `CHECK_CONTROL`, regardless of the export mode, application or utility used to access the database or update its data. 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`.

  `EXEMPT ACCESS POLICY` is a very powerful privilege and should be carefully managed.

  Note that this privilege does not affect the enforcement of standard Oracle Database object privileges such as `SELECT`, `INSERT`, `UPDATE`, and `DELETE`. These privileges are enforced even if a user has been granted the EXEMPT ACCESS POLICY privilege.

Related Topics

- Categories of Policy Enforcement Options
  Oracle Label Security enforces policies using three categories: label management options, access control options, and overriding options.

10.2.11 Data Dictionary Views for Viewing Policy Options on Tables and Schemas

Oracle Label Security provides data dictionary views that describe the policy enforcement options currently applied to tables and schemas.

- DBA_SA_TABLE_POLICIES
- DBA_SA_SCHEMA_POLICIES

10.3 Labeling Functions

Labeling functions can compute and return a label using resources such as context variables (for example, date or username) and data values.

- Labeling Data Rows under Oracle Label Security
  There are three ways to label data that is being inserted or updated.
- How Labeling Functions in Oracle Label Security Policies Works
  Labeling functions enable you to consider, in your rules for assigning labels, information drawn from the application context.
- Creating a Labeling Function for a Policy
  You can use the CREATE OR REPLACE FUNCTION SQL statement to create a labeling function.
- Specifying a Labeling Function in a Policy
  You can use the SA_POLICY_ADMIN package to specify a labeling function.

10.3.1 Labeling Data Rows under Oracle Label Security

There are three ways to label data that is being inserted or updated.

- You can explicitly specify a label in every INSERT or UPDATE to the table.
- You can set the LABEL_DEFAULT option, which causes the session's row label to be used if an explicit row label is not included in the INSERT or UPDATE statement.
- You can create a labeling function, automatically calls on every INSERT or UPDATE statement and independently of any user's authorization.

The recommended approach is to write a labeling function to implement your rules for labeling data. If you specify a labeling function, then Oracle Label Security embeds a call to that function in INSERT and UPDATE triggers to compute a label.

For example, you could create a labeling function named my_label to use the contents of COL1 and COL2 of the new row to compute and return the appropriate label for the row. Then, you could insert, into your INSERT or UPDATE statements, the following reference:

```
my_label(:new.col1,:new.col2)
```

If you do not specify a labeling function, then specify the LABEL_DEFAULT option. Otherwise, you must explicitly specify a label on every INSERT or UPDATE statement.

10.3.2 How Labeling Functions in Oracle Label Security Policies Works

Labeling functions enable you to consider, in your rules for assigning labels, information drawn from the application context.

For example, you can use as a labeling consideration the IP address to which the user is attached. There are many opportunities to use SYS_CONTEXT in this way.

Note:

If the SQL statement is invalid, then an error will occur when you apply the labeling function to the table or policy. You should thoroughly test a labeling function before using it with tables.

Labeling functions override the LABEL_DEFAULT and LABEL_UPDATE options.

A labeling function is called in the context of a before-row trigger. This enables you to pass in the old and new values of the data record, as well as the old and new labels.

You can construct a labeling function to permit an explicit label to be passed in by the user.

All labeling functions must have return types of the LBACSYS.LBAC_LABEL data type. The TO_LBAC_DATA_LABEL function can be used to convert a label in character string format to a data type of LBACSYS.LBAC_LABEL. Note that LBACSYS must have the EXECUTE privilege on your labeling function. The owner of the labeling function must have the EXECUTE privilege on the LBAC_TRIGGER schema, with the GRANT option.

Note:

LBACSYS is a unique schema providing opaque types for Oracle Label Security.

Related Topics

- Performing DBA Functions Under Oracle Label Security
  Oracle Label Security supports the standard Oracle Database utilities, but certain restrictions apply, which may require extra steps to get the expected results.

10.3.3 Creating a Labeling Function for a Policy

You can use the `CREATE OR REPLACE FUNCTION` SQL statement to create a labeling function.

- To use the `CREATE OR REPLACE FUNCTION` statement to create a labeling function for a policy, set the return value to `LBACSYS.LBAC_LABEL`.

For example:

```
CREATE OR REPLACE FUNCTION sa_demo.gen_emp_label
           (Job varchar2,
            Deptno number,
            Total_sal number)
       Return LBACSYS.LBAC_LABEL
     as
       i_label varchar2(80);
     Begin
       /************* Determine Class Level *************/
       if total_sal > 2000 then
       i_label := 'L3:';
       elsif total_sal > 1000 then
       i_label := 'L2:';
       else
       i_label := 'L1:';
       end if;

       /************* Determine Compartment *************/
       IF Job in ('MANAGER','PRESIDENT') then
       i_label := i_label||'M:';
       else
       i_label := i_label||'E:';
       end if;
       /************* Determine Groups *************/
       i_label := i_label||'D'||to_char(deptno);
       return TO_LBAC_DATA_LABEL('human_resources',i_label);
     End;
     /
```

10.3.4 Specifying a Labeling Function in a Policy

You can use the `SA_POLICY_ADMIN` package to specify a labeling function.

- Use `SA_POLICY_ADMIN.REMOVE_TABLE_POLICY` and `SA_POLICY_ADMIN.APPLY_TABLE_POLICY` to specify the labeling function.

For example:

```
SA_POLICY_ADMIN.REMOVE_TABLE_POLICY('human_resources','sa_demo','emp');


SA_POLICY_ADMIN.APPLY_TABLE_POLICY(
  POLICY_NAME     => 'human_resources',
  SCHEMA_NAME     => 'sa_demo',
  TABLE_NAME      => 'emp',
  TABLE_OPTIONS   => 'READ_CONTROL,WRITE_CONTROL,CHECK_CONTROL',
  LABEL_FUNCTION  => 'sa_demo.gen_emp_label(:new.job,:new.deptno,:new.sal)',
  PREDICATE       => NULL);
```

10.4 Inserting Labeled Data Using Policy Options and Labeling Functions

It is important to understand how enforcement options and labeling functions affect the insertion of labeled data.

- Outcome of Insert or Updates Operations on Data Based on Authorizations
  When you attempt to insert or update data based on your authorizations, the outcome depends upon what policy enforcement controls are active.
- Label Insertions When a Labeling Function Is Specified
  A labeling function takes precedence over labels entered by the user.
- Child Row Insertions in Tables with Declarative Referential Integrity
  If declarative referential integrity protects a parent table, then the parent row must be visible before a child row can be inserted.

10.4.1 Outcome of Insert or Updates Operations on Data Based on Authorizations

When you attempt to insert or update data based on your authorizations, the outcome depends upon what policy enforcement controls are active.

- If `INSERT_CONTROL` is active, then rows you insert can only have labels within your write authorizations. If you attempt to update data that you can read, but for which you do not have write authorization, an error is raised. For example, if you can read compartments A and B, but you can only write to compartment A, then if you attempt to insert data with compartment B, then the statement will fail.
- If `INSERT_CONTROL` is *not* active, then you can use any valid label on rows you insert.
- If the `CHECK_CONTROL` option is active, then rows you insert can only have labels you are authorized to read, even if the labels are generated by a labeling function.

10.4.2 Label Insertions When a Labeling Function Is Specified

A labeling function takes precedence over labels entered by the user.

If the administrator has set up an automatic labeling function, then no data label a user enters will have effect (unless the labeling function itself makes use of the user's proposed label). New row labels are always determined by an active labeling function, if present.

Note that a labeling function can set the label of a row being inserted to a value outside the range that the user writing that row can see. If such a function is in use, then the user can potentially insert a row but not be authorized to see that row. You can prevent this situation by specifying the `CHECK_CONTROL` option in the policy. If this option is active, then the new data label is checked against the user's read authorization, and if the user cannot read it, then the insert operation is not performed.

10.4.3 Child Row Insertions in Tables with Declarative Referential Integrity

If declarative referential integrity protects a parent table, then the parent row must be visible before a child row can be inserted.

The user must be able to see the parent row for the insert operation to succeed, that is, the user must have read access to the parent row.

If `READ_CONTROL` is active on the parent table, then the user's read authorization must be sufficient to authorize a `SELECT` operation on the parent row. For example, a user who cannot read department 20 cannot insert child rows for department 20. Note that all records will be visible if the user has `FULL` or `READ` privileges on the table or schema.

10.5 Inserts of Rows into Foreign Key Tables That Do Not Exist Yet in Referential Tables

If you insert a row into a foreign key table that does not yet exist in the referenced table with a primary key, then an `ORA-28117: integrity constraint violated - parent record not found` error occurs when you run the statement to perform the insert, not at `COMMIT` time.

Inserts into a table with a `DEFERRED` foreign key constraint involving an Oracle Label Security protected table are evaluated immediately and not deferred till a `COMMIT` has been performed. This is expected behavior because Oracle Label Security uses label authorizations to while performing referential constraint checks. If the referenced table with the primary key is not Oracle Label Security protected, then the error occurs at `COMMIT` time with normal error messages.

10.6 Updating Labeled Data Using Policy Options and Labeling Functions

Users must be authorized to change rows that are protected by Oracle Label Security.

- Updating Labels Using CHAR_TO_LABEL
  To change a row label from `SENSITIVE` to `CONFIDENTIAL`, you can change the label by using the `CHAR_TO_LABEL` function.
- Evaluation of Enforcement Control Options and UPDATE
  When you attempt to update data based on your authorizations, the outcome depends on which enforcement controls are active.
- Updates to Labels When a Labeling Function Is Specified
  A labeling function takes precedence over labels entered by the user.
- Updates to Child Rows in Tables with Declarative Referential Integrity Enabled
  If a child row is in a table with a referential integrity constraint, then the parent row must be visible for the update to succeed.

10.6.1 Updating Labels Using CHAR_TO_LABEL

To change a row label from `SENSITIVE` to `CONFIDENTIAL`, you can change the label by using the `CHAR_TO_LABEL` function.

- To change a row label, use the `UPDATE` SQL statement.

For example:

```
UPDATE emp
SET hr_label = char_to_label ('HR', 'CONFIDENTIAL')
WHERE ename = 'ESTANTON';
```

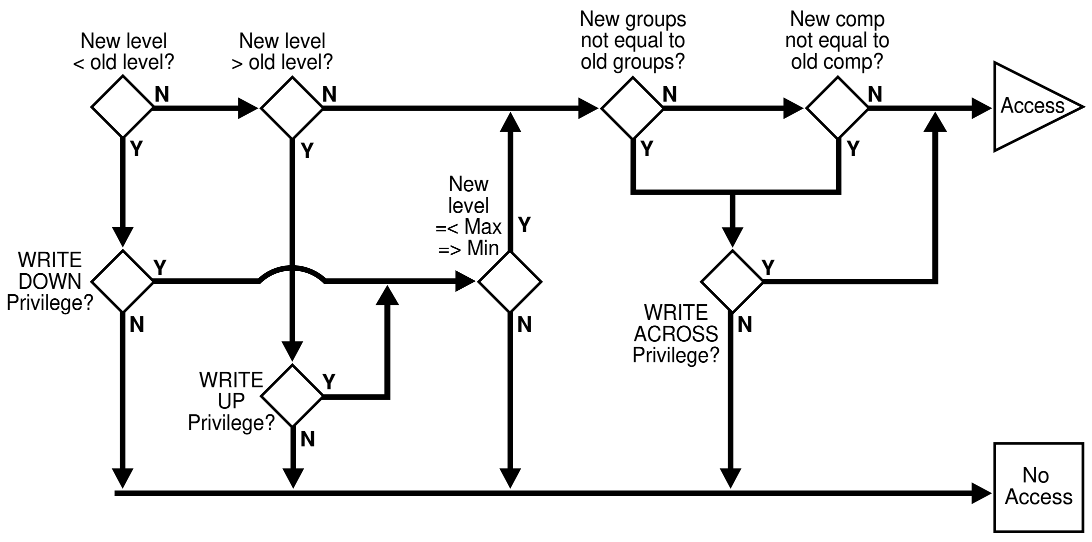
10.6.2 Evaluation of Enforcement Control Options and UPDATE

When you attempt to update data based on your authorizations, the outcome depends on which enforcement controls are active.

- If `UPDATE_CONTROL` is active, then you can only update rows whose labels fall within your write authorizations. If you attempt to update data that you can read, but for which you do not have write authorization, then an error is raised. Assume, for example, that you can read compartments A and B, but you can only write to compartment A. In this case, if you attempt to update data with compartment B, then the statement will fail.
- If `UPDATE_CONTROL` is not active, then you can update all rows to which you have read access.
- If `LABEL_UPDATE` is active, then you must have the appropriate privilege (`WRITEUP`, `WRITEDOWN`, or `WRITEACROSS`) to change a label by raising or lowering its sensitivity level, or altering its groups or compartments.
- If `LABEL_UPDATE` is *not* active but `UPDATE_CONTROL` *is* active, then you can update a label to any new label value within your write authorization.
- If `CHECK_CONTROL` is active, then you can only write labels you are authorized to read.

The following figure illustrates the label evaluation process for `LABEL_UPDATE`.

Figure 10-1 Label Evaluation Process for LABEL_UPDATE



10.6.3 Updates to Labels When a Labeling Function Is Specified

A labeling function takes precedence over labels entered by the user.

If the administrator has set up an automatic labeling function, then no label a user enters will have effect (unless the labeling function itself makes use of the user's proposed label). New row labels are always determined by an active labeling function, if present.

Note that the security administrator can establish a labeling function that sets the label of a row being updated to a value outside the range that you can see. If this is the case, then you can update a row, but not be authorized to see the row. If the `CHECK_CONTROL` option is on, then you will not be able to perform such an update. The `CHECK_CONTROL` option verifies your read authorization on the new label.

10.6.4 Updates to Child Rows in Tables with Declarative Referential Integrity Enabled

If a child row is in a table with a referential integrity constraint, then the parent row must be visible for the update to succeed.

That is, this user must be able to see the parent row.

If the parent table has `READ_CONTROL` on, then the user's read authorization must be sufficient to authorize a `SELECT` on the parent row.

For example, a user who cannot read department 20 in a parent table cannot update an employee's department to department 20 in a child table. (If the user has `FULL` or `READ` privilege, then all records will be visible.)

See Also:

Oracle Database Development Guide

10.7 Deletion of Labeled Data Using Policy Options and Labeling Functions

You can delete labeled data.

Note the following:

- If DELETE_CONTROL is active, then you can delete only rows within your write authorization.
- If DELETE_CONTROL is *not* active, then you can delete only rows that you can read.
- With DELETE_CONTROL active, and declarative referential integrity defined with cascading deletes, you must have write authorization on *all* the rows to be deleted, or the statement will fail.

You cannot delete a parent row if there are any child rows attached to it, regardless of your write authorization. To delete such a parent row, you must first delete each of the child rows. If DELETE_CONTROL is active on any of the child rows, then you must have write authorization to delete the child rows.

Consider, for example, a situation in which the user is UNCLASSIFIED and there are three rows as follows:

Row	Table	Sensitivity
Parent row:	DEPT	UNCLASSIFIED
Child row:	EMP	UNCLASSIFIED
Child row:	EMP	UNCLASSIFIED

In this case, the UNCLASSIFIED user cannot delete the parent row.

DELETE_CONTROL has no effect when DELETERESTRICT is active. DELETERESTRICT is always enforced. In some cases (depending on the user's authorizations and the data's labels) it may look as though a row has no child rows, when it actually does have children but the user cannot see them. Even if a user cannot see child rows, they still cannot delete the parent row.

10.8 SQL Predicates with an Oracle Label Security Policy

You can use a SQL predicate to provide extensibility for selective enforcement of data access rules.

- Modifications to an Oracle Label Security Policy with a SQL Predicate
  A SQL predicate is a condition, optionally preceded by AND or OR.
- How Multiple SQL Predicates Affect Oracle Label Security Policies
  Predicates can be appended to other predicates.

10.8.1 Modifications to an Oracle Label Security Policy with a SQL Predicate

A SQL predicate is a condition, optionally preceded by `AND` or `OR`.

The SQL predicate can be appended for `READ_CONTROL` access mediation. The following predicate, for example, adds an application-specific test based on `COL1` to determine if the session has access to the row.

```
AND my_function(col1)=1
```

The combined result of the policy and the user-specified predicate limits the rows that a user can read. So, this combination affects the labels and data that `CHECK_CONTROL` will permit a user to change. An `OR` clause, for example, increases the number of rows a user can read.

A SQL predicate can be useful if you want to avoid performing label-based filtering. In certain situations, a SQL predicate can easily implement row-level security on tables. Used instead of `READ_CONTROL`, a SQL predicate will filter the data for `SELECT`, `UPDATE`, and `DELETE` operations.

Similarly, in a typical, Web-enabled human resources application, a user might have to be a manager to access rows in the employee table. In such cases, the user's user label would have to dominate the label on the employee's row. A SQL predicate like the following could be added, so that an employee could bypass label-based filtering if they wanted to view their own record in the employee table. (An `OR` is used so that *either* the label policy will apply, *or* this statement will apply.)

```
OR SYS_CONTEXT ('USERENV', 'SESSION_USER') = employee_name
```

This predicate enables you to have additional access controls so that each employee can access their own record.

You can use such a predicate in conjunction with `READ_CONTROL`s or as a standalone predicate even if `READ_CONTROL` is not implemented.

Note:

Verify that the predicate accomplishes your security goals before you implement it in an application.

If a syntax error occurs in a predicate under Oracle Label Security, then an error will *not* arise when you try to apply the policy to a table. Rather, a predicate error message will arise when you first attempt to reference the table.

10.8.2 How Multiple SQL Predicates Affect Oracle Label Security Policies

Predicates can be appended to other predicates.

A predicate applied to a table with an Oracle Label Security policy is appended to other predicates that are applied by other Oracle Label Security policies, or by Oracle Database fine-grained access control or Oracle Virtual Private Database policies. The predicates are `AND`ed together.

Consider the following predicates applied to the `EMP` table in the `SCOTT` schema:

- A predicate generated by an Oracle VPD policy, such as `deptno=10`
- A label-based predicate generated by an Oracle Label Security policy, such as `label=100`, with a user-specified predicate such as

  ```
  OR SYS_CONTEXT ('USERENV', 'SESSION_USER') = ename
  ```

Correct: These predicates would be `AND`ed together as follows:

```
WHERE deptno=10 AND (label=100 OR SYS_CONTEXT ('USERENV', 'SESSION_USER') =
ename)
```

Incorrect: The predicates would *not* be combined in the following way:

```
WHERE deptno=10 AND label=100 OR SYS_CONTEXT ('USERENV', 'SESSION_USER') = ename
```

11 Administering and Using Trusted Stored Program Units

You can use trusted stored program units to enhance system security.

- About Trusted Stored Program Units
  Oracle Database stored procedures, functions, and packages are sets of PL/SQL statements stored in a database in compiled form.
- How a Trusted Stored Program Unit Runs
  A trusted stored program unit runs using its own privileges, and the caller's labels.
- Example: Trusted Stored Program Unit
  A trusted stored program unit with the `READ` privilege can read all unprotected data and all data protected by this policy.
- Creating and Compiling Trusted Stored Program Units
  You can create and compile trusted stored program units for use in Oracle Label Security.
- How Setting and Returning Label Information Works
  The `SA_UTL` package has functions to return information about current values of session security attributes using numeric label values.

11.1 About Trusted Stored Program Units

Oracle Database stored procedures, functions, and packages are sets of PL/SQL statements stored in a database in compiled form.

The single difference between functions and procedures is that functions return a value and procedures do not. Trusted stored program units are like any other stored program units in *Oracle Database*: the underlying logic is the same.

A *package* is a set of procedures and functions, together with the cursors and variables they use, stored as a unit. There are two parts to a package, the package specification and the package body. The package specification declares the external definition of the public procedures, functions, and variables that the package contains. The package body contains the actual text of the procedures and functions, as well as any private procedures and variables.

A *trusted stored program unit* is a stored procedure, function, or package that has been granted one or more Oracle Label Security privileges. Trusted stored program units are typically used to let users perform privileged operations in a controlled manner, or update data at several labels. This is the optimal approach to permit users to access data beyond their authorization.

Trusted stored program units provide fine-grained control over the use of privileges. Although you can potentially grant privileges to many users, the granting of privileges should be done with great discretion because it might violate the security policy established for your application. Rather than assigning privileges to users, you can identify any application operations requiring privileges, and implement them as trusted program units. When you grant privileges to these stored program units, you effectively restrict the Oracle Label Security privileges required by users. This approach employs the principle of *least privilege*.

For example, if a user with the label `CONFIDENTIAL` needs to insert data into `SENSITIVE` rows, then you can grant the `WRITEUP` privilege to a trusted stored program to which the user has access. In this way, the user can perform the task by means of the trusted stored program, while staying at the `CONFIDENTIAL` level.

The trusted program unit performs all the actions on behalf of the user. You can thus effectively encapsulate the security policy into a module that can be verified to make sure that it is valid.

11.2 How a Trusted Stored Program Unit Runs

A trusted stored program unit runs using its own privileges, and the caller's labels.

In this way, the trusted stored program unit can perform privileged operations on the set of rows constrained by the user's labels.

Oracle Database system and object privileges are intended to be bundled into roles. Users are then granted roles as necessary. By contrast, Oracle Label Security privileges can only be assigned to users or to stored program units. These trusted stored program units provide a more manageable mechanism than roles to control the use of Oracle Label Security privileges.

11.3 Example: Trusted Stored Program Unit

A trusted stored program unit with the `READ` privilege can read all unprotected data and all data protected by this policy.

Consider, for example, a user who is responsible for creating purchasing forecast reports. The user must perform a summation operation on the amount of all purchases. Regardless of whether or not user's own labels authorize access to the individual purchase orders. The syntax for creating the summation procedure in this example is as follows:

```
CREATE FUNCTION sum_purchases RETURN NUMBER IS
  psum NUMBER;
BEGIN
  SELECT SUM(amount) INTO psum
  FROM purchase_orders;
RETURN psum;
END sum_purchases;
```

In this way, the program unit can gather information the end user is not able to gather, and can make it available by means of a summation.

Note that to run `SUM_PURCHASES`, the user would need to be granted the standard Oracle Database `EXECUTE` object privilege upon this procedure.

Related Topics

- Access Controls and Privileges
  Oracle provides access controls and privileges that determine the *type* of access users can have to labeled rows.

11.4 Creating and Compiling Trusted Stored Program Units

You can create and compile trusted stored program units for use in Oracle Label Security.

- Creation of Trusted Stored Program Units
  You can create a trusted stored program unit in the same way that you create a standard procedure, function, or package.
- Privileges for Trusted Stored Program Units
  An Oracle Label Security administrator can verify the correctness of a stored program unit code before granting the privileges to it.
- Recompiling of Trusted Stored Program Units
  Recompiling a trusted stored program unit, either automatically or manually (using `ALTER PROCEDURE`), does not affect its Oracle Label Security privileges.
- Re-creation of Trusted Stored Program Units
  Oracle Label Security privileges are revoked if you perform a `CREATE` or `REPLACE` operation on a trusted stored program unit.
- Execution of Trusted Stored Program Units
  Under Oracle Label Security all the standard Oracle Database controls on procedure call (regarding access to tables and schemas) are still in force.

11.4.1 Creation of Trusted Stored Program Units

You can create a trusted stored program unit in the same way that you create a standard procedure, function, or package.

To do this, you can use the `CREATE PROCEDURE`, `CREATE FUNCTION`, or `CREATE PACKAGE` and `CREATE PACKAGE BODY` statements.

The program unit becomes trusted when you grant it Oracle Label Security privileges.

See Also:

Oracle Database SQL Language Reference

11.4.2 Privileges for Trusted Stored Program Units

An Oracle Label Security administrator can verify the correctness of a stored program unit code before granting the privileges to it.

Typically another user, such as a developer, creates the stored program unit. Whenever the trusted stored program unit is re-created or replaced, Oracle Label security removes its privileges. The Oracle Label Security administrator must then verify the code again and grant the privileges once again.

The Oracle Label Security administrator should review the program unit code carefully and evaluate the privileges that are to be granted to it. For example, procedures in trusted packages should not perform privileged database operations and then write result or status information into a public variable of the package. In this way, you can make sure that no violations of your site's Oracle Label Security policy can occur.

Related Topics

- SA_USER_ADMIN.SET_PROG_PRIVS
  The `SA_USER_ADMIN.SET_PROG_PRIVS` procedure sets policy-specific privileges for program units.

11.4.3 Recompiling of Trusted Stored Program Units

Recompiling a trusted stored program unit, either automatically or manually (using `ALTER PROCEDURE`), does not affect its Oracle Label Security privileges.

You must, however, grant the `EXECUTE` privilege on the program unit again after recompiling.

11.4.4 Re-creation of Trusted Stored Program Units

Oracle Label Security privileges are revoked if you perform a `CREATE` or `REPLACE` operation on a trusted stored program unit.

This limits the potential for misuse of a procedure's Oracle Label Security privileges.

Note that the procedure, function, or package can still run even if the Oracle Label Security privileges have been removed.

If you re-create a procedure, function, or package, then you should carefully review its text. When you are certain that the re-created program unit does not violate your site's Oracle Label Security policy, you can then grant it the required privileges again.

In a development environment where trusted stored program units must frequently be replaced (for example, during the first few months of a live system), it is advisable to create a script that can grant the proper Oracle Label Security privileges, as required.

11.4.5 Execution of Trusted Stored Program Units

Under Oracle Label Security all the standard Oracle Database controls on procedure call (regarding access to tables and schemas) are still in force.

Oracle Label Security complements these security mechanisms by controlling access to rows.

When a trusted stored program unit is carried out, the policy privileges in force are a union of the invoking user's privileges and the program unit's privileges. Whether a trusted stored program unit calls another trusted program unit or a non-trusted program unit, the program unit called runs with the same privileges as the original program unit.

If a sequence of non-trusted and trusted stored program units is carried out, the first trusted program unit will determine the privileges of the entire calling sequence from that point on. Consider the following sequence:

Procedure A (non-trusted)
Procedure B with `WRITEUP`
Procedure C with `WRITEDOWN`
Procedure D (non-trusted)

Here, Procedures B, C, and D all runs with the `WRITEUP` privilege, because B was the first trusted procedure in the sequence. When the sequence ends, the privilege pertaining to Procedure B is no longer in force for subsequent procedures.

Note:

Unhandled exceptions raised in trusted program units are caught by Oracle Label Security. This means that error messages may not be displayed to the user. For this reason, you should always thoroughly test and debug any program units before granting them privileges.

11.5 How Setting and Returning Label Information Works

The `SA_UTL` package has functions to return information about current values of session security attributes using numeric label values.

Although these functions can be used in program units that are not trusted, they are primarily for use in trusted stored program units.

Note that these are public functions; you do not need the *`policy`*`_DBA` role to use them. In addition, each of the functions has a parallel `SA_SESSION` function that returns the same labels in character string format.

Related Topics

- Duties of Oracle Label Security Administrators
  Oracle Label Security administrators have a set of package- and role-based privileges.

12

Using Oracle Label Security with a Distributed Database

You should understand the special considerations for using Oracle Label Security in a distributed configuration.

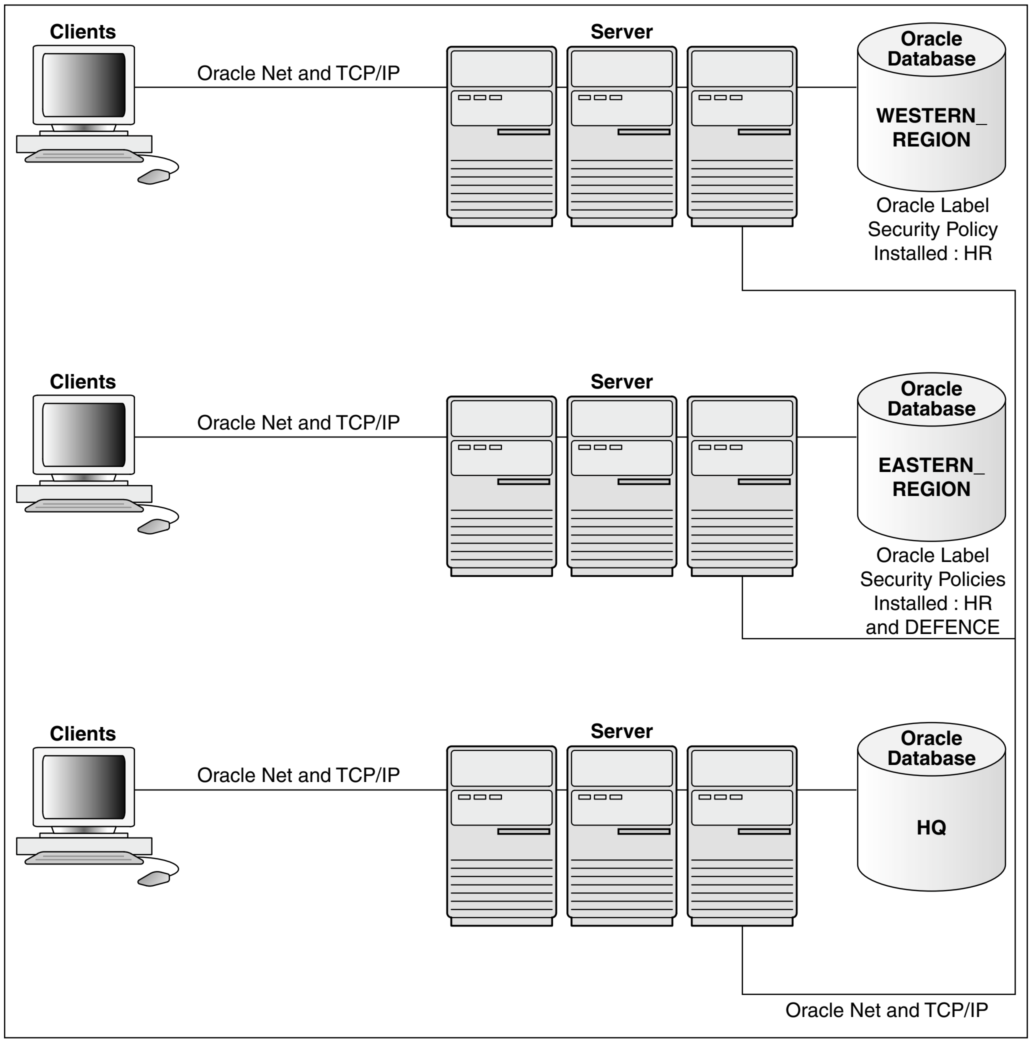
12.1 About the Oracle Label Security Distributed Configuration

In a network configuration that supports distributed databases, multiple Oracle Database (or other) servers can run on the same or different operating systems.

Each cooperative server in a distributed system communicates with other clients and servers over a network.

Figure 12-1 illustrates a distributed database that includes clients and servers with and without Oracle Label Security. As described in this chapter, if you establish database links from the `WESTERN_REGION` database to the `EASTERN_REGION` database, then you can access data if your user ID on `EASTERN_REGION` is authorized to see it, even if locally (on `WESTERN_REGION`) you do not have this access.

Figure 12-1 Using Oracle Label Security with a Distributed Database



12.2 How Connections to a Remote Database Under Oracle Label Security Work

Distributed databases act in the standard way with Oracle Label Security: the local user ends up connected as a particular remote user.

Oracle Label Security protects the labeled data, whether you connect locally or remotely. If the remote user has the proper labels, then you can access the data. If not, then you cannot access the data.

The database link sets up the connection to the remote database and identifies the user who will be associated with the remote session. Your Oracle Label Security authorizations on the remote database are based on those of the remote user identified in the database link.

For example, local user `JANE` might connect as remote user `AUSTEN`, in the database referenced by the connect string `sales`, as follows:

```
CREATE DATABASE LINK sales
  CONNECT TO austen IDENTIFIED BY pride
  USING 'sales'
```

When `JANE` connects, their authorizations are based on the labels and privileges of remote user `AUSTEN`, because `AUSTEN` is the user identified in the database link. When `JANE` makes the first reference to the remote database, the remote session is actually established. For example, the remote session would be created if `JANE` enters:

```
SELECT * FROM emp@sales
```

You need not be an Oracle Label Security policy user in the local database. If you connect as a policy user on the remote database, you can access protected data.

12.3 Session Labels and Row Labels in Remote Sessions

When connecting remotely, you can directly control the session label and row label in effect when you establish the connection.

When you connect, Oracle Label Security passes these values (for all policies) over to the remote database. Notice that:

- The local session label and row label are used as the default for the remote session, if they are valid for the remote user.
- The remote session is constrained by the minimum and maximum authorizations of the remote user.
- Although the local user's session labels are passed to the remote database, the local user's privileges are not passed. The privileges for the remote session are those associated with the remote user.

Consider a local user, `Diana`, with a maximum level of `HS`, and a session level of `S`. On the remote database, the remote user identified in the database link has a maximum level of `S`.

- If Diana's session label is `S` when the database link is established, then the `S` label is passed over. This is a valid label. Diana can connect and read `SENSITIVE` data.
- If Diana's session label is `HS` when the database link is established, then the `HS` level is passed across, but it is not valid for the remote user. Diana will pick up the remote user's default label (`S`).

Be aware of the label at which you are running the first time you connect to the remote database. The first time you reference a database link, your local session labels are sent across to the remote system when a connection is made. Later, you can change the label, but to do so, you must run the `SA_SESSION.SET_LABEL` procedure on the remote database.

Diana can connect at level `HS`, set the label to `S`, and then perform a remote access. Connection is implicitly made when the database link is established. Diana's default label is `S` on the remote database.

On the local database, Diana can set their session label to their maximum level of `HS`, but if the label of the remote user is set to `S`, then Diana can only retrieve `S` data from the remote database. If Diana performs a distributed query, then they will get `HS` data from the local database, and `S` data from the remote database.

12.4 Labels in a Distributed Environment

You should use the same label component definitions and label tags on any database that is to be protected by the policy.

- Label Tags in a Distributed Environment
  In a distributed environment, you may choose to use the same label tags across multiple databases.
- Numeric Form of Label Components in a Distributed Environment
  In a distributed environment, the same relative ranking of the numeric form of the level component ensures that the labels are properly sorted.

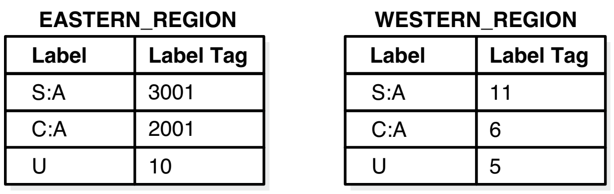
12.4.1 Label Tags in a Distributed Environment

In a distributed environment, you may choose to use the same label tags across multiple databases.

However, if you choose *not* to use the same tags across multiple databases, then you should retrieve the character form of the label when performing remote operations. This will ensure that the labels are consistent.

In the following example, the character string representation of the label string is the same. However, the label tag does not match. If the retrieved label tag has a value of `11` on the `WESTERN_REGION` database but a tag of `2001` on the `EASTERN_REGION` database, then the tags have no meaning. Serious consequences can result.

Figure 12-2 Label Tags in a Distributed Database



EASTERN_REGION

Label	Label Tag
S:A	3001
C:A	2001
U	10

WESTERN_REGION

Label	Label Tag
S:A	11
C:A	6
U	5

When retrieving labels from a remote system, you should return the character string representation (rather than the numeric label tag), unless you are using the same numeric labels on both databases.

If you allow Oracle Label Security to automatically generate labels on different databases, then the label tags will not be identical. Character strings will have meaning, but the numeric values will not, unless you have predefined labels with the same label tags on both instances.

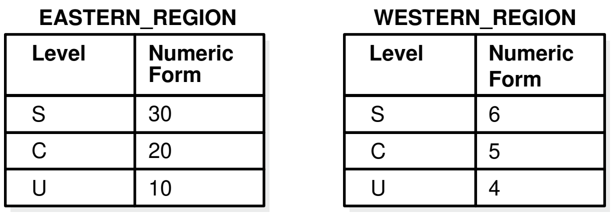
To avoid the complexities of label tags, you can convert labels to strings on retrieval (using `LABEL_TO_CHAR`) and use `CHAR_TO_LABEL` when you store labels. Operations will succeed as long as the component names are the same.

12.4.2 Numeric Form of Label Components in a Distributed Environment

In a distributed environment, the same relative ranking of the numeric form of the level component ensures that the labels are properly sorted.

In the following example, the levels in the two databases are effectively the same. Although the numeric form is different, the relative ranking of the levels numeric form is the same. As long as the relative order of the components is the same, the labels are perceived as identical.

Figure 12-3   Label Components in a Distributed Database



EASTERN_REGION

Level	Numeric Form
S	30
C	20
U	10

WESTERN_REGION

Level	Numeric Form
S	6
C	5
U	4

12.5 Oracle Label Security Policies in a Distributed Environment

Oracle Label Security supports all standard Oracle Database distributed configurations.

Whether or not you can access protected data depends on the policies installed in each distributed database.

Be sure to take into account the relationships between databases in a distributed environment:

- If the same application runs on two databases and you want them to have the same protection, then you must apply the same Oracle Label Security policy to both the local and the remote databases.
- If the local and remote databases have a policy in common, then your local session label and row label will override the default labels for the remote user.
- If the remote database has a different policy than the local database, then the remote policy can restrict access to the data independent of your local policies. On the other hand, when you make a connection as a remote user who has authorization on the remote policy, you can access any data to which the remote user has access to, regardless of your local authorizations.

If the remote database has no policy applied to it, you can access its data just as you would with a standard distributed database.

Consider a situation in which three databases exist, with different Oracle Label Security policies in force:

Database 1 has Policy A and Policy B
Database 2 has Policy A
Database 3 had Policy C

Users authorized for Policy A can obtain protected data from Database 1 and Database 2. If the remote user is authorized for Policy C, then this user can obtain data from Database 3 as well.

12.6 Replication with Oracle Label Security

You should understand how to use the replication option with tables protected by Oracle Label Security policies.

- About Replication Under Oracle Label Security
  You can replicate data in Oracle Label Security.
- Contents of a Materialized View
  Oracle Label Security can create materialized views.
- Requirements for Creating Materialized Views Under Oracle Label Security
  The requirements for creating a materialized view depend on the type of materialized view you are creating.
- How to Refresh Materialized Views
  If the contents or definition of a master table changes, then you should refresh the materialized view.

12.6.1 About Replication Under Oracle Label Security

You can replicate data in Oracle Label Security.

- Replication Functionality Supported by Oracle Label Security
  Oracle Label Security supports replication using read-only materialized views (snapshots).
- Row-Level Security Restriction on Replication Under Oracle Label Security
  An Oracle Label Security policy applies Row Level Security (RLS) to a table if `READ_CONTROL` is specified as one of the policy options.

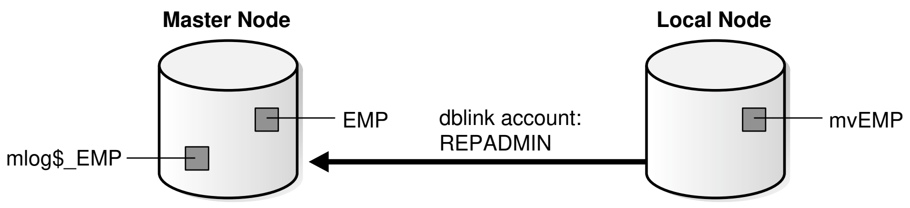
12.6.1.1 Replication Functionality Supported by Oracle Label Security

Oracle Label Security supports replication using read-only materialized views (snapshots).

Oracle Database uses materialized views for replicating data. A *materialized view* is a local copy of a local or remote master table that reflects a recent state of the master table.

As illustrated in the following figure, a master table is a table you wish to replicate, on a node that you designate as the master node. Using a `dblink` account, you can create a materialized view of the table in a different database. (This can also be done in the same database, and on the same system.) You can select rows from the remote master table, and copy them into the local materialized view. Here, `mvEMP` represents the materialized view of table `EMP`, and `mlog$_EMP` represents the materialized view log.

Figure 12-4 Use of Materialized Views for Replication



In a distributed environment, a materialized view alleviates query traffic over the network and increases data availability when a node is not available.

12.6.1.2 Row-Level Security Restriction on Replication Under Oracle Label Security

An Oracle Label Security policy applies Row Level Security (RLS) to a table if `READ_CONTROL` is specified as one of the policy options.

Problems occur if *both* of the following conditions are true:

- The Oracle Label Security policy is applied to any table relevant to replication (such as the master table, materialized view, or materialized view log), and
- The policy returns a predicate in the `WHERE` clause of `SELECT` statements.

To avoid the additional predicate (and therefore avoid this problem), the users involved in a replication environment should be given the necessary Oracle Label Security privileges. To be specific, the designated users in the database link (such as `REPADMIN` and the materialized view owner) must have the `READ` or the `FULL` privilege. As a result, the queries used to perform the replication will not be modified by RLS.

12.6.2 Contents of a Materialized View

Oracle Label Security can create materialized views.

- How Materialized View Contents Are Determined
  Oracle Label Security performs a set steps when creating materialized views.
- Complete Materialized Views
  Oracle Label Security supports complete materialized views.
- Partial Materialized Views
  A partial materialized view is created when you specify a `WHERE` clause in the materialized view definition.

12.6.2.1 How Materialized View Contents Are Determined

Oracle Label Security performs a set steps when creating materialized views.

The following steps determine the contents of the view:

1. It reads the definition of the master table in the remote database.
2. It reads the rows in the master table that meet the conditions defined in the materialized view definition.
3. It writes these rows to the materialized view in the local database.

Because Oracle Label Security writes only those rows to which you have write access in the local database, the contents of the materialized view vary according to:

- The policy options in effect
- The privileges you have defined in the local database
- The session label

12.6.2.2 Complete Materialized Views

Oracle Label Security supports complete materialized views.

If you read all of the rows in the master table and have write access in the local database to each label in the materialized view, then the result is a complete materialized view of the master table. To ensure that the materialized view is complete, you should have read access to all of the data in the master table and write access in the local database to all labels at which data is stored in the master table.

Note:

Never revoke privileges that you granted when you created the materialized view. If you do, then you may not be able to perform a replication refresh.

12.6.2.3 Partial Materialized Views

A partial materialized view is created when you specify a `WHERE` clause in the materialized view definition.

A partial materialized view is a convenient way to pass subsets of data to a remote database.

To create a partial materialized view, a user must have write access to all the rows being replicated. You can find the currently granted privileges for a user by querying the `DBA_SA_USER_PRIVS` data dictionary view.

12.6.3 Requirements for Creating Materialized Views Under Oracle Label Security

The requirements for creating a materialized view depend on the type of materialized view you are creating.

- Requirements for a Replication Administrator
  Requirements for a replication administrator, typically using a `REPADMIN` account, vary depending on the configuration.
- Requirements for the Owner of the Materialized View
  The privileges that belong to the owner of the materialized view are used during the refresh of the materialized view.
- Requirements for Creating Partial Multilevel Materialized Views
  A partial materialized view can include only some of the rows in a remote master table that is protected by Oracle Label Security.
- Requirements for Creating Complete Multilevel Materialized Views
  A complete materialized view can include every row in a remote master table that is protected by Oracle Label Security.

12.6.3.1 Requirements for a Replication Administrator

Requirements for a replication administrator, typically using a `REPADMIN` account, vary depending on the configuration.

In general, however, it should meet the following requirements:

- It must have the FULL Oracle Label Security privilege (mandatory for all configurations).
- It must have the SELECT privilege on the master table.
- It must be the account that establishes the database link from the remote node to the database containing the master table.

12.6.3.2 Requirements for the Owner of the Materialized View

The privileges that belong to the owner of the materialized view are used during the refresh of the materialized view.

If these privileges are not sufficient, then there are two options:

- The materialized view can be created in the REPADMIN account, or
- Additional privileges must be granted to the owner of the materialized view.

Consider, for example, the following materialized view created by user SCOTT:

```
CREATE MATERIALIZED VIEW mvemp as
SELECT *
FROM EMP@link_to_master
WHERE label_to_char(sa_label) = 'HS';
```

Here, SCOTT should have permission to insert records at the HS level in the local database. If Oracle Label Security policies are applied on the materialized view, then SCOTT must have the FULL privilege to avoid the RLS restriction.

Different configurations can be set up depending on whether Oracle Label Security policies are applied on the materialized view, what privileges are granted to the owner of the materialized view, and so on. If Oracle Label Security policies are applied to the materialized view, but SCOTT should not be granted the FULL privilege, then the REPADMIN account must be used to create the materialized view. SCOTT can then be granted the SELECT privilege on that table.

If no policies are applied to the materialized view, then the view can be created in SCOTT's schema without any additional privileges. In this case, the materialized view should be created in such a way that a WHERE condition limits the records to those which SCOTT can read.

Finally, if SCOTT can be granted the FULL privilege, then the materialized view can be created in SCOTT's schema, and Oracle Label Security policies can also be applied on the materialized view.

Note that the master table can have Oracle Label Security policies containing any set of policy options. If SCOTT has the FULL or the READ privilege, then he can select all rows, regardless of policy options.

12.6.3.3 Requirements for Creating Partial Multilevel Materialized Views

A partial materialized view can include only some of the rows in a remote master table that is protected by Oracle Label Security.

If the partial materialized view is used in a table that Oracle Label Security protects, then you should ensure that you have sufficient privileges to WRITE in the local database at every label retrieved by your query. You can find your currently granted privileges by querying the ALL_SA_USER_PRIVS data dictionary view.

12.6.3.4 Requirements for Creating Complete Multilevel Materialized Views

A complete materialized view can include every row in a remote master table that is protected by Oracle Label Security.

If the complete materialized view is used in a table that Oracle Label Security protects, then you must be able to have `WRITE` access in the local database at the labels of all of the rows retrieved by the defined materialized view query. You can find your currently granted privileges by querying the `ALL_SA_USER_PRIVS` data dictionary view.

12.6.4 How to Refresh Materialized Views

If the contents or definition of a master table changes, then you should refresh the materialized view.

This ensures that the materialized view accurately reflects the contents of the master table.

To refresh a materialized view of a remote multilevel table, you must also have privileges to write in the local database at the labels of all of the rows that the materialized view query retrieves

WARNING:

A materialized view can potentially contain outdated rows if you refresh a partial or full materialized view but do not have READ access to all the rows in the master table, and consequently do not overwrite the rows in the original materialized view with the updated rows from the master table.

To ensure an accurate materialized view refresh, you should use job queues to refresh the views automatically. These processes must have sufficient privileges both to read all of the rows in the master table and to write those rows to the materialized view, ensuring that the view is completely refreshed. Remember that the privileges used by these processes are those of the materialized view owner.

See Also:

Oracle Database Data Warehousing Guide for information about job queues

13

Performing DBA Functions Under Oracle Label Security

Oracle Label Security supports the standard Oracle Database utilities, but certain restrictions apply, which may require extra steps to get the expected results.

- Oracle Data Pump Export Use with Oracle Label Security
  Oracle Data Pump enables high-speed movement of data and metadata from one database to another.
- Data Pump Import Use with Oracle Label Security
  Oracle Data Pump enables high-speed movement of data and metadata from one database to another.
- SQL*Loader Use with Oracle Label Security
  SQL*Loader moves data from external files into tables in Oracle Database.
- Performance Tips for Oracle Label Security
  You can achieve optimal performance with Oracle Label Security.
- Creation of Additional Databases After Installation
  You can create and configure additional databases after you install Oracle Label Security.
- Oracle Label Security Upgrades and Downgrades
  You should be aware of how to manage Oracle Label Security upgrades and downgrades.

13.1 Oracle Data Pump Export Use with Oracle Label Security

Oracle Data Pump enables high-speed movement of data and metadata from one database to another.

- Full Database Export
  Starting with Oracle Database 12*c*, Oracle Label Security metadata in the `LBACSYS` schema can be included when doing a full database export and import operation.
- Schema and Table-Level Export
  The Data Pump export utility functions in the standard way under Oracle Label Security.

13.1.1 Full Database Export

Starting with Oracle Database 12*c*, Oracle Label Security metadata in the `LBACSYS` schema can be included when doing a full database export and import operation.

The source database can be Oracle Database 11*g* release 2 (11.2.0.3), or higher, but the target database must be Oracle Database 12*c* or higher.

Before starting the Data Pump import on the target database, you must enable Oracle Label Security.

13.1.2 Schema and Table-Level Export

The Data Pump export utility functions in the standard way under Oracle Label Security.

There are, however, a few differences resulting from the enforcement of Oracle Label Security policies.

> **Note:**
>
> You must have the `EXEMPT ACCESS POLICY` privilege in order to export all rows in the table, or else no rows are exported.

- For any tables protected by an Oracle Label Security policy, only rows with labels authorized for read access are exported. Unauthorized rows are not included in the export file. Consequently, to export all the data in protected tables, you must have a privilege (such as `FULL` or `READ`) that gives you complete access.
- SQL statements to reapply policies are exported along with tables and schemas that are exported. These statements are carried out during import to reapply policies with the same enforcement options as in the original database.
- The `HIDE` property is not exported. When protected tables are exported, the label columns in those tables are also exported (as numeric values). However, if a label column is hidden, then it is exported as a normal, unhidden column.
- The user must have `EXEMPT ACCESS POLICY` in order to export all rows in the table, or else no rows are exported.

13.2 Data Pump Import Use with Oracle Label Security

Oracle Data Pump enables high-speed movement of data and metadata from one database to another.

- Full Database Import for the LBACSYS Schema Metadata
  Oracle Label Security metadata in the `LBACSYS` schema can be included when you perform a full database export and import operation.
- Schema and Table Level Import
  You can use the Oracle Data Pump Import utility functions under Oracle Label Security.

13.2.1 Full Database Import for the LBACSYS Schema Metadata

Oracle Label Security metadata in the `LBACSYS` schema can be included when you perform a full database export and import operation.

The source database can be Oracle Database 11*g* release 2 (11.2.0.3), or higher, but the target database must be Oracle Database 12*c* release 1 (12.1) or higher.

Oracle Data Pump import utility, `impdp`, automatically imports Label Security metadata including policies, labels, user authorizations, schema and table policy enforcements.

You must register and enable Oracle Label Security for the target database before beginning the import operation.

Related Topics

- Checking if Oracle Label Security Has Been Registered and Enabled
  You can query the `DBA_OLS_STATUS` data dictionary view to find if Oracle Label Security has already been registered and enabled.

13.2.2 Schema and Table Level Import

You can use the Oracle Data Pump Import utility functions under Oracle Label Security.

- Requirements for Import Under Oracle Label Security
  You can use the `impdp` under Oracle Label Security.
- Definition of Data Labels for Import
  The label definitions at the time of import must include all the policy labels used in the export file.
- Imports of Labeled Data Without Installing Oracle Label Security
  When data type for policy label columns is `NUMBER`, they can be imported into databases that do not have Oracle Label Security installed.
- Imports of Unlabeled Data
  You can import unlabeled data into an existing table protected by an Oracle Label Security policy.
- Importing Tables with Hidden Columns
  A hidden column is exported as a normal column, but the fact that it was hidden is lost.

13.2.2.1 Requirements for Import Under Oracle Label Security

You can use the `impdp` under Oracle Label Security.

To use the `impdp` under Oracle Label Security, you must prepare the import database and ensure that the import user has the proper authorizations.

- Preparing the Import Database
  Before you can use the Import utility with Oracle Label Security, you must prepare the import database.
- Verification of Import User Authorizations
  You must be authorized to run the import operation for labels required to insert data and labels in the export file.

13.2.2.1.1 Preparing the Import Database

Before you can use the Import utility with Oracle Label Security, you must prepare the import database.

1. Ensure that Oracle Label Security is enabled. See Checking if Oracle Label Security Has Been Registered and Enabled.
2. Create any Oracle Label Security policies that protect the data to be imported.

   Ensure that the policies use the same column names as in the export database.
3. Define in the import database all of the label components and individual labels used in tables being imported.

Ensure that the same tag values are assigned to the policy labels in each database. (Note that if you are importing into a database from which you exported, then the components are most likely already defined.)

13.2.2.1.2 Verification of Import User Authorizations

You must be authorized to run the import operation for labels required to insert data and labels in the export file.

Errors will be raised upon import if you do not meet the following requirements.

- To import tables or schemas with Label Security policies on them, you must have the `EXECUTE` privilege on the `SA_POLICY_ADMIN` package.

  To ensure that all rows can be imported, you must have the *`policy`*`_DBA` role for all policies with data being imported. After each schema or table is imported, any policies from the export database are reapplied to the imported objects.
- You must also have the ability to write all rows that have been exported as follows:

  Requirement 2:

  – You can granted the `FULL` privilege or given sufficient authorization to write all labels contained in the import file.

  – A user-defined labeling function can be applied to the table.

13.2.2.2 Definition of Data Labels for Import

The label definitions at the time of import must include all the policy labels used in the export file.

The `DBA_SA_LABELS` data dictionary view lists data labels. You can use the views `DBA_SA_LEVELS`, `DBA_SA_COMPARTMENTS`, `DBA_SA_GROUPS`, and in the export database to design SQL scripts that re-create the label components and labels for each policy in the import database. The following example shows how to generate a PL/SQL block that re-creates the individual labels for the `HR` policy:

```
set serveroutput on
BEGIN
   dbms_output.put_line('BEGIN');
   FOR l IN (SELECT label_tag, label
                FROM dba_sa_labels
                WHERE policy_name='HR'
                ORDER BY label_tag) LOOP
       dbms_output.put_line
           ('  SA_LABEL_ADMIN.CREATE_LABEL(''HR'', ' ||
            l.label_tag || ', ''' || l.label || ''');');
   END LOOP;
   dbms_output.put_line ('END;');
   dbms_output.put_line ('/');
END;
/
```

If the individual labels do not exist in the import database with the same numeric values and the same character string representations as in the export database, then the label values in the imported tables will be meaningless. The numeric label value in the table may refer to a different character string representation, or it may be a label value that has not been defined at all in the import database.

If a user attempts to access rows containing invalid numeric labels, then the operation will fail.

13.2.2.3 Imports of Labeled Data Without Installing Oracle Label Security

When data type for policy label columns is `NUMBER`, they can be imported into databases that do not have Oracle Label Security installed.

In this case, the values in the policy label column are imported as numbers. Without the corresponding Oracle Label Security label definitions, the numbers will not reference any specific label.

Note that errors will be raised during the import if Oracle Label Security is not installed, because the SQL statements to reapply the policy to the imported tables and schemas will fail.

13.2.2.4 Imports of Unlabeled Data

You can import unlabeled data into an existing table protected by an Oracle Label Security policy.

Either the `LABEL_DEFAULT` option or a labeling function must be specified for each table being imported, so that the labels for the rows can be automatically initialized as they are inserted into the table.

13.2.2.5 Importing Tables with Hidden Columns

A hidden column is exported as a normal column, but the fact that it was hidden is lost.

If you want to preserve the hidden property of the label column, then you must first create the table in the import database.

1. Before you perform the import, create the table and apply the policy with the `HIDE` option. This adds the policy label column to the table as a hidden column.
2. Remove the policy from the table, so that the enforcement options specified in the export file can be reapplied to the table during the import operation.
3. Perform the import with `IGNORE=Y`. Setting the `IGNORE` parameter to `Y` ignores errors during import.
4. Manually apply the policy to the table with the `HIDE` option.

13.3 SQL*Loader Use with Oracle Label Security

SQL*Loader moves data from external files into tables in Oracle Database.

- Requirements for Using SQL*Loader Under Oracle Label Security
  You can use SQL*Loader with the conventional path to load data into a database protected by Oracle Label Security.
- Oracle Label Security Input to SQL*Loader
  If the policy column for a table is hidden, then you must use the `HIDDEN` keyword to convey this information to SQL*Loader.

13.3.1 Requirements for Using SQL*Loader Under Oracle Label Security

You can use SQL*Loader with the conventional path to load data into a database protected by Oracle Label Security.

Because SQL*Loader performs `INSERT` operations, all of the standard requirements apply when using SQL*Loader on tables protected by Oracle Label Security policies.

13.3.2 Oracle Label Security Input to SQL*Loader

If the policy column for a table is hidden, then you must use the `HIDDEN` keyword to convey this information to SQL*Loader.

To specify row labels in the input file, you must include the policy label column in the `INTO TABLE` clause in the control file.

To load policy labels along with the data for each row, you can specify the `CHAR_TO_LABEL` function or the `TO_DATA_LABEL` function in the SQL*Loader control file.

Table 13-1 shows the variations that you can use when you load Oracle Label Security data with SQL*Loader.

Table 13-1 Input Choices for Oracle Label Security Input to SQL*Loader

Form of Data	Explanation of Results
`col1 hidden integer external`	Hidden column loaded with tag value of data directly from data file
`col2 hidden char(5) "func(:col2)"`	Hidden column loaded with character value of data from data `file.func()` used to translate between the character label and its tag value. Note: `func()` might be `char_to_label()`.
`col3 hidden "func(:col3)"`	Same as in `col2`, field type defaults to **char**
`col4 hidden expression "func(:col4)"`	Hidden column not mapped to input `data.func()` will be called to provide the label value. This could be a user function.

For example, the following is a valid `INTO TABLE` clause in a control file that is loading data into the `DEPT` table:

```
INTO TABLE dept
(hr_label HIDDEN POSITION (1:22) CHAR "CHAR_TO_LABEL('HR',:hr_label)",
deptno    POSITION (23:26) INTEGER EXTERNAL,
dname     POSITION (27:40) CHAR,
loc       POSITION(41,54)  CHAR)
```

The following could be an entry in the data file specified by this control file:

```
HS:FN                 231 ACCOUNTING  REDWOOD SHORES
```

13.4 Performance Tips for Oracle Label Security

You can achieve optimal performance with Oracle Label Security.

- Use of ANALYZE to Improve Oracle Label Security Performance
  You can run the `ANALYZE` statement on the Oracle Label Security data dictionary tables in the `LBACSYS` schema.
- Creation of Indexes on the Policy Label Column
  Creating the appropriate type of index on the policy label column improves the performance of user-raised queries on protected tables.
- Label Tag Strategy Plan to Enhance Performance
  For optimal performance, you can plan a strategy for assigning values to label tags.
- Partitioned Data Based on Numeric Label Tags
  Using a numeric ordering strategy with the numeric label tags applied to the labels can a basis for Oracle Database data partitioning.

13.4.1 Use of ANALYZE to Improve Oracle Label Security Performance

You can run the `ANALYZE` statement on the Oracle Label Security data dictionary tables in the `LBACSYS` schema.

This enables the cost-based optimizer to improve execution plans on queries, which improves Oracle Label Security performance.

Running `ANALYZE` on application tables improves the application SQL performance.

See Also:

Oracle Database SQL Language Reference for the `ANALYZE` syntax

13.4.2 Creation of Indexes on the Policy Label Column

Creating the appropriate type of index on the policy label column improves the performance of user-raised queries on protected tables.

If you have applied an Oracle Label Security policy on a database table in a particular schema, then you should compare the number of different labels to the amount of data. Based on this information, you can decide which type of index to create on the policy label column.

- If the cardinality of data in the policy label column (that is, the number of labels compared to the number of rows) is low, then consider creating a bitmapped index.
- If the cardinality of data in the policy label column is high, then consider creating a B-tree index.

Consider the following case, in which the `EMP` table is protected by an Oracle Label Security policy with the `READ_CONTROL` enforcement option set, and `HR_LABEL` is the name of the policy label column. A user raises the following query:

```
SELECT COUNT (*) FROM SCOTT.EMP;
```

In this situation, Oracle Label Security adds a predicate based on the label column. For example:

```
SELECT COUNT (*) FROM SCOTT.EMP
  WHERE hr_label=100;
```

In this way, Oracle Label Security uses the security label to restrict the rows that are processed, based on the user's authorizations. To improve performance of this query, you could create an index on the `HR_LABEL` column.

Consider a more complex query (once again, with `READ_CONTROL` applied to the EMP table):

```
SELECT COUNT (*) FROM SCOTT.EMP
  WHERE deptno=10
```

Again, Oracle Label Security adds a predicate based on the label column:

```
SELECT COUNT (*) FROM SCOTT.EMP
  WHERE deptno=10
  AND hr_label=100;
```

In this case, you might want to create a composite index based on the `DEPTNO` and `HR_LABEL` columns, to improve application performance.

13.4.3 Label Tag Strategy Plan to Enhance Performance

For optimal performance, you can plan a strategy for assigning values to label tags.

In general, it is best to assign higher numeric values to labels with higher sensitivity levels.

This is because, typically, many more users can see data at comparatively low levels and fewer users at higher levels can see many levels of data.

In addition, with `READ_CONTROL` set, Oracle Label Security generates a predicate that uses a `BETWEEN` clause to restrict the rows to be processed by the query. As illustrated in the following example, if the higher-sensitivity labels do not have a higher label tag than the lower-sensitivity labels, then the query will potentially examine a larger set of rows. This will affect performance.

Table 13-2 shows a set of label tags assigned as follows:

Table 13-2   Label Tag Performance Example: Correct Values

Label	Label Tag
`TS:A,B`	`100`
`S:A`	`50`
`S`	`20`
`U:A`	`10`

Here, a user whose maximum authorization is S:A can potentially access data at labels `S:A`, `S`, and `U:A`. Consider what happens when this user raises the following query:

```
SELECT COUNT (*) FROM SCOTT.EMP
```

Oracle Label Security adds a predicate that includes a `BETWEEN` clause (based on the maximum and minimum authorizations) to restrict the set of rows this user can see:

```
SELECT COUNT (*) FROM SCOTT.EMP
  WHERE hr_label BETWEEN 10 AND 50;
```

Performance improves, because the query examines only a subset of data based on the user's authorizations. It does not fruitlessly process rows that the user is not authorized to access.

Table 13-3 shows how unnecessary work is performed if the tag values were assigned as follows:

Table 13-3 Label Tag Performance Example: Incorrect Values

Label	Label Tag
`TS:A,B`	`50`
`S:A`	`100`
`S`	`20`
`U:A`	`10`

In this case, the user with `S:A` authorization can see only some of the labels between 100 and 10. Although the user cannot see `TS:A,B` labels (that is, rows with a label tag of 50). A query would nonetheless pick up and process these rows, even though the user ultimately will not have access to them.

13.4.4 Partitioned Data Based on Numeric Label Tags

Using a numeric ordering strategy with the numeric label tags applied to the labels can a basis for Oracle Database data partitioning.

Depending on the application, partitioning data based on label values may or may not be useful. Consider, for example, a business-hosting CRM application to which many companies subscribe. In the same `EMP` table, there might be rows (and labels) for Subscriber 1 and Subscriber 2. That is, information for both companies can be stored in the same table, as long as it is labeled differently. In this case, employees of Subscriber 1 will never need to access data for Subscriber 2, so it might make sense to partition based on label. You could put rows for Subscriber 1 in one partition, and rows for Subscriber2 in a different partition. When a query is raised, it will access only one or the other partition, depending on the label. Performance improves because partitions that are not relevant are not examined by the query.

The following example shows this is done. It places labels in the 2000 series on one partition, labels in the 3000 series on another partition, and labels in the 4000 series on a third partition.

```
CREATE TABLE EMPLOYEE(
    EMPNO NUMBER(10) CONSTRAINT PK_EMPLOYEE PRIMARY KEY,
    ENAME VARCHAR2(10),
    JOB VARCHAR2(9),
    MGR NUMBER(4),
    HIREDATE DATE,
    SAL NUMBER(7,2),
    COMM NUMBER(7,2),
    DEPTNO NUMBER(4),
    HR_LABEL NUMBER(10))
    TABLESPACE PERF_DATA
    STORAGE (initial 2M
    NEXT 1M
    MINEXTENTS 1
    MAXEXTENTS unlimited)
    PARTITION BY RANGE (hr_label)
    (partition sx1 VALUES LESS THAN (2000) NOLOGGING,
     partition sx2 VALUES LESS THAN (3000),
     partition sx3 VALUES LESS THAN (4000)
);
```

13.5 Creation of Additional Databases After Installation

You can create and configure additional databases after you install Oracle Label Security.

- About the Creation of Additional Databases After Installation
  When you install Oracle Database Enterprise Edition and Oracle Label Security, an initial Oracle database is created.
- Creating Additional Databases When the Label Security Schema Is in the Seed
  You can configure Oracle Label Security if the database was installed with the label security schema is in the seed database.
- Creating Additional Databases with the Custom Installation Option
  You can configure Oracle Label Security after a custom database installation.

13.5.1 About the Creation of Additional Databases After Installation

When you install Oracle Database Enterprise Edition and Oracle Label Security, an initial Oracle database is created.

If you want to create additional databases, then you should do this using the Database Configuration Assistant. Alternatively, you can create additional databases by following the steps listed in *Oracle Database Administrator's Guide*.

Each time you create a new database, you must install the Oracle Label Security data dictionary tables, views, and packages into it, and create the `LBACSYS` account.

For the first database, this is done automatically when you install Oracle Label Security, regardless of whether or not you choose the custom install. If you do not choose the custom install, then you are installing the database with the label security schema in the seed.

To create additional databases, there are different processes for configuring label security, depending on whether the first database was installed with the custom install or with the label security schema in the seed.

If you initially chose custom install, but did not install label security, you can install and configure label security using either process described in this section.

13.5.2 Creating Additional Databases When the Label Security Schema Is in the Seed

You can configure Oracle Label Security if the database was installed with the label security schema is in the seed database.

1. Select the Oracle Label Security option in DBCA.
2. Select the check box to configure Oracle Label Security.

13.5.3 Creating Additional Databases with the Custom Installation Option

You can configure Oracle Label Security after a custom database installation.

1. Connect to the Oracle Database instance as user `SYS`, using the `AS SYSDBA` syntax.
2. Run the script `$ORACLE_HOME/rdbms/admin/catols.sql`.

   This script installs the label-based framework, data dictionary, data types, and packages. After the script is run, the `LBACSYS` account exists, with the password `LBACSYS`. All the Oracle Label Security packages exist under this account.
3. Change the default password of the `LBACSYS` user.
4. Change this password to a secure password.

13.6 Oracle Label Security Upgrades and Downgrades

You should be aware of how to manage Oracle Label Security upgrades and downgrades.

- About Oracle Label Security Upgrades and Downgrades
  Oracle provides preprocess scripts that perform upgrade and downgrade operations.
- Oracle Label Security Upgrades
  Oracle provides a preprocess script that you must run before you perform an upgrade.
- Oracle Label Security Downgrades
  Oracle provides a preprocess script that you must run before you downgrade.

13.6.1 About Oracle Label Security Upgrades and Downgrades

Oracle provides preprocess scripts that perform upgrade and downgrade operations.

As a safety measure, before you run either the upgrade or downgrade preprocess script, Oracle recommends that you back up your audit records. To do this, you can archive the audit trail as described in *Oracle Database Security Guide*.

Before they run, the preprocess scripts check that there is enough space in the audit tablespace to copy all the audit records, and will exit without processing if there is not.

You may continue running your applications on the database while OLS preprocess scripts are running.

See Also:

Oracle Database Upgrade Guide for requirements for upgrading databases that use Oracle Label Security and Oracle Database Vault

13.6.2 Oracle Label Security Upgrades

Oracle provides a preprocess script that you must run before you perform an upgrade.

- About Oracle Label Security Upgrades
  You must upgrade Oracle Label Security for pre-Oracle Database 12c release 1 (12.1) databases.
- Running the Oracle Label Security Preprocess Script Before Upgrading
  You can run the Oracle Label Security preprocess script before upgrading.

13.6.2.1 About Oracle Label Security Upgrades

You must upgrade Oracle Label Security for pre-Oracle Database 12c release 1 (12.1) databases.

> **Note:**
>
> Running the `olspreupgrade.sql` script before upgrading is mandatory for upgrading databases earlier than Oracle Database 12c release (12.1) that use Oracle Label Security or Database Vault.
>
> After you have upgraded to Oracle Database release 12c or later, you do not need to run the Oracle Label Security preprocessing script when you patch or upgrade the database.

Before performing the OLS upgrade process, you must run the Oracle Label Security preprocess upgrade script, `olspreupgrade.sql`, to process the `AUD$` table contents. The OLS upgrade moves the `AUD$`table from the `SYSTEM` schema to the `SYS` schema. The `olspreupgrade.sql` script is a preprocessing script required for this move. It creates a temporary table, `PREUPG_AUD$`, in the `SYS` schema and moves the `SYSTEM.AUD$` records to `SYS.PREUPG_AUD$`. The moved records can no longer be viewed through the `DBA_AUDIT_TRAIL` view, but can be viewed by directly accessing the `SYS.PREUPG_AUD$` table, until the upgrade completes. Once the upgrade completes, the `SYS.PREUPG_AUD$` table is permanently deleted and all audit records, can be viewed through the `DBA_AUDIT_TRAIL` view.

13.6.2.2 Running the Oracle Label Security Preprocess Script Before Upgrading

You can run the Oracle Label Security preprocess script before upgrading.

1. Copy the *`ORACLE_HOME`*`/rdbms/admin/olspreupgrade.sql` script from the newly installed Oracle home to the Oracle home of the database to be upgraded.
2. Connect to the database to be upgraded. At the system prompt, enter:

   ```
   CONNECT SYS AS SYSDBA
   Enter password password
   ```
3. Run the Oracle Label Security preprocess script:

   ```
   @$ORACLE_HOME/rdbms/admin/olspreupgrade.sql
   ```

Note:

The upgrade status for the Oracle Label Security component will be marked `INVALID` if the Oracle Label Security preprocess script reports an error. If this happens, you must correct the errors and then rerun the upgrade process. See *Oracle Database Upgrade Guide* for more information about rerunning the upgrade process for Oracle Database.

13.6.3 Oracle Label Security Downgrades

Oracle provides a preprocess script that you must run before you downgrade.

- About Oracle Label Security Downgrades
  You can downgrade from an Oracle Database 12*c* release 1 (12.1) or later database that uses Oracle Label Security or Oracle Database Vault.
- Running the Oracle Label Security Preprocess Script Before Downgrading
  You must connect as `SYS` wth the `SYSDBA` administrative privilege before running the Oracle Label Security preprocess script for a downgrade.

13.6.3.1 About Oracle Label Security Downgrades

You can downgrade from an Oracle Database 12*c* release 1 (12.1) or later database that uses Oracle Label Security or Oracle Database Vault.

To do this, you must run the OLS preprocessing script, `olspredowngrade.sql` to process the `AUD$` table contents. The OLS downgrade script moves the `AUD$` table from the `SYS` schema to the `SYSTEM` schema. The `olspredowngrade.sql` script is a processing script required in preparation for this move. It creates a temporary table, `PREDWG_AUD$`, in the `SYSTEM` schema and moves the `SYS.AUD$` records to `SYSTEM.PREDWG_AUD$`. The moved records can no longer be viewed through the `DBA_AUDIT_TRAIL` view, but you can view them by directly accessing the `SYSTEM.PREDWG_AUD$` table until the downgrade completes. Once the downgrade completes, the `SYSTEM.PREDWG_AUD$` table is permanently deleted. At this point, all audit records are available for viewing in the `DBA_AUDIT_TRAIL` data dictionary view.

13.6.3.2 Running the Oracle Label Security Preprocess Script Before Downgrading

You must connect as `SYS` wth the `SYSDBA` administrative privilege before running the Oracle Label Security preprocess script for a downgrade.

1. Connect to the database to be downgraded. At the system prompt, enter:

   ```
   CONNECT SYS AS SYSDBA
   Enter password password
   ```

2. Run the OLS preprocess downgrade script:

   ```
   @$ORACLE_HOME/rdbms/admin/olspredowngrade.sql
   ```

14
Releasability Using Inverse Groups

Oracle Label Security can implement the releasability using inverse groups.

- About Inverse Groups and Releasability
  Inverse groups indicate *releasability* of information.
- Comparison of Standard Groups and Inverse Groups
  Groups in Oracle Label Security identify organizations that own or access data.
- How Inverse Groups Work
  Inverse groups are implemented in a special way and are organized to suit the needs of Oracle Label Security.
- Algorithm for Read Access with Inverse Groups
  You should understand how the algorithm for read access with inverse groups works.
- Algorithm for Write Access with Inverse Groups
  You should understand the algorithm for write access with inverse groups.
- Algorithms for COMPACCESS Privilege with Inverse Groups
  Oracle provides algorithms for read and write access with inverse groups, for users who have `COMPACCESS` privilege.
- Session Labels and Inverse Groups
  Inverse groups affect session labels and row labels.
- Changes in Behavior of Procedures with Inverse Groups
  The `INVERSE_GROUP` option affects algorithms that determine the read and write access of the user to labeled data.
- Dominance Rules for Labels with Inverse Groups
  You should understand how dominance rules work for Oracle labels and inverse groups.

14.1 About Inverse Groups and Releasability

Inverse groups indicate *releasability* of information.

They are used to mark the dissemination of data. When you add an inverse group to a data label, the data becomes less classified.

For example, a user with inverse groups UK and US cannot access data that only has inverse group UK. Adding US to that data makes it accessible to all users with the inverse groups UK and US.

When you assign releasabilities to a user, you mark the communication channel to the user. For data to flow across the communication channel, the data releasabilities must dominate the releasabilities assigned to the user. In other words, releasabilities assigned to a data record must contain all the releasabilities assigned to a user.

The advantage of releasabilities lies in their power to broadly disseminate information. Releasing data to the entire marketing organization becomes as simple as adding the Marketing releasability to the data record.

14.2 Comparison of Standard Groups and Inverse Groups

Groups in Oracle Label Security identify organizations that own or access data.

Like standard groups, inverse groups control the dissemination of information. However, the behavior of inverse groups differs from Oracle Label Security standard group behavior. By default, all policies created in Oracle Label Security use the standard group behavior.

The term, *releasabilities* is sometimes used to refer to the behavior provided by inverse groups. When you include inverse groups in a data label, the effect is similar to assigning label compartment authorizations to a user. When Oracle Label Security evaluates whether a user can view a row of data assigned to a label with inverse groups, it checks to see whether the data, not the user, has the appropriate group authorizations. It checks whether the data has *all* the inverse groups assigned to the user. With standard groups, by contrast, Oracle Label Security checks to see whether a user is authorized for *at least one* of the groups assigned to a row of data.

Consider a policy that contains three standard groups such as, Eastern, Western, and Southern. User1's label authorizations include the groups Eastern and Western. Assuming that User1 has been assigned the appropriate level and compartment authorizations in the policy, then:

- With standard Oracle Label Security groups, User1 can view *all* data records that have the group Eastern, or the group Western, or both Eastern and Western.
- With inverse groups, User1 can only view data records that have, *at a minimum, all* the groups assigned to the user, that is, both Eastern and Western. User1 *cannot* view records that have only the Eastern group, only the Western group, or that have no groups at all.

Table 14-1 shows all the rows that User1 can potentially access, given the type of group that is used in the policy.

Table 14-1 Access to Standard Groups and Inverse Groups

If row label contains groups:	User1 access with standard groups?	User1 access with inverse groups?
None	Y	N
`Eastern`	Y	N
`Western`	Y	N
`Southern`	N	N
`Eastern, Western`	Y	Y
`Eastern, Southern`	Y	N
`Western, Southern`	Y	N
`Eastern, Western, Southern`	Y	Y

Standard groups indicate *ownership* of information. In this way, all data pertaining to a certain department can have that department's group in the label. When you add a group to a data label, the data becomes more classified. For example, a user with no groups can access data that has no groups in its label. If you add the group US to the data label, the user can no longer access the data.

See Also:

Group Components

14.3 How Inverse Groups Work

Inverse groups are implemented in a special way and are organized to suit the needs of Oracle Label Security.

- Implementation of Inverse Groups with INVERSE_GROUP Enforcement
  When creating an Oracle Label Security policy, you can specify whether the policy can use inverse group functionality to implement releasability.
- Inverse Groups and Label Components
  An Oracle Label Security policy created with the inverse group option uses the same policy label components as standard groups.
- Computed Labels with Inverse Groups
  Inverse groups affect computed label values.
- Inverse Groups and Hierarchical Structure
  Standard groups in Oracle Label Security are hierarchical, so that a group can be associated with a parent group.
- Inverse Groups and User Privileges
  With inverse groups implemented, the meaning of user privileges remains the same.

14.3.1 Implementation of Inverse Groups with INVERSE_GROUP Enforcement

When creating an Oracle Label Security policy, you can specify whether the policy can use inverse group functionality to implement releasability.

To do this, you must specify `INVERSE_GROUP` as one of the `default_options` in the `CREATE_POLICY` statement.

The `INVERSE_GROUP` option can be set only at policy creation time. Once a policy is created, this option cannot be changed.

The `INVERSE_GROUP` option is thus policywide. It cannot be turned on or off when the policy is applied to a table or schema. If you attempt to do so, using the procedure `APPLY_TABLE_POLICY` or `APPLY_SCHEMA_POLICY`, then an error will be generated.

While other policy enforcement options can be dropped from a policy, the `INVERSE_GROUP` policy configuration option cannot be dropped once it is set. To remove the option, you must drop and then re-create the policy.

You can give individual users authorization for one or more inverse groups.

14.3.2 Inverse Groups and Label Components

An Oracle Label Security policy created with the inverse group option uses the same policy label components as standard groups.

These components include levels, compartments, and groups.

With inverse groups, however, the user's read groups and write groups have a different meaning and role in data access.

Consider the following policy example, with three levels, one compartment, and three groups:

Table 14-2 Policy Example

Policy Component	Abbreviation
Levels:	-
`UNCLASSIFIED`	`UN`
`CONFIDENTIAL`	`CON`
`SECRET`	`SE`
Compartments:	-
`FINANCIAL`	`FIN`
Groups:	-
`EASTERN`	`EAS`
`WESTERN`	`WES`
`SOUTHERN`	`SOU`

Two user labels have been assigned, `CON:FIN` and `SE:FIN:EAS,WES`

Two data labels have been assigned, `CON:FIN:EAS` and `SE:FIN:EAS`

User access to the data differs, depending on the type of group being used:

- If the policy uses standard groups, then:

  The user with the label `CON:FIN` *cannot* read `CON:FIN:EAS` data.

  The user with the label `SE:FIN:EAS,WES` *can* read `SE:FIN:EAS` data.
- If the policy has the `INVERSE GROUPS` policy enforcement option, then:

  The user with the label `CON: FIN` *can* read `CON:FIN:EAS` data.

  The user with the label `SE:FIN:EAS,WES` *cannot* read `SE:FIN:EAS` data.

14.3.3 Computed Labels with Inverse Groups

Inverse groups affect computed label values.

- Computed Session Labels with Inverse Groups
  After the administrator assigns label authorizations to a user, Oracle Label Security automatically computes a number of labels.
- Inverse Groups and Computed Max Read Groups and Max Write Groups
  Oracle Label Security provides different inverse groups to handle read and write operations.

14.3.3.1 Computed Session Labels with Inverse Groups

After the administrator assigns label authorizations to a user, Oracle Label Security automatically computes a number of labels.

With inverse groups, these labels are as follows:

Table 14-3 Computed Session Labels with Inverse Groups

Computed Label	Definition
Max Read Label	The user's maximum level combined with their authorized compartments and the minimum set of inverse groups that should be in the user label (session label)
Max Write Label	The user's maximum level combined with the compartments for which the user has been granted write access. Contains the maximum authorized inverse groups that can be set in any label. The user has write authorizations on all these inverse groups.
Min Write Label	The user's minimum level.
Default Read Label	The default level, combined with compartments and inverse groups that have been designated as default for the user.
Default Write Label	A subset of the default read label, containing the compartments and inverse groups for which the user has been granted write access. However the inverse groups component has no significance as it is the Max Write Groups that is always used for write access.
Default Row Label	The combination of components between the user's minimum write label and the maximum write label, which has been designated as the default for the data label for inserted data. The Inverse groups should be a superset of inverse groups in the default label and a subset of Max Write Groups.

Related Topics

- Computed Session Labels
  Oracle Label Security automatically computes a number of labels based on the value of the session label.

14.3.3.2 Inverse Groups and Computed Max Read Groups and Max Write Groups

Oracle Label Security provides different inverse groups to handle read and write operations.

From the computed values in Table 14-3, two sets of groups are identified for label evaluation of read and write access.

Table 14-4 Sets of Groups for Evaluating Read and Write Access

Sets of Groups	Meaning
Max Read Groups	Max Read Groups are the groups contained in the Max Read Label, identifying the *minimum* set of inverse groups that can be set in any user label.

Table 14-4 (Cont.) Sets of Groups for Evaluating Read and Write Access

Sets of Groups	Meaning
Max Write Groups	Max Write Groups are the groups contained in the Max Write Label, identifying the *maximum* authorized inverse groups that can be set in any user label. This set of groups is checked at the time of write access, and also when setting session labels. Note that Max Write Groups is a superset of Max Read Groups.

As shown in Table 14-5, for standard groups you can have `READ ONLY` and `READ`/`WRITE` authorizations; for inverse groups you can have `WRITE ONLY` and `READ`/`WRITE` authorizations.

Table 14-5 Read and Write Authorizations for Standard Groups and Inverse Groups

Type of Group	READ ONLY	READ/WRITE	WRITE ONLY
Standard Groups	The group is present only in Max Read Label, not in Max Write Label.	The group is present in both Max Read Label and Max Write Label.	Not supported
Inverse Groups	Not supported	The group is present in both Max Read Label and Max Write Label.	The group is present only in Max Write Label, not in Max Read Label.

Although Max Read Groups identifies the set of groups contained in the Max Read Label, this value represents the *minimum* set of inverse groups that can be set. For example:

Max Read Groups: `S:C1:G1`,`G2`

Max Write Groups: `S:C1:G1`,`G2`,`G3`,`G4`,`G5`

Here, the user can read data that contains at least the two groups listed in Max Read Groups.

Note that in standard groups, there can never be a situation in which there are more groups in the Max Write Label than in the Max Read Label.

14.3.4 Inverse Groups and Hierarchical Structure

Standard groups in Oracle Label Security are hierarchical, so that a group can be associated with a parent group.

For example, the `EASTERN` region can be the parent of two subordinate groups: `EAS_SALES`, and `EAS_HR`.

In a policy with standard groups, if the user label has the parent group, then it can access all data of the subordinate groups.

With inverse groups, parent-child relationships are not supported.

14.3.5 Inverse Groups and User Privileges

With inverse groups implemented, the meaning of user privileges remains the same.

When the user has no special privileges, then the read algorithm and the write algorithm are different for standard groups and inverse groups. The differences are described later, in Algorithm for Read Access with Inverse Groups and Algorithm for Write Access with Inverse Groups.

The effect of inverse groups on the COMPACCESS privilege is described later, in Algorithms for COMPACCESS Privilege with Inverse Groups.

Inverse groups have no impact upon the following user privileges:

- `PROFILE_ACCESS`
- `WRITEUP`
- `WRITEDOWN`
- `WRITEACROSS`

14.4 Algorithm for Read Access with Inverse Groups

You should understand how the algorithm for read access with inverse groups works.

To read data in a table with the `INVERSE GROUP` option in effect, the label evaluation process proceeds from levels to groups to compartments, as illustrated in Figure 14-1. (Note that the current session label is the label being evaluated.)

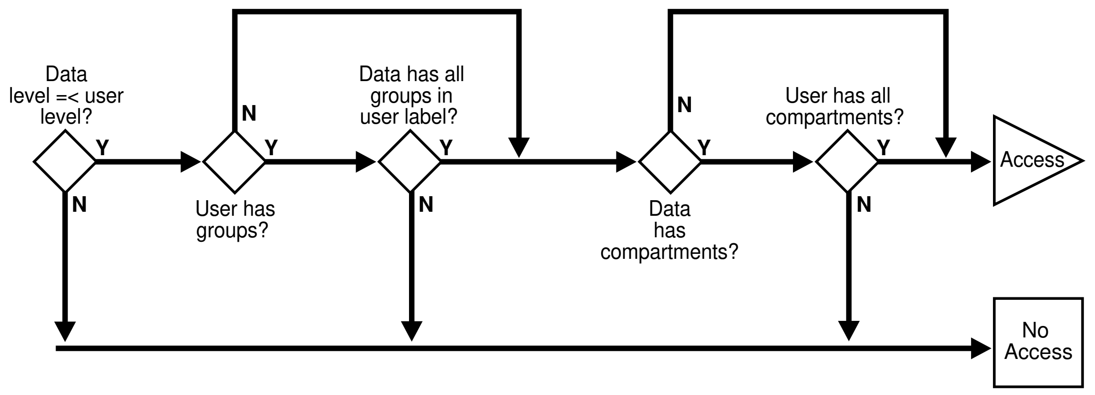
1. The user's level must be greater than or equal to the level of data.
2. The user's label must include all the compartments assigned to the data
3. The groups in the data label must be a superset of the groups in the user label.

If the user's label passes these tests, then the user can access the data. If not, the user is denied access. Note that if the data label is null or invalid, then the user is denied access.

Note:

This flow diagram is true only when the user has no special privileges.

Figure 14-1 Read Access Label Evaluation with Inverse Groups


Data level =< user level?
Y
N
User has groups?
N
Y
Data has all groups in user label?
Y
N
Data has compartments?
N
Y
User has all compartments?
Y
N
Access
No Access

Related Topics

- How Oracle Label Security Algorithm for Read Access Works
  The `READ_CONTROL` enforcement determines the ability to read data in a row.

14.5 Algorithm for Write Access with Inverse Groups

You should understand the algorithm for write access with inverse groups.

To write data in a table with the `INVERSE GROUP` option, the label evaluation process proceeds from levels to groups to compartments, as illustrated in Figure 14-2. (Note that the current session label is the label being evaluated.)

1. The level in the data label must be greater than or equal to the user's minimum level, and less than or equal to the user's session level.
2. One of the following conditions must be met:

   The groups in the data label must be a superset of the groups in the user label.

   or

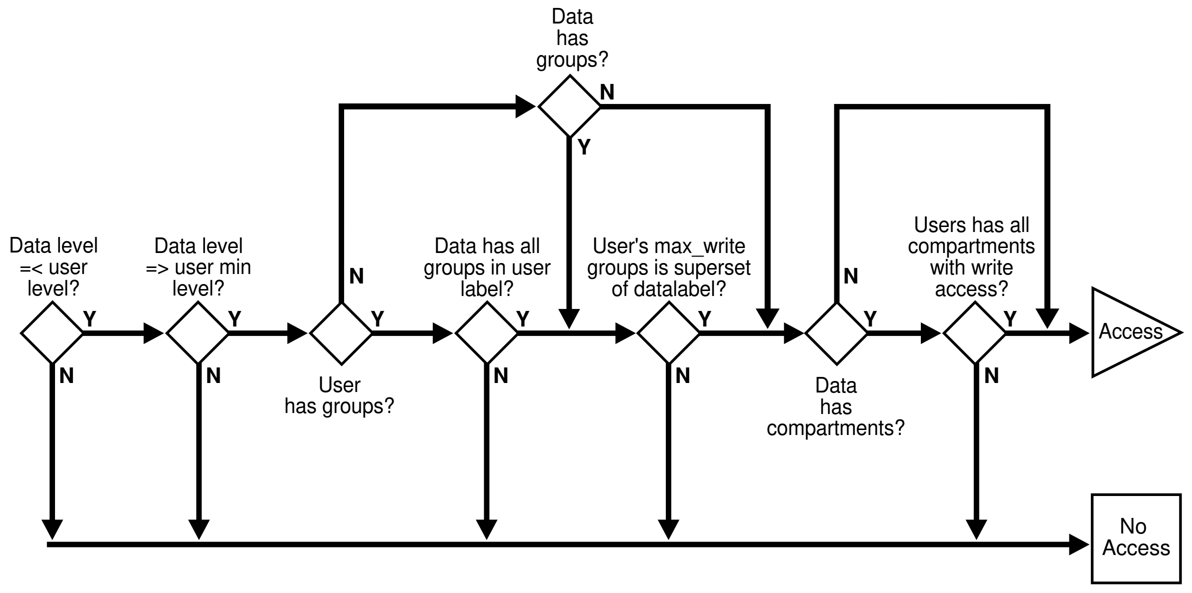
   The user has `READ` access privilege on the policy.
3. The user's Max Write Groups must be a superset of the data label groups.
4. The user label must have write access on all of the compartments in the data label.

Note that if the data label is null or invalid, then the user is denied access.

Note:

This flow diagram is true only when the user has no special privileges.

Figure 14-2 Write Access Label Evaluation with Inverse Groups


Data has groups?
N
Y
Data level =< user level?
Data level => user min level?
N
Data has all groups in user label?
User's max_write groups is superset of datalabel?
N
Users has all compartments with write access?
Y
Y
Y
Y
Y
Y
Y
Access
N
N
User has groups?
N
N
N
Data has compartments?
N
No Access

> **See Also:**
>
> How the Oracle Label Security Algorithm for Write Access Works

14.6 Algorithms for COMPACCESS Privilege with Inverse Groups

Oracle provides algorithms for read and write access with inverse groups, for users who have `COMPACCESS` privilege.

The `COMPACCESS` privilege allows a user to access data based on the row's compartments, independent of the row's groups.

- When compartments exist and access to them is authorized, then the group authorization is bypassed.
- If a row has no compartments, then access is determined by the inverse group authorizations.

Figure 14-3 and Figure 14-4 show the label evaluation process for read access and write access for a user with the `COMPACCESS` privilege. If the data label is null or invalid, then the user is denied access.

(Note that the current session label is the label being evaluated.)

Figure 14-3 Read Access Label Evaluation: COMPACCESS Privilege and Inverse Groups

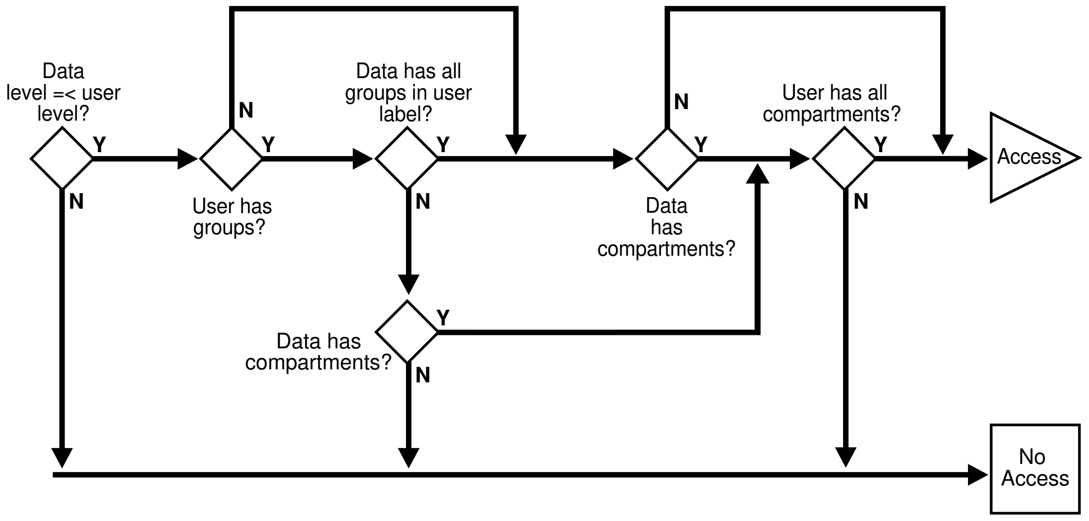
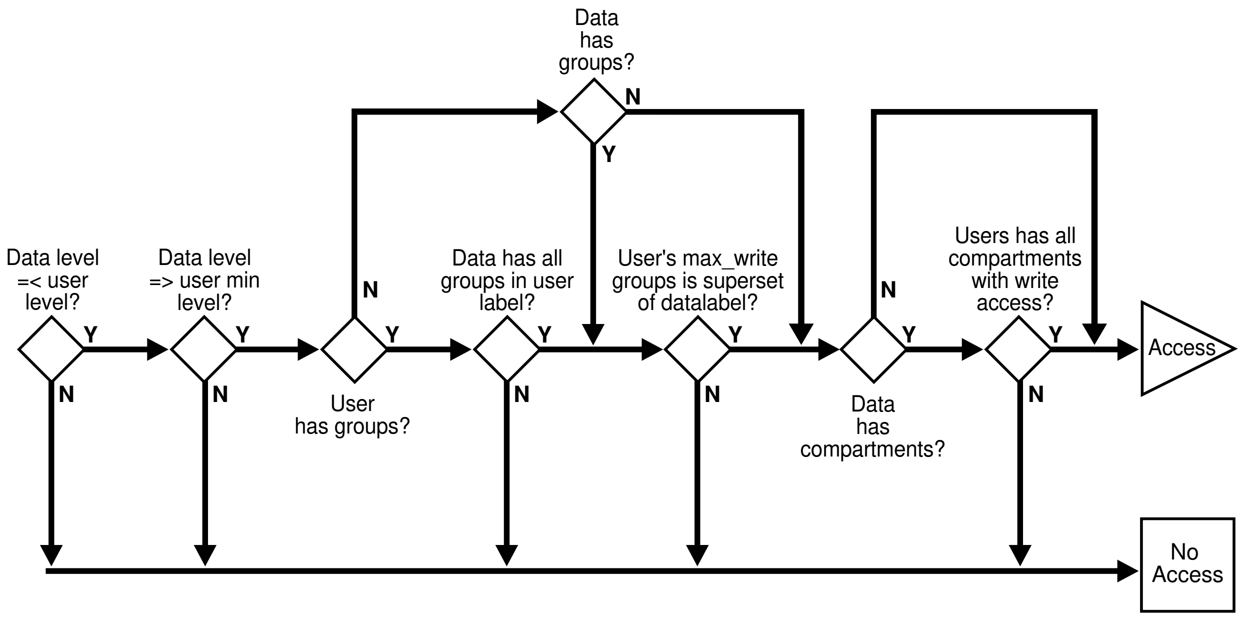


Figure 14-4 Write Access Label Evaluation: COMPACCESS Privilege and Inverse Groups


Data level =< user level?
Y
N
Data level => user min level?
Y
N
User has groups?
N
Y
Data has groups?
N
Y
Data has all groups in user label?
Y
N
User's max_write groups is superset of datalabel?
Y
N
Data has compartments?
N
Y
Users has all compartments with write access?
Y
N
Access
No Access

14.7 Session Labels and Inverse Groups

Inverse groups affect session labels and row labels.

- Initial Session and Row Labels for Standard or Inverse Groups
  Oracle provides initial session and row labels for standard and inverse groups.
- Setting Current Session or Row Labels for Standard or Inverse Groups
  You can set the current session or row labels for standard or inverse groups.
- Examples of Session Labels and Inverse Groups
  Oracle provides examples of using inverse groups.

14.7.1 Initial Session and Row Labels for Standard or Inverse Groups

Oracle provides initial session and row labels for standard and inverse groups.

- About the Initial Session and Row Labels for Standard or Inverse Groups
  The use of inverse groups affects the behavior of Oracle Label Security procedures that determine the session label.
- Standard Groups: Rules for Changing Initial Session/Row Labels
  A user's default session label can be changed using `SA_USER_ADMIN.SET_DEFAULT_LABEL`.
- Inverse Groups: Rules for Changing Initial Session/Row Labels
  The default session label can include groups in the authorized list if the new write label dominates the current default row label.

14.7.1.1 About the Initial Session and Row Labels for Standard or Inverse Groups

The use of inverse groups affects the behavior of Oracle Label Security procedures that determine the session label.

The `SA_USER_ADMIN.SET_DEFAULT_LABEL` and `SA_USER_ADMIN.SET_ROW_LABEL` procedures set the user's initial session label and row label, respectively, to the one specified.

14.7.1.2 Standard Groups: Rules for Changing Initial Session/Row Labels

A user's default session label can be changed using `SA_USER_ADMIN.SET_DEFAULT_LABEL`.

In the case of standard groups, the default session label can be set to include any groups in the authorized list, as long as the current default row label will still be dominated by the new write label. That is, the row label will have *the same or fewer standard groups* than the new write label.

The same rule applies for `SA_USER_ADMIN.SET_ROW_LABEL`.

14.7.1.3 Inverse Groups: Rules for Changing Initial Session/Row Labels

The default session label can include groups in the authorized list if the new write label dominates the current default row label.

That is, the row label will have *the same or more inverse groups* than the new write label. The same rule applies for `SA_USER_ADMIN.SET_ROW_LABEL`.

Related Topics

- SA_USER_ADMIN.SET_DEFAULT_LABEL
  The `SA_USER_ADMIN.SET_DEFAULT_LABEL` procedure sets the user's initial session label to the one specified.
- SA_USER_ADMIN.SET_ROW_LABEL
  The `SA_USER_ADMIN.SET_ROW_LABEL` procedure sets a user's initial row label to the one specified.
- Dominance Rules for Labels with Inverse Groups
  You should understand how dominance rules work for Oracle labels and inverse groups.

14.7.2 Setting Current Session or Row Labels for Standard or Inverse Groups

You can set the current session or row labels for standard or inverse groups.

- About Setting Current Session or Row Labels for Standard or Inverse Groups
  The use of inverse groups affects the behavior of the `SA_SESSION.SET_LABEL` and `SA_SESSION.SET_ROW_LABEL` procedures.
- Standard Groups: Rules for Changing Current Session/Row Labels
  With standard groups, the `SA_SESSION.SET_LABEL` procedure can set the session label to include groups in the user's authorized group list.
- Inverse Groups: Rules for Changing Current Session/Row Labels
  With inverse groups, the addition of groups to the session label *decreases* a user's ability to access sensitive data with fewer groups.

14.7.2.1 About Setting Current Session or Row Labels for Standard or Inverse Groups

The use of inverse groups affects the behavior of the `SA_SESSION.SET_LABEL` and `SA_SESSION.SET_ROW_LABEL` procedures.

These procedures can be used to set the user's current session label and row label, respectively.

14.7.2.2 Standard Groups: Rules for Changing Current Session/Row Labels

With standard groups, the `SA_SESSION.SET_LABEL` procedure can set the session label to include groups in the user's authorized group list.

Subgroups of authorized groups are implicitly included in the authorized list.

Note that if you change the session label, then this may affect the value of the session's row label.

Use the `SET_ROW_LABEL` procedure to set the row label value for the current database session. The compartments and groups in the label must be a subset of compartments and groups in the session label to which the user has write access.

14.7.2.3 Inverse Groups: Rules for Changing Current Session/Row Labels

With inverse groups, the addition of groups to the session label *decreases* a user's ability to access sensitive data with fewer groups.

The removal of groups enables the user to access *more* sensitive information. So, the user should be allowed to add groups to the session label, as long as Max Read Groups is a subset of the groups in the session label, and Max Write Groups is a superset of groups in the session label. The same restriction applies when a user removes groups from the session label.

Note that there are no subgroups of authorized groups when using inverse groups. This is because parent groups are not allowed in policies using inverse groups.

Use the `SET_ROW_LABEL` procedure to set the row label value for the current database session. The compartments in the label must be a subset of compartments in the session label to which the user has write access.

The user is allowed to add inverse groups to the row label, as long as the session label inverse groups are a subset of the row label inverse groups, and Max Write Groups is a superset of inverse groups in the row label.

For example:

- If the user has the inverse groups UK and US as their Max Read Groups, and `UK,US,CAN` as their Max Write Groups. The user can set their session label to `C:ALPHA:UK,US,CAN` but not to `C:ALPHA:UK`.
- If the user has the inverse group `UK` as their Max Read Groups, and `UK,CAN` as their Max Write Groups.assigned to him. The user can set the session label to `C:ALPHA:UK,CAN` but cannot change it to either `C:ALPHA` or `C:ALPHA:UK,US,CAN`.

Related Topics

- SA_SESSION.SET_LABEL
  The `SA_SESSION.SET_LABEL` procedure sets the label of the current database session.
- SA_SESSION.SET_ROW_LABEL
  The `SA_SESSION.SET_ROW_LABEL` procedure sets the default row label value for the current database session.

14.7.3 Examples of Session Labels and Inverse Groups

Oracle provides examples of using inverse groups.

- Example: Simple Inverse Groups
  You can create a simple policy that implements inverse groups with a set of special labels.
- Example: Complex Inverse Groups
  You can create a more complex policy that implements inverse groups with a set of special labels.

14.7.3.1 Example: Simple Inverse Groups

You can create a simple policy that implements inverse groups with a set of special labels.

Table 14-6 Labels for Inverse Groups Example 1

Name	Definition
Max Read Label	`SE:ALPHA,BETA:G1,G2`
Max Write Label	`SE:ALPHA:G1,G2,G3`
Default Read Label	`SE:ALPHA,BETA:G1,G2`
Default Write Label	`SE:ALPHA:G1,G2`
Default Row Label	`SE:ALPHA:G1,G2`
From which the following values are derived:	-
Max Read Groups	`G1,G2`

Table 14-6 (Cont.) Labels for Inverse Groups Example 1

Name	Definition
Max Write Groups	`G1,G2,G3`

The following conclusions can be drawn:

- `User01` can update data with label `SE:ALPHA:G1,G2` as well as data with label `SE:ALPHA:G1,G2,G3`. `User1` *cannot*, however, update label `SE:ALPHA:G1`.

  If standard groups were being used, rather than inverse groups, then `User1` could update data with label `SE:ALPHA:G1`.
- Data that `User01` inserts has the label `SE:ALPHA:G1,G2`. (This is the same as with standard groups.)
- If `User01` leaves the default label as is, and sets the row label to `SE:ALPHA:G1,G2,G3`, then `user1` will insert `SE:ALPHA:G1,G2,G3` in new rows of data that is written. (In standard groups, User1 can never set more groups in the row label than in the default label.)

14.7.3.2 Example: Complex Inverse Groups

You can create a more complex policy that implements inverse groups with a set of special labels.

Table 14-7 Labels for Inverse Groups Example 2

Name	Definition
Max Read Label	`C:ALPHA:`
Max Write Label	`C:ALPHA:G1,G2,G3`
Default Read Label	`C:ALPHA:`
Default Write Label	`C:ALPHA:`
Default Row Label	`C:ALPHA:`
From which the following values are derived:	-
Max Read Groups	(an empty set)
Max Write Groups	`G1,G2,G3`

The following conclusions can be drawn:

- `User01` can update any data with level C, compartment `ALPHA`, and any combination of groups `G1`, `G2`, `G3`, or no groups. `User01` inserts the label `C:ALPHA:` in new data that `User01` writes.
- `User02`, who has Max Read Groups of `G1,G2` or `G1,G3`, and so on, will not be able to view the data written by `User01`. This is because `User01`'s Default Row Label contains no groups.

- `User01` can choose to set inverse groups in the row label, as long as the inverse groups in the session label dominates the row label (that is, `User01`'s session label contains the same or fewer groups than contained in the row label).

  This is true because the row label must have at least the groups in the session label, and can at most have the Maximum Write Groups. If the session label is `G1`, then you can set the groups in the row label from `G1` to the Max Write Groups (`G1`,`G2`,`G3`).
- If `User01` sets their session label and row label to `C:ALPHA:G1:G2:G3`, then their data becomes accessible to anyone who has any combination of `G1`,`G2`,`G3` in their Max Read Groups.

14.8 Changes in Behavior of Procedures with Inverse Groups

The `INVERSE_GROUP` option affects algorithms that determine the read and write access of the user to labeled data.

- SA_SYSDBA.CREATE_POLICY with Inverse Groups
  The `SA_SYSDBA.CREATE_POLICY` procedure creates the policy, defines an optional policy-specific column name, and specifies policy options.
- SA_SYSDBA.ALTER_POLICY with Inverse Groups
  The `SA_SYSDBA.ALTER_POLICY` procedure changes a policy's default enforcement options, except for the `INVERSE_GROUP` option.
- SA_USER_ADMIN.ADD_GROUPS with Inverse Groups
  The `SA_USER_ADMIN.ADD_GROUPS` procedure adds groups to a user, indicating whether the groups are authorized for write as well as read.
- SA_USER_ADMIN.ALTER_GROUPS with Inverse Groups
  The `SA_USER_ADMIN.ALTER_GROUPS` procedure changes the write access, default label indicator, and row label indicator for each group.
- SA_USER_ADMIN.SET_GROUPS with Inverse Groups
  The `SA_USER_ADMIN.SET_GROUPS` procedure assigns groups to a user and identifies default values for the user's session label and row label.
- SA_USER_ADMIN.SET_USER_LABELS with Inverse Groups
  The `SA_USER_ADMIN.SET_USER_LABELS` procedure sets the user's levels, compartments, and groups using a set of labels, instead of the individual components.
- SA_USER_ADMIN.SET_DEFAULT_LABEL with Inverse Groups
  The `SA_USER_ADMIN.SET_DEFAULT_LABEL` procedure sets the user's initial session label.
- SA_USER_ADMIN.SET_ROW_LABEL with Inverse Groups
  The `SA_USER_ADMIN.SET_ROW_LABEL` procedure sets the user's initial row label.
- SA_COMPONENTS.CREATE_GROUP with Inverse Groups
  The `SA_COMPONETS.CREATE_GROUP` procedure create a group, including its short name and long name, and optionally a parent group.
- SA_COMPONENTS.ALTER_GROUP_PARENT with Inverse Groups
  The `SA_COMPONENTS.ALTER_GROUP_PARENT` function is disabled for policies with the inverse group option.
- SA_SESSION.SET_LABEL with Inverse Groups
  The `SA_SESION.SET_LABEL` procedure sets the label of the current database session.

- SA_SESSION.SET_ROW_LABEL with Inverse Groups
  The `SET_ROW_LABEL` procedure sets the default row label value for the current database session.
- OLS_LEAST_UBOUND with Inverse Groups
  The `OLS_LEAST_UBOUND` (`OLS_LUBD`) function returns a character string label that is the least upper bound of `label1` and `label2`.
- OLS_GREATEST_LBOUND with Inverse Groups
  The `OLS_GREATEST_LBOUND` function determines the lowest label of the data that can be involved in an operation, given two different labels.

14.8.1 SA_SYSDBA.CREATE_POLICY with Inverse Groups

The `SA_SYSDBA.CREATE_POLICY` procedure creates the policy, defines an optional policy-specific column name, and specifies policy options.

With inverse group support the, user has one more policy enforcement option, `INVERSE_GROUP`. For example:

```
PROCEDURE CREATE_POLICY (
 HR IN VARCHAR2,
 SA_LABEL IN VARCHAR2 DEFAULT NULL,
 INVERSE_GROUP IN VARCHAR2 DEFAULT NULL);
```

Related Topics

- SA_SYSDBA.CREATE_POLICY
  The `SA_SYSDBA.CREATE_POLICY` procedure creates a new Oracle Label Security policy, defines a policy-specific column name, and specifies default policy options.

14.8.2 SA_SYSDBA.ALTER_POLICY with Inverse Groups

The `SA_SYSDBA.ALTER_POLICY` procedure changes a policy's default enforcement options, except for the `INVERSE_GROUP` option.

Once a policy is configured for inverse groups, it cannot be changed. You can also change the column names associated with an OLS policy.

Related Topics

- SA_SYSDBA.ALTER_POLICY
  The `SA_SYSDBA.ALTER_POLICY` procedure sets and modifies column names that are associated with the policy.

14.8.3 SA_USER_ADMIN.ADD_GROUPS with Inverse Groups

The `SA_USER_ADMIN.ADD_GROUPS` procedure adds groups to a user, indicating whether the groups are authorized for write as well as read.

The type of access authorized depends on the `access_mode` parameter.

Table 14-8 Access Authorized by Values of access_mode Parameter

Access_Mode Parameter	Meaning
`READ_WRITE`	Indicates that write is authorized. (That is, the group is contained in both Max Read Groups and Max Write Groups.)
`WRITE_ONLY`	Indicates that the group is contained in Max Write Groups and not in Max Read Groups
`access_mode`	If `access_mode` is set to `READ_WRITE`, then the group is added to both Max Read Groups and Max Write Groups. If `access_mode` is set to `SA_UTL.WRITE_ONLY`, then the group is added only to the Max Write Groups. If `access_mode` is `NULL`, then it is set to `SA_UTL.READ_WRITE`.
`in_def`	Specifies whether these groups should be in the default groups (`Y`/`N`). If `in_def` is `NULL`, then it will be set to `Y` or `N` as follows: If access mode is `READ_WRITE`, `in_def` is set to `Y`. If access mode is `WRITE_ONLY`, `in_def` is set to `N`.
`in_row`	Specifies whether these groups should be in the row label (`Y`/`N`), using the identical criteria as for `in_def`. However, if `in_def` is `Y`, then `in_row` must also be `Y`.

Note that if `in_def` is `Y` in a row, then `in_row` must also be set to `Y`, but not the other way round.

The same is the case with the `in_row` field.

See Also:

- Syntax for SA_USER_ADMIN.ADD_GROUPS
- Inverse Groups and Computed Max Read Groups and Max Write Groups

14.8.4 SA_USER_ADMIN.ALTER_GROUPS with Inverse Groups

The `SA_USER_ADMIN.ALTER_GROUPS` procedure changes the write access, default label indicator, and row label indicator for each group.

The behavior of inverse groups is the same as described in the case of `ADD_GROUPS`.

See Also:

Syntax for SA_USER_ADMIN.ALTER_GROUPS

14.8.5 SA_USER_ADMIN.SET_GROUPS with Inverse Groups

The `SA_USER_ADMIN.SET_GROUPS` procedure assigns groups to a user and identifies default values for the user's session label and row label.

Inverse groups are handled differently than standard groups, as follows:

Table 14-9 Assigning Groups to a User

Group Set Name	Meaning
`read_groups`	A comma-delimited list of groups that would be Max Read Groups
`write_groups`	A comma-delimited list of groups that would be Max Write Groups. It must be a superset of `read_groups`. If `write_groups` is `NULL`, then they are set to `read_groups`.
def_groups	Specifies the default groups. It should at least have `read_groups`, and `write_groups` should be a superset of `def_groups`. If `def_groups` is `NULL`, then they are set to the `read_groups`.
`row_groups`	Specifies the row groups. It should at least have the `def_groups` and should be a subset of max write groups. If `row_groups` is `NULL`, then they are set to the `def_groups`, because for inverse groups, all `def_groups` are also in `write_groups`.

See Also:

Syntax for SA_USER_ADMIN.SET_GROUPS

14.8.6 SA_USER_ADMIN.SET_USER_LABELS with Inverse Groups

The `SA_USER_ADMIN.SET_USER_LABELS` procedure sets the user's levels, compartments, and groups using a set of labels, instead of the individual components.

Inverse groups are handled differently than standard groups, as follows:

Table 14-10 Inverse Group Label Definitions

Name	Definition
`max_read_label`	Specifies the label string to be used to initialize the user's maximum authorized read label. Composed of the user's maximum level, compartments authorized for read access, and if inverse groups, minimum set of groups that can be set in any label.(Max Read Groups)

Table 14-10 (Cont.) Inverse Group Label Definitions

Name	Definition
`max_write_label`	Specifies the label string to be used to initialize the user's maximum authorized write label. Composed of the user's maximum level, compartments authorized for write access, and if inverse groups, the maximum authorized groups that can be set in any label (Max Write Groups). All the inverse groups in this have write authorization also. It should be a superset of groups in `max_read_label`. If `max_write_label` is not specified, then it is set to `max_read_label`.
`def_label`	Specifies the label string to be used to initialize the user's session label, including level, compartments, and groups (a subset of `max_read_label`). If `default_label` is not specified, then it is set to `max_read_label`. For inverse groups, component it should at least have the groups in `max_read_label`, and groups in `max_write_label` should be a superset of the groups in the `def_label`.
`row_label`	Specifies the label string to be used to initialize the program's row label. Includes levels, compartments, and groups: subsets of `max_write_label` and `def_label`. If `row_label` is not specified, then it is set to `def_label`, with only the compartments and groups authorized for write access. The inverse groups component is set to the same as that in `def_label` if the `row_label` is not specified. The inverse groups in row label should at least be those in default label and should be a subset of Max Write Groups.

See Also:

Syntax for SA_USER_ADMIN.SET_USER_LABELS

14.8.7 SA_USER_ADMIN.SET_DEFAULT_LABEL with Inverse Groups

The `SA_USER_ADMIN.SET_DEFAULT_LABEL` procedure sets the user's initial session label.

All the rules mentioned for setting inverse groups component of session label mentioned in Session Labels and Inverse Groups are applicable here.

See Also:

Syntax for SA_USER_ADMIN.SET_DEFAULT_LABEL

14.8.8 SA_USER_ADMIN.SET_ROW_LABEL with Inverse Groups

The `SA_USER_ADMIN.SET_ROW_LABEL` procedure sets the user's initial row label.

When specifying the `row_label`, the inverse groups component must contain at least all the inverse groups in `def_label` and should be a subset of Max Write Groups.

See Also:

- Syntax for SA_USER_ADMIN.SET_ROW_LABEL
- Initial Session and Row Labels for Standard or Inverse Groups

14.8.9 SA_COMPONENTS.CREATE_GROUP with Inverse Groups

The `SA_COMPONETS.CREATE_GROUP` procedure create a group, including its short name and long name, and optionally a parent group.

With inverse groups, the `parent_name` field should always be `NULL`. If the user specifies a value for this field, then an error message is displayed, indicating that the group hierarchy is disabled.

See Also:

Syntax for SA_COMPONENTS.CREATE_GROUP

14.8.10 SA_COMPONENTS.ALTER_GROUP_PARENT with Inverse Groups

The `SA_COMPONENTS.ALTER_GROUP_PARENT` function is disabled for policies with the inverse group option.

An error message is displayed if the user calls this function.

See Also:

Syntax for SA_COMPONENTS.ALTER_GROUP

14.8.11 SA_SESSION.SET_LABEL with Inverse Groups

The `SA_SESION.SET_LABEL` procedure sets the label of the current database session.

For the current user, this procedure follows the same rules for setting the session label as does the `SA_USER_ADMIN.SET_USER_LABEL` function.

See Also:

- Syntax for SA_SESSION.SET_LABEL.
- Setting Current Session or Row Labels for Standard or Inverse Groups

14.8.12 SA_SESSION.SET_ROW_LABEL with Inverse Groups

The `SET_ROW_LABEL` procedure sets the default row label value for the current database session.

For the current user, this procedure follows the same rules for setting the row label as does the `sa_user_admin.set_row_label` function.

See Also:

- Syntax for SA_SESSION.SET_ROW_LABEL
- Initial Session and Row Labels for Standard or Inverse Groups

14.8.13 OLS_LEAST_UBOUND with Inverse Groups

The `OLS_LEAST_UBOUND` (`OLS_LUBD`) function returns a character string label that is the least upper bound of `label1` and `label2`.

With *standard* groups, the least upper bound is the highest level, the union of the compartments in the labels, and t*he union of the groups* in the labels.

With *inverse* groups, the least upper bound is the highest level, the union of the compartments in the labels, and *the intersection of the inverse groups* in the labels.

For example, with inverse groups, the least upper bound of `HIGHLY_SENSITIVE:ALPHA:G1,G2` and `SENSITIVE:BETA:G1` is `HIGHLY_SENSITIVE:ALPHA,BETA:G1`.

14.8.14 OLS_GREATEST_LBOUND with Inverse Groups

The `OLS_GREATEST_LBOUND` function determines the lowest label of the data that can be involved in an operation, given two different labels.

This function returns a character string label that is the greatest lower bound of `label1` and `label2`.

With *standard* groups, the greatest lower bound is the lowest level, and the *intersection of the compartments in the labels and the groups* in the labels.

With *inverse* groups, the greatest lower bound is the lowest level, and the *intersection of the compartments in the labels and the union of inverse groups* in the labels.

For example, with inverse groups the greatest lower bound of `HIGHLY_SENSITIVE:ALPHA:G1,G3` and `SENSITIVE::G1` is `SENSITIVE:G1,G3`

Related Topics

- Determination of the Upper and Lower Bounds of Labels
  Oracle Label Security provides functions that determine the least upper bound or the greatest lower bound of two or more labels.

14.9 Dominance Rules for Labels with Inverse Groups

You should understand how dominance rules work for Oracle labels and inverse groups.

Dominance rules for Oracle Label Security with standard groups can be summarized as follows:

A user label dominates a data label if:

- User level is greater than or equal to the data level
- User compartments are a superset of the data compartments
- User groups intersects (have at least one group from) the data groups

Dominance rules for Oracle Label Security with inverse groups can be summarized as follows:

A user label dominates a data label if:

- User level is greater than or equal to the data level
- User compartments are a superset of the data compartments
- Data groups are a superset of user groups

Related Topics

- About Dominant and Dominated Labels
  The relationship between two labels can be described in terms of *dominance.*

15

Auditing Oracle Label Security

All activities in Oracle Label Security can be audited, including Oracle Label Security administrator activities.

- Auditing Oracle Label Security Using Unified Auditing
  Oracle recommends that you migrate all your Oracle Label Security audit policies to unified auditing.
- Auditing Oracle Label Security Using Traditional Auditing
  Traditional auditing is desupported, but traditional audit settings in an upgraded database can be used with some limitations.
- Disabling Traditional Auditing for Oracle Label Security Policies
  Oracle recommends that you disable traditional auditing for Oracle Label Security policies and use unified auditing instead.

15.1 Auditing Oracle Label Security Using Unified Auditing

Oracle recommends that you migrate all your Oracle Label Security audit policies to unified auditing.

To create **new** audit policies in Oracle Label Security, you must use unified auditing. Traditional auditing is no longer supported as of Oracle Database 23ai for new audit settings, though the current existing traditional audit settings are still honored.

The unified audit trail will capture Oracle Label Security unified audit policy records, as well as mandatory Oracle Label Security audits.

Unified auditing enables you to create custom policies that capture more fine-tuned data than you can capture with traditional Oracle Label Security auditing. For example, you can create unified auditing policies that capture Oracle Label Security-specific events such as the creation, modification, and removal of Oracle Label Security policies, as well as activities such as authorization and actions that require Oracle Label Security privileges. The policies can incorporate standard unified audit features, such as conditions, application context values, and the ability to audit top-level statements.

In addition to this functionality, unified auditing provides the following predefined audit policies:

- `ORA_DV_SCHEMA_CHANGES`, which monitors Oracle Label Security `LBACSYS` and Oracle Database Vault `DVSYS` schema objects when Oracle Database Vault is in use.
- `ORA_OLS_SCHEMA_CHANGES`, which also monitors `LBACSYS` schema objects. You can use this audit policy can if Oracle Database Vault is not in use. You do not need to enable this policy if `ORA_DV_SCHEMA_CHANGES` is already enabled.

In a new Oracle Database installation, `ORA_DV_SCHEMA_CHANGES` is enabled by default. In an upgraded Oracle database, it is not enabled by default. Uninstallation of Oracle Database Vault drops `ORA_DV_SCHEMA_CHANGES`. To ensure that `LBACSYS` objects are still audited, `ORA_OLS_SCHEMA_CHANGES` will be enabled during uninstallation of Oracle Database Vault, if `ORA_DV_SCHEMA_CHANGES` was enabled.

When you create a new policy, a label column for that policy is added to the database audit trail. The label column is created regardless of whether auditing is enabled or disabled, and independent of whether database auditing or operating system auditing is used. Whenever a record is written to the audit table, each policy provides a label for that record to indicate the session label. The administrator can create audit views to display these labels. Note that in the audit table, the label does not control access to the row, instead it only records the sensitivity of the row.

When you use unified auditing, the Oracle Label Security traditional audit settings will continue to collect audit records, until you do the following: complete migrating the audit settings to unified auditing, disable traditional auditing, and then archive and purge the older audit records. From then on, you can manage Oracle Label Security audit policies through the unified audit policy PL/SQL statements.

All configuration changes made to Oracle Label Security are mandatorily audited and these audit records are written to the unified audit trail, including actions of unprivileged users who attempt to modify Oracle Label Security policies.

Consider auditing any operations that require Oracle Label Security privileges. Because these privileges perform sensitive operations, and because their abuse could jeopardize security, you should closely monitor their dissemination and use.

To learn how to create unified audit policies for Oracle Label Security, see *Oracle Database Security Guide*. See also *Oracle Database Security Guide* for more information about how the desupport of traditional auditing works.

15.2 Auditing Oracle Label Security Using Traditional Auditing

Traditional auditing is desupported, but traditional audit settings in an upgraded database can be used with some limitations.

Though traditional auditing is desupported starting in Oracle Database release 23ai, any current existing Oracle Label Security traditional audit settings that you have will be maintained when you upgrade to release 23ai and the Oracle Label Security data dictionary views will continue to capture this audit information. You can delete existing traditional audit settings. However, you cannot create new traditional audit settings. Instead, you must create unified audit policies to replace the traditional audit policies that you need to update.

The traditional audit settings are controlled through the `SA_AUDIT_ADMIN` PL/SQL package. The audit indicates if the user's action succeeded (that is, the policy enabled the user to accomplish a task) or if the user's action failed (the policy was violated). Oracle Label Security uses this package to collect audit records and write these audit records to the Oracle Label Security data dictionary views.

When you install a new database and configure it to use Oracle Label Security, then it uses unified auditing only. If you have upgraded from a previous release, then Oracle Label Security uses the auditing that was implemented in that release.

The traditional audit trail lists only the action numbers. To find the corresponding audit action names, you can query the `LABACSYS.OLS$AUDIT_ACTIONS` system table. You must have the `AUDIT_VIEWER`, `AUDIT_ADMIN`, or *`policy`*`_DBA` role to query this table.

See *Oracle Database Security Guide* for more information about how the desupport of traditional auditing works.

15.3 Disabling Traditional Auditing for Oracle Label Security Policies

Oracle recommends that you disable traditional auditing for Oracle Label Security policies and use unified auditing instead.

1. Log in to the database instance as a user who has the `LBAC_DBA` database role (or the *`policy`*`_DBA` role for a specific policy).
2. If necessary, query the `ALL_SA_POLICIES` data dictionary view to find the policies whose traditional auditing you want to disable.
3. Run the `SA_AUDIT_ADMIN.NOAUDIT` procedure to disable auditing for the policy.

   For example:

   ```
   BEGIN
    SA_AUDIT_ADMIN.NOAUDIT(
     policy_name      => 'hr_ols_pol',
     users            => 'jjones',
     audit_option     => NULL);
   END;
   /
   ```

   In this specification,

   - `policy_name`: The policy name is required. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
   - `users`: This setting is a comma-delimited list of users who were audited. If you omit this setting or specify `NULL` for the `users` parameter, then auditing is disabled for all users. To find users who have privileges to modify Oracle Label Security policies, query the `USER_NAME` column of the `ALL_SA_AUDIT_OPTIONS` view.
   - `audit_option`: `NULL` is specified to disable all audit options that have been applied to the `hr_ols_pol` policy. If you want to disable only specific options, then the choices are as follows:
     – `APPLY`: Disables auditing of the application of specified Oracle Label Security policies to tables and schemas
     – `REMOVE`: Disables auditing of the removal of specified Oracle Label Security policies from tables and schemas
     – `SET`: Disables auditing of the setting of user authorizations, and user and program privileges
     – `PRIVILEGES`: Disables auditing of the use of all policy-specific privileges

Related Topics

- SA_AUDIT_ADMIN.NOAUDIT
  The `SA_AUDIT_ADMIN.NOAUDIT` procedure disables Oracle Label Security policy-specific auditing.

- ALL_SA_POLICIES
  The ALL_SA_POLICIES data dictionary view shows for the current user information about Oracle Label Security policies, based on the SA_SYSDBA.CREATE_POLICY procedure.

Part V
Appendixes

Part IV contains reference material for using Oracle Label Security.

A

Disabling, Enabling, Uninstalling, and Reinstalling Oracle Label Security

You can disable, enable, uninstall, and reinstall Oracle Label Security from the command line.

- Disabling and Enabling Oracle Label Security
  You can disable and enable Oracle Label Security as necessary.
- Uninstalling and Reinstalling Oracle Label Security
  You can uninstall and reinstall Oracle Label Security as necessary.

A.1 Disabling and Enabling Oracle Label Security

You can disable and enable Oracle Label Security as necessary.

- When You Must Disable Oracle Label Security
  You may need to disable Oracle Label Security to perform upgrade tasks or correct erroneous configurations.
- Disabling Oracle Label Security
  If Oracle Database Vault has been enabled, then do not disable Oracle Label Security.
- Enabling Oracle Label Security
  You can enable Oracle Label Security in SQL*Plus.

A.1.1 When You Must Disable Oracle Label Security

You may need to disable Oracle Label Security to perform upgrade tasks or correct erroneous configurations.

Another reason for disabling Oracle Label Security is if you want to test an application without enforcing Oracle Label Security. You can reenable Oracle Label Security after you complete the tasks.

You must also disable Oracle Label Security before you uninstall it.

Related Topics

- Checking if Oracle Label Security Has Been Registered and Enabled
  You can query the `DBA_OLS_STATUS` data dictionary view to find if Oracle Label Security has already been registered and enabled.

A.1.2 Disabling Oracle Label Security

If Oracle Database Vault has been enabled, then do not disable Oracle Label Security.

1. Log into the PDB the Oracle Label Security administrator.

   To find the available PDBs, query the `DBA_PDBS` data dictionary view. To check the current PDB, run the `show con_name` command.

2. Query the `DBA_DV_STATUS` data dictionary view to find if Oracle Database Vault has been enabled in this PDB.

   Oracle Database Vault depends on Oracle Label Security. If Oracle Database Vault is installed in the PDB, then do not disable Oracle Label Security.

   ```
   SELECT * FROM DBA_DV_STATUS;
   ```

   Output similar to the following should appear:

   ```
   NAME                 STATUS
   -------------------- -----------
   DV_CONFIGURE_STATUS  FALSE
   DV_ENABLE_STATUS     FALSE
   ```

   If the output is `FALSE`, then you can disable Oracle Label Security.
3. Run the following procedure:

   ```
   EXEC LBACSYS.OLS_ENFORCEMENT.DISABLE_OLS;
   ```
4. Connect to the CDB as a user with the `SYSDBA` administrative privilege.

   ```
   CONNECT / AS SYSDBA
   ```
5. Close and reopen the PDB.

   For example:

   ```
   ALTER PLUGGABLE DATABASE pdb_name CLOSE IMMEDIATE;
   ALTER PLUGGABLE DATABASE pdb_name OPEN;
   ```
6. For Oracle Real Application Clusters (Oracle RAC), repeat these steps for each Oracle RAC node on which you enabled Oracle Label Security.

A.1.3 Enabling Oracle Label Security

You can enable Oracle Label Security in SQL*Plus.

1. Log into the PDB the Oracle Label Security administrator.

   To find the available PDBs, query the `DBA_PDBS` data dictionary view. To check the current PDB, run the `show con_name` command.
2. Run the following procedure:

   ```
   EXEC LBACSYS.OLS_ENFORCEMENT.ENABLE_OLS;
   ```
3. Connect to the CDB as a user with the `SYSDBA` administrative privilege.

   ```
   CONNECT / AS SYSDBA
   ```
4. Close and reopen the PDB.

   For example:

   ```
   ALTER PLUGGABLE DATABASE pdb_name CLOSE IMMEDIATE;
   ALTER PLUGGABLE DATABASE pdb_name OPEN;
   ```
5. For Oracle Real Application Clusters (Oracle RAC), repeat these steps for each Oracle RAC node on which you enabled Oracle Label Security.

A.2 Uninstalling and Reinstalling Oracle Label Security

You can uninstall and reinstall Oracle Label Security as necessary.

- Uninstalling Oracle Label Security
  You can use the `catnools.sql` script to uninstall Oracle Label Security from a pluggable database (PDB).
- Reinstalling Oracle Label Security
  You can reinstall Oracle Label Security in a PDB but not in the CDB root.

A.2.1 Uninstalling Oracle Label Security

You can use the `catnools.sql` script to uninstall Oracle Label Security from a pluggable database (PDB).

You can only uninstall Oracle Label Security from a PDB, not from the CDB root or the application container root.

1. Log into the PDB as the Oracle Label Security administrator.

   To find the available PDBs, query the `DBA_PDBS` data dictionary view. To check the current PDB, run the `show con_name` command.

2. Ensure that Oracle Database Vault is not installed on this PDB by querying the `DBA_USERS` data dictionary view for the user `DVSYS`.

   ```
   SELECT USERNAME FROM DBA_USERS WHERE USERNAME = 'DVSYS';
   ```

   If the output shows `DVSYS`, then Oracle Database Vault is installed and you will need to uninstall Oracle Database Vault before you uninstall Oracle Label Security from the PDB.

3. Disable Oracle Label Security.

   You can find if Oracle Label Security has been enabled by querying the `DBA_OLS_STATUS` data dictionary view.

4. Enter the following command to uninstall Oracle Label Security:

   ```
   @?/rdbms/admin/catnools.sql value [value]
   ```

   In this specification, *`value`* refers to one of the following settings:

   - `LBACSYS` uninstalls the `LBACSYS` schema only. You must be user `SYS` with the `SYSDBA` administrative privilege to use this option. For example:

     ```
     @?/rdbms/admin/catnools.sql LBACSYS
     ```

   - `POLICIES` uninstalls policies but retains the OLS label column. You must have the `LBAC_DBA` role (or be user `LBACSYS`) to use this option.

     ```
     @?/rdbms/admin/catnools.sql POLICIES
     ```

- `POLICIES_WITH_DATA` uninstalls OLS policies without retaining the `OLS` label column. You must have the `LBAC_DBA` role (or be user `LBACSYS`) to use this option.

  ```
  @?/rdbms/admin/catnools.sql POLICIES_WITH_DATA
  ```

5. Connect to the CDB as a user with the `SYSDBA` administrative privilege.

   ```
   CONNECT / AS SYSDBA
   ```

6. Close and reopen the PDB.

   For example:

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

7. For Oracle Real Application Clusters (Oracle RAC), repeat these steps for each Oracle RAC node on which you enabled Oracle Label Security.

Related Topics

- Checking if Oracle Label Security Has Been Registered and Enabled
  You can query the `DBA_OLS_STATUS` data dictionary view to find if Oracle Label Security has already been registered and enabled.
- Disabling Oracle Label Security
  If Oracle Database Vault has been enabled, then do not disable Oracle Label Security.
- *Oracle Database Vault Administrator’s Guide*

A.2.2 Reinstalling Oracle Label Security

You can reinstall Oracle Label Security in a PDB but not in the CDB root.

1. Log into the PDB as the Oracle Label Security administrator.

   To find the available PDBs, query the `DBA_PDBS` data dictionary view. To check the current PDB, run the `show con_name` command.

2. Query the `DBA_OLS_STATUS` data dictionary view to find if Oracle Label Security has already been installed and enabled in this PDB.

   For example:

   ```
   SELECT NAME, STATUS FROM DBA_OLS_STATUS;
   ```

   If the `DBA_OLS_STATUS` data dictionary view is not recognized, then Oracle Label security is not installed and you can reinstall it. If the output is as follows, showing `FALSE`, then you only need to register Oracle Label Security.

   ```
   NAME                 STATUS
   -------------------- -----------
   OLS_CONFIGURE_STATUS  FALSE
   OLS_DIRECTORY_STATUS  FALSE
   OLS_ENABLE_STATUS     FALSE
   ```

3. Exit SQL*Plus.

4. Enter the following command to reinstall Oracle Label Security:

   ```
   @?/rdbms/admin/catols.sql
   ```

5. Connect to the CDB as a user with the `SYSDBA` administrative privilege.

   ```
   CONNECT / AS SYSDBA
   ```

6. Close and reopen the PDB.

   For example:

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

7. Register Oracle Label Security for the PDB.
8. For Oracle Real Application Clusters (Oracle RAC), repeat these steps for each Oracle RAC node on which you enabled Oracle Label Security.

Related Topics

- Registering and Logging in to Oracle Label Security
  Before using Oracle Label Security, you must register (configure) it with the database and then you can log in to Oracle Label Security.

B Advanced Topics in Oracle Label Security

Oracle provides advanced functionality for Oracle Label Security, such as the ability to analyze relationships between labels.

- Analyzing the Relationships Between Labels
  You can analyze the relationships between labels.
- Queries for Audited Oracle Label Security Session Labels
  You can use the unified audit trail to capture information from various audit sources, including Oracle Label Security.
- Oracle Call Interface for Setting Session Labels
  You can use an Oracle Call Interface (OCI) to set session labels.

B.1 Analyzing the Relationships Between Labels

You can analyze the relationships between labels.

- About Dominant and Dominated Labels
  The relationship between two labels can be described in terms of *dominance.*
- Non-Comparable Labels
  It is important to understand how labels can be compared with regard to dominance.
- Using Dominance Functions
  Oracle Label Security provides functions to control dominance.

B.1.1 About Dominant and Dominated Labels

The relationship between two labels can be described in terms of *dominance.*

A user's ability to access an object depends on whether the user's label dominates the label of the object. If a user's label does not dominate the object's label, then the user is not allowed to access the object.

Label dominance is analyzed in terms of all its components: levels, compartments, and groups.

Table B-1 Dominance in the Comparison of Labels

Factor	Criteria for Dominance
Level	For `label1` to dominate `label2`, the level of `label1` must be greater than or equal to that of `label2`.
Compartment	For `label1` to dominate `label2`, the compartments of `label1` must contain *all* the compartments of `label2`.
Group	For `label1` to dominate `label2`, `label1` must contain *at least one* of the groups of `label2`.

One label *dominates* another label if all of its components dominate the components of the other label. For example, the label `HIGHLY_SENSITIVE:FINANCE,OPERATIONS` dominates the label `HIGHLY_SENSITIVE:FINANCE`. Similarly, the label `HIGHLY_SENSITIVE::WR_AP` dominates the label `HIGHLY_SENSITIVE::WR_AP, WR_AR`.

Related Topics

- Dominance Rules for Labels with Inverse Groups
  You should understand how dominance rules work for Oracle labels and inverse groups.

B.1.2 Non-Comparable Labels

It is important to understand how labels can be compared with regard to dominance.

The relationship between two labels cannot always be defined by dominance. Two labels are *non-comparable* if neither label dominates the other.

If any compartments differ between the two labels (as with `HS:A` and `HS:B`), then they are non-comparable. Similarly, the labels `HS:A` and `S:B` are non-comparable.

You can find existing labels by querying the `DBA_SA_LABELS` data dictionary view.

B.1.3 Using Dominance Functions

Oracle Label Security provides functions to control dominance.

- About the Dominance Functions
  You can use dominance functions to specify ranges in queries.
- OLS_DOMINATES Standalone Function
  The `OLS_DOMINATES` (`OLS_DOM`) function returns `1` (`TRUE`) if `label1` dominates `label2`, or `0` (`FALSE`) if it does not.
- OLS_LABEL_DOMINATES Standalone Function
  The standalone `OLS_LABEL_DOMINATES` function checks the dominance of session labels.
- SA_UTL.DOMINATES
  The `SA_UTL.DOMINATES` function returns `TRUE` if `label1` dominates `label2` or if the session label for the given OLS policy dominates `label`.
- OLS_STRICTLY_DOMINATES Standalone Function
  The `OLS_STRICTLY_DOMINATES` (`OLS_S_DOM`) function returns `1` (`TRUE`) if `label1` dominates `label2` and is not equal to it.
- OLS_DOMINATED_BY Standalone Function
  The `OLS_DOMINATED_BY` (`OLS_DOM_BY`) function returns `1` (`TRUE`) if `label1` is dominated by `label2`.
- OLS_STRICTLY_DOMINATED_BY Standalone Function
  The `OLS_STRICTLY_DOMINATED_BY` (`OLS_S_DOM_BY`) function returns `1` (`TRUE`) if `label1` is dominated by `label2` and is not equal to it.

Related Topics

- Ordering Labeled Data Rows
  The `ORDER BY` clause of a `SELECT` statement can be used to order rows by the numeric label tag.

B.1.3.1 About the Dominance Functions

You can use dominance functions to specify ranges in queries.

The following functions enable you to indicate dominance relationships between specified labels.

Table B-2 Functions to Determine Dominance

Function	Description
`OLS_DOMINATES`	The value of `label1` dominates, or is equal to, that of `label2`.
`OLS_LABEL_DOMINATES`	The value of the session label for the corresponding `policy_name` dominates, or is equal to, that of `label`.
`OLS_STRICTLY_DOMINATES`	The value of `label1` dominates that of `label2`, and is not equal to it.
`OLS_DOMINATED_BY`	The value of `label1` is dominated by that of `label2`.
`OLS_STRICTLY_DOMINATED_BY`	The value of `label1` is dominated by that of `label2`, and is not equal to it.

Note that there are two types of dominance function. While the `SA_UTL` dominance functions return `BOOLEAN` values, the standalone dominance functions return integers.

B.1.3.2 OLS_DOMINATES Standalone Function

The `OLS_DOMINATES` (`OLS_DOM`) function returns `1` (`TRUE`) if `label1` dominates `label2`, or `0` (`FALSE`) if it does not.

Syntax

```
OLS_DOMINATES (
  label1          IN NUMBER,
  label2          IN NUMBER)
RETURN INTEGER;
```

Parameters

Table B-3 OLS_DOMINATES Parameters

Parameter	Description
`label1`	The first label to check. To find existing label values, query the `LABEL` and `TAG` columns of the `ALL_SA_LABELS` view.
`label2`	The second label to check

Example

The following example compares existing label tags `1111` and `1112`.

```
SELECT OLS_DOMINATES ('1111', '1112') FROM DUAL;
```

```
OLS_DOMINATES('1111','1112')
----------------------------
                           0
```

B.1.3.3 OLS_LABEL_DOMINATES Standalone Function

The standalone `OLS_LABEL_DOMINATES` function checks the dominance of session labels.

It returns `1` (`TRUE`) if the session label of the specified `policy_name` value dominates or is equal to the label that is specified by the `label` parameter. Otherwise, this function returns `0` (`FALSE`). This function is publicly available.

Note:

This feature is available starting with Oracle Database 12*c* release 1 (12.1.0.2).

In addition to Oracle Label Security policies, you can use this function with both Oracle Data Redaction and Oracle Database Vault policies.

Syntax

```
OLS_LABEL_DOMINATES (
  policy_name     IN VARCHAR2,
  label           IN VARCHAR2)
RETURN INTEGER;
```

Parameters

Table B-4 OLS_LABEL_DOMINATES Parameters

Parameter	Description
`policy_name`	The name of the Oracle Label Security policy whose session label must be checked for dominance. To find existing label values for policies, query the `POLICY_NAME` and `LABEL` columns of the `ALL_SA_LABELS` view.
`label`	The base label against whom the dominance has to be checked

Examples

The following example checks if the session label for the `hr_ols_pol` policy dominates or is equal to the `hs` label.

```
SELECT OLS_LABEL_DOMINATES ('hr_ols_pol', 'hs') FROM DUAL;

OLS_LABEL_DOMINATES('HR_OLS_POL','HS')
--------------------------------------
                                     0
```

This example shows how you can use the `OLS_LABEL_DOMINATES` function in an Oracle Data Redaction policy:

```
BEGIN
 DBMS_REDACT.ADD_POLICY(
   object_schema   => 'oe',
   object_name     => 'customers',
   column_name     => 'customer_id',
   policy_name     => 'redact_cust_user_ids',
   function_type   => DBMS_REDACT.FULL,
   expression      => 'OLS_LABEL_DOMINATES(''hr_ols_pol'', ''hs'') = 0');
END;
/
```

The following example shows how you can use the `OLS_LABEL_DOMINATES` function in an Oracle Database Vault rule definition:

```
EXEC DBMS_MACADM.CREATE_RULE('Check OLS Factor', 'OLS_LABEL_DOMINATES(''hr_ols_pol'',
''hs'') = 1');
```

See Also:

- *Oracle Database Advanced Security Guide* for more information about Data Redaction
- *Oracle Database Vault Administrator’s Guide* for more information about Database Vault realms

B.1.3.4 SA_UTL.DOMINATES

The `SA_UTL.DOMINATES` function returns `TRUE` if `label1` dominates `label2` or if the session label for the given OLS policy dominates `label`.

Syntax

```
SA_UTL.DOMINATES (
  label1          IN NUMBER,
  label2          IN NUMBER)
RETURN BOOLEAN;
```

Parameters

Table B-5 SA_UTL.DOMINATES Parameters

Parameter	Description
`label1`	The first label to check. To find existing label values, query the `LABEL` and `TAG` columns of the `ALL_SA_LABELS` view.
`label2`	The second label to check

Example

The following example compares existing label tags `1111` and `1112`.

```
SET SERVEROUTPUT ON
BEGIN
 IF SA_UTL.DOMINATES(1111, 1112)
```

```
  THEN DBMS_OUTPUT.PUT_LINE('Label 1111 dominates label 1112.');
 ELSE
  DBMS_OUTPUT.PUT_LINE('Label 1112 dominates label 1111.');
 END IF;
END;
/

Label 1112 dominates label 1111.
```

Note:

You cannot use this function with Oracle Data Redaction and Oracle Database Vault conditions. Oracle recommends that you use the OLS_LABEL_DOMINATES function instead.

B.1.3.5 OLS_STRICTLY_DOMINATES Standalone Function

The OLS_STRICTLY_DOMINATES (OLS_S_DOM) function returns 1 (TRUE) if label1 dominates label2 and is not equal to it.

Syntax

```
OLS_STRICTLY_DOMINATES (
  label1          IN NUMBER,
  label2          IN NUMBER)
RETURN INTEGER;
```

Parameters

Table B-6 OLS_STRICTLY_DOMINATES Parameters

Parameter	Description
label1	The first label to check. To find existing label values, query the LABEL and TAG columns of the ALL_SA_LABELS view.
label2	The second label to check

Examples

The following example compares existing label tags 1111 and 1112.

```
SELECT OLS_STRICTLY_DOMINATES ('1111', '1112') FROM DUAL;

OLS_STRICTLY_DOMINATES('1111','1112')
-------------------------------------
                                    0
```

B.1.3.6 OLS_DOMINATED_BY Standalone Function

The `OLS_DOMINATED_BY` (`OLS_DOM_BY`) function returns `1` (`TRUE`) if `label1` is dominated by `label2`.

Syntax

```
OLS_DOMINATED_BY (
  label1          IN NUMBER,
  label2          IN NUMBER)
RETURN INTEGER;
```

Parameters

Table B-7 OLS_STRICTLY_DOMINATES Parameters

Parameter	Description
`label1`	The first label to check. To find existing label values, query the `LABEL` and `TAG` columns of the `ALL_SA_LABELS` view.
`label2`	The second label to check

Example

The following example compares existing label tags `1111` and `1112`.

```
SELECT OLS_DOMINATED_BY ('1111', '1112') FROM DUAL;

OLS_DOMINATED_BY('1111','1112')
-------------------------------
                              1
```

B.1.3.7 OLS_STRICTLY_DOMINATED_BY Standalone Function

The `OLS_STRICTLY_DOMINATED_BY` (`OLS_S_DOM_BY`) function returns `1` (`TRUE`) if `label1` is dominated by `label2` and is not equal to it.

Syntax

```
OLS_STRICTLY_DOMINATED_BY (
  label1          IN NUMBER,
  label2          IN NUMBER)
RETURN INTEGER;
```

Parameters

Table B-8 OLS_DOMINATES Parameters

Parameter	Description
`label1`	The first label to check. To find existing label values, query the `LABEL` and `TAG` columns of the `ALL_SA_LABELS` view.
`label2`	The second label to check

Example

The following example compares existing label tags 1111 and 1112.

```
SELECT OLS_STRICTLY_DOMINATES ('1111', '1112') FROM DUAL;

OLS_STRICTLY_DOMINATES('1111','1112')
-------------------------------------
                                    0
```

B.2 Queries for Audited Oracle Label Security Session Labels

You can use the unified audit trail to capture information from various audit sources, including Oracle Label Security.

- About Queries for Auditing Oracle Label Security Session Labels
  You must configure Oracle Label Security auditing by creating unified audit policies.
- ORA_GET_AUDITED_LABEL Function
  The ORA_GET_AUDITED_LABEL function returns the audited session label for the specified OLS policy and APPLICATION_CONTEXTS column value.

B.2.1 About Queries for Auditing Oracle Label Security Session Labels

You must configure Oracle Label Security auditing by creating unified audit policies.

OLS auditing enables you to audit additional events such as enabling and disabling of OLS policies.

The session labels that the audit trail captures are stored in the APPLICATION_CONTEXTS column of the UNIFIED_AUDIT_TRAIL view. You can use the LBACSYS.ORA_GET_AUDITED_LABEL function to retrieve session labels that are stored in the APPLICATION_CONTEXTS column. This function accepts the UNIFIED_AUDIT_TRAIL.APPLICATION_CONTEXTS column value, and the Oracle Label Security policy name as arguments, and then returns the session label that is stored in the column for the specified policy.

See Also:

Oracle Database Security Guide for detailed information about configuring and using OLS auditing in a unified audit trail

B.2.2 ORA_GET_AUDITED_LABEL Function

The ORA_GET_AUDITED_LABEL function returns the audited session label for the specified OLS policy and APPLICATION_CONTEXTS column value.

The AUDIT_VIEWER role has EXECUTE privilege on the ORA_GET_AUDITED_LABEL function.

Syntax

```
ORA_GET_AUDITED_LABEL (
  appctx_col_value   IN VARCHAR2,
  ols_policy_name    IN VARCHAR2)
RETURN VARCHAR2;
```

Parameters

Table B-9 ORA_GET_AUDITED_LABEL Parameters

Parameter	Description
`appctx_col_value`	Value in the `UNIFIED_AUDIT_TRAIL.APPLICATION_CONTEXTS` column
`policy_name`	The label security policy name

Example

The following example returns the audited session label for the `hr_ols_pol` policy.

```
SELECT ORA_GET_AUDITED_LABEL ('cust_ctx', 'hr_ols_pol') FROM DUAL;

ORA_GET_AUDITED_LABEL('X','HR_OLS_POL')
---------------------------------------
                                     HS
```

B.3 Oracle Call Interface for Setting Session Labels

You can use an Oracle Call Interface (OCI) to set session labels.

- About Using the Oracle Call Interface to Set Session Labels
  When you connect using Oracle Call Interface (OCI), you can use the `SYS_CONTEXT` variables to initialize the session label and the row label.
- Using the Oracle Call Interface to Set Session Labels
  You can use the Oracle Call Interface to set the session labels.
- Example: Using Oracle Call Interface with the SYS_CONTEXT Function
  You can create an OCI call that uses an externalized `SYS_CONTEXT` function with Oracle Label Security.

B.3.1 About Using the Oracle Call Interface to Set Session Labels

When you connect using Oracle Call Interface (OCI), you can use the `SYS_CONTEXT` variables to initialize the session label and the row label.

You can set the variables using the `OCIAttrSet` function to initialize *externally initialized* `SYS_CONTEXT` variables. These are available when Oracle Label Security is enabled.

Each policy has a `SYS_CONTEXT` named `SA$`*`policy_name`*`_X`. You can set these two variables, `INITIAL_LABEL` and `INITIAL_ROW_LABEL`.

When the new values are set to valid labels within the user's authorizations, they will be used instead of the default values stored for the user. This is the same mechanism used for remote connections.

Related Topics

- Using Oracle Label Security with a Distributed Database
  You should understand the special considerations for using Oracle Label Security in a distributed configuration.

B.3.2 Using the Oracle Call Interface to Set Session Labels

You can use the Oracle Call Interface to set the session labels.

1. Call `OCIAttrSet` with `OCI_ATTR_APPCTX_SIZE` to initialize the context array size with the desired number of context attributes:

   ```
   OCIAttrSet(session, OCI_HTYPE_SESSION,
                    (dvoid *)&size, (ub4)0, OCI_ATTR_APPCTX_SIZE,
   error_handle);
   ```

   This defines additional attributes for `OCIAttrSet`.

   Note that the size is `ub4` type.

2. Call `OCIAttrGet` with `OCI_ATTR_APPCTX_LIST` to get a handle on the application context list descriptor for the session:

   ```
   OCIAttrGet(session, OCI_HTYPE_SESSION,
                  (dvoid *)&ctxl_desc, (ub4)0, OCI_ATTR_APPCTX_LIST,
   error_handle);
   ```

   Note that `ctxl_desc` is (`OCIParam *`) type.

3. Call `OCIParamGet` with the application context list descriptor to obtain an individual descriptor for the i-*th* application context:

   ```
   OCIParamGet(ctxl_desc, OCI_DTYPE_PARAM, error_handle,(dvoid **)&ctx_desc,
   i);
   ```

   Note that `ctx_desc` is (`OCIParam *`) type.

4. Call `OCIAttrSet` with each of the three new attributes, `OCI_ATTR_APPCTX_NAME`, `OCI_ATTR_APPCTX_ATTR`, and `OCI_ATTR_APPCTX_VALUE`, to set the proper values in the application context:

   ```
   OCIAttrSet(ctx_desc, OCI_DTYPE_PARAM,
                    (dvoid *)ctx_name, sizeof(ctx_name), OCI_ATTR_APPCTX_NAME,
                    error_handle);

   OCIAttrSet(ctx_desc, OCI_DTYPE_PARAM,
                    (dvoid *)attr_name, sizeof(attr_name), OCI_ATTR_APPCTX_ATTR,
                    error_handle);

   OCIAttrSet(ctx_desc, OCI_DTYPE_PARAM,
                    (dvoid *)value, sizeof(value), OCI_ATTR_APPCTX_VALUE,
                    error_handle);
   ```

   Note that only character type is supported, because application context operations are based on the `VARCHAR2` type.

B.3.3 Example: Using Oracle Call Interface with the SYS_CONTEXT Function

You can create an OCI call that uses an externalized `SYS_CONTEXT` function with Oracle Label Security.

Example B-1 shows how to accomplish this.

Example B-1   Using OCI to Externalize SYS_CONTEXT with OLS

```
#ifdef RCSID
static char *RCSid =
   "$Header: ext_mls.c 09-may-00.10:07:08 jdoe Exp $ ";
#endif /* RCSID */

/* Copyright (c) Oracle Corporation 1999, 2000. All Rights Reserved. */

/*

   NAME
ext_mls.c - externalized SYS_CONTEXT with Label Security

   DESCRIPTION
Run olsdemo.sql script before executing this example.
Usage: <executable obtained with .c file> <user_name> <password> <session-initial-label
Example: avg_sal sa_demo sa_demo L3:M,E:D10

   PUBLIC FUNCTION(S)
<list of external functions declared/defined - with one-line descriptions>

   PRIVATE FUNCTION(S)
<list of static functions defined in .c file - with one-line descriptions>

   RETURNS
The average salary in the EMP table of the SA_DEMO schema querying as the specified
user with the specified session label.

   NOTES
<other useful comments, qualifications, and so on>

   MODIFIED   (MM/DD/YY)
jlev      09/18/03 - cleanup
jdoe      05/09/00 - cleanup
   jdoe      10/13/99 - standalone OCI program to test MLS SYS_CONTEXT
   jdoe      10/13/99 - Creation

*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <oci.h>

static OCIEnv *envhp;
static OCIError *errhp;

int main(/*_ int argc, char *argv[] _*/);

/* get and print error */
static void checkerr(/*_OCIError *errhp, sword status _*/);
```

```
/* print error */
static void printerr(char *call);
static sword status;

/* return the average of employees' salary */
static CONST text *const selectstmt = (text *)
     "select avg(sal) from sa_demo.emp";

int main(argc, argv)
int argc;
char *argv[];
{
  OCISession *authp = (OCISession *) 0;
  OCIServer *srvhp;
  OCISvcCtx *svchp;
  OCIDefine *defnp = (OCIDefine *) 0;
  dvoid *parmdp;
  ub4 ctxsize;
  OCIParam *ctxldesc;
  OCIParam *ctxedesc;
  OCIStmt *stmtp = (OCIStmt *) 0;
  ub4 avg_sal = 0;
  sword status;

  if (OCIInitialize((ub4) OCI_DEFAULT, (dvoid *) 0,
                    (dvoid * (*)(dvoid *, size_t)) 0,
                    (dvoid * (*)(dvoid *, dvoid *, size_t)) 0,
                    (void (*)(dvoid *, dvoid *)) 0))
    printerr("OCIInitialize");

  if (OCIEnvInit((OCIEnv **) &envhp, OCI_DEFAULT, (size_t) 0, (dvoid **) 0))
    printerr("OCIEnvInit");

  if (OCIHandleAlloc((dvoid *) envhp, (dvoid **) &errhp, OCI_HTYPE_ERROR,
                     (size_t) 0, (dvoid **) 0))
    printerr("OCIHandleAlloc:OCI_HTYPE_ERROR");

  if (OCIHandleAlloc((dvoid *) envhp, (dvoid **) &srvhp, OCI_HTYPE_SERVER,
                     (size_t) 0, (dvoid **) 0))
    printerr("OCIHandleAlloc:OCI_HTYPE_SERVER");

  if (OCIHandleAlloc((dvoid *) envhp, (dvoid **) &svchp, OCI_HTYPE_SVCCTX,
                     (size_t) 0, (dvoid **) 0))
    printerr("OCIHandleAlloc:OCI_HTYPE_SVCCTX");

  if (OCIServerAttach(srvhp, errhp, (text *) "", strlen(""), 0))
    printerr("OCIServerAttach");

  /* set attribute server context in the service context */
  if (OCIAttrSet((dvoid *) svchp, OCI_HTYPE_SVCCTX, (dvoid *) srvhp,
                 (ub4) 0, OCI_ATTR_SERVER, (OCIError *) errhp))
    printerr("OCIAttrSet:OCI_HTYPE_SVCCTX");

  if (OCIHandleAlloc((dvoid *) envhp, (dvoid **) &authp,
                     (ub4) OCI_HTYPE_SESSION, (size_t) 0, (dvoid **) 0))
    printerr("OCIHandleAlloc:OCI_HTYPE_SESSION");

  /* set application context to 1 */
  ctxsize = 1;

  /* set up app ctx buffer */
```

```
if (OCIAttrSet((dvoid *) authp, (ub4) OCI_HTYPE_SESSION, (dvoid *) &ctxsize,
               (ub4) 0, (ub4) OCI_ATTR_APPCTX_SIZE, errhp))
  printerr("OCIAttrSet:OCI_ATTR_APPCTX_SIZE");

/* retrieve the list descriptor */
if (OCIAttrGet((dvoid *) authp, (ub4) OCI_HTYPE_SESSION,
               (dvoid *) &ctxldesc, 0, OCI_ATTR_APPCTX_LIST, errhp))
  printerr("OCIAttrGet:OCI_ATTR_APPCTX_LIST");

if (status = OCIParamGet(ctxldesc, OCI_DTYPE_PARAM, errhp,
                          (dvoid **) &ctxedesc, 1))
  {
    if (status == OCI_NO_DATA)
      {
        printf("No Data found!\n");
        exit(1);
      }
  }

/* set context namespace to SA$<pol_name>_X */
if (OCIAttrSet((dvoid *) ctxedesc, (ub4) OCI_DTYPE_PARAM,
               (dvoid *) "SA$HUMAN_RESOURCES_X",
               (ub4) strlen((char *) "SA$HUMAN_RESOURCES_X"),
               (ub4) OCI_ATTR_APPCTX_NAME, errhp))
  printerr("OCIAttrSet:OCI_ATTR_APPCTX_NAME:SA$HUMAN_RESOURCES_X");

/* set context attribute to INITIAL_LABEL */
if (OCIAttrSet((dvoid *) ctxedesc, (ub4) OCI_DTYPE_PARAM,
               (dvoid *) "INITIAL_LABEL",
               (ub4) strlen((char *) "INITIAL_LABEL"),
               (ub4) OCI_ATTR_APPCTX_ATTR, errhp))
  printerr("OCIAttrSet:OCI_DTYPE_PARAM:INITIAL_LABEL");

/* set context value to argv[3] - initial label */
if (OCIAttrSet((dvoid *) ctxedesc, (ub4) OCI_DTYPE_PARAM,
               (dvoid *) argv[3],
               (ub4) strlen((char *) argv[3]),
               (ub4) OCI_ATTR_APPCTX_VALUE, errhp))
  printerr("OCIAttrSet:argv[3]");

/* username first command line argument */
if (OCIAttrSet((dvoid *) authp, (ub4) OCI_HTYPE_SESSION, (dvoid *) argv[1],
               (ub4) strlen((char *) argv[1]), (ub4) OCI_ATTR_USERNAME,
               errhp))
  printerr("OCIAttrSet:username");

/* password second command line argument */
if (OCIAttrSet((dvoid *) authp, (ub4) OCI_HTYPE_SESSION, (dvoid *) argv[2],
               (ub4) strlen((char *) argv[2]), (ub4) OCI_ATTR_PASSWORD,
               errhp))
  printerr("OCIAttrSet:password");

if (OCISessionBegin(svchp, errhp, authp, OCI_CRED_RDBMS, (ub4) OCI_DEFAULT))
  printerr("OCISessionBegin");

if (OCIAttrSet((dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX, (dvoid *) authp,
               (ub4) 0, (ub4) OCI_ATTR_SESSION, errhp))
  printerr("OCIAttrSet:OCI_ATTR_SESSION");

if (OCIHandleAlloc((dvoid *) envhp, (dvoid **) &stmtp, OCI_HTYPE_STMT,
                   0, 0))
    printerr("OCIHandleAlloc:OCI_HTYPE_STMT");

  if (OCIStmtPrepare(stmtp, errhp, (CONST OraText *) selectstmt,
                     (ub4) strlen((const char *) selectstmt),
                     (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT))
    printerr("OCIStmtPrepare");

  if (OCIDefineByPos(stmtp, &defnp, errhp, (ub4) 1, (dvoid *) &avg_sal,
                     (sb4) sizeof(avg_sal), SQLT_INT, 0, 0, 0, OCI_DEFAULT))
    printerr("OCIDefineByPos");

  if (status = OCIStmtExecute(svchp, stmtp, errhp, 1, 0, NULL, NULL,
                              OCI_DEFAULT))
    {
      if (status == OCI_NO_DATA)
        {
          printf("No Data found!\n");
          exit(1);
        }
    }

  if (OCISessionEnd(svchp, errhp, authp, OCI_DEFAULT))
    printerr("OCISessionEnd");

  printf("average salary is: %d\n", avg_sal);
}

void checkerr(errhp, status)
     OCIError *errhp;
     sword status;
{
  text errbuf[512];
  sb4 errcode = 0;

  switch (status)
    {
    case OCI_ERROR:
      (void) OCIErrorGet((dvoid *) errhp, 1, NULL, &errcode, errbuf,
                         (ub4) sizeof(errbuf), OCI_HTYPE_ERROR);
      printf("Error - %.*s\n", 512, errbuf);
      break;
    default:
      break;
    }
}

void printerr(call)
     char *call;
{
  printf("Error: %s\n", call);
}
/* end of file ext_mls.c */
```

C

Oracle Label Security in an Oracle RAC Environment

You can use Oracle Label Security in an Oracle Real Application Clusters (Oracle RAC) environment.

- Oracle Label Security Policy Functions in an Oracle RAC Environment
  Policy changes made on one instance are available to other instances in the Oracle Real Application Clusters (Oracle RAC) environment immediately.
- Transparent Application Failover in Oracle Label Security
  Session information is preserved on Transparent Application Failover.

C.1 Oracle Label Security Policy Functions in an Oracle RAC Environment

Policy changes made on one instance are available to other instances in the Oracle Real Application Clusters (Oracle RAC) environment immediately.

It is not necessary to restart the other instances to pick up the changes.

Important changes made on one database instance are automatically propagated to the other instances. One example would be creating a new policy. Another would be altering the policy options.

Propagating such changes ensures two valuable protections:

- That all users of the table are subject to the same policy
- That if any instance fails, continuation of its work by other instances will use the same policies and parameters that were in force immediately prior to that failure. So, if a policy had been enabled or disabled, it would be seen as such in all instances.

If an administrator changes policy information in one instance by using the policy functions listed in Table C-1, Oracle Label Security stores the relevant information about whatever that function call changed. The new information is immediately available to the other active instances in the Oracle RAC, enabling uniformity among users of the affected policies.

Table C-1 Policy Functions Preserving Status in an Oracle RAC Environment

Policy Functions	Description
`SA_SYSDBA.CREATE_POLICY`	Creates a new policy
`SA_SYSDBA.DROP_POLICY`	Drops an existing policy
`SA_SYSDBA.ENABLE_POLICY`	Enables an existing policy
`SA_SYSDBA.DISABLE_POLICY`	Disables an existing policy
`SA_SYSDBA.ALTER_POLICY`	Alters an existing policy

C.2 Transparent Application Failover in Oracle Label Security

Session information is preserved on Transparent Application Failover.

Any changes to the session's information by way of session functions listed in Table C-2 are preserved on Transparent Application Failover.

For example, suppose a user `SCOTT` is logged on with default label `Top Secret`. If the user calls `sa_session.set_label()` to change their session label to `Secret`, and a failover to another instance occurs, they will see no change but their session label remains `Secret`.

Preserving current user session information means that the access permissions and restrictions on what data that user can see or affect remain as they were. Despite the failover, the user can see and affect only the tables and rows accessible before the failover. If preservation were not the case, failing over to another instance could cause or enable the user to see a different set of data.

Whenever one of the session functions listed in Table C-2 is used, Oracle Label Security stores the relevant information about whatever was changed by that function call.

Table C-2 Session Functions Preserving Status in an Oracle RAC Environment

Session Functions	Description
`SA_SESSION.SET_LABEL`	Lets the user set a new level and new compartments and groups to which they have read access
`SA_SESSION.SET_ROW_LABEL`	Lets the user set the default row label that will be applied to new rows
`SA_SESSION.SAVE_DEFAULT_LABELS`	Lets the user store the current session label and row label as the default for future sessions
`SA_SESSION.RESTORE_DEFAULT_LABELS`	Lets the user reset the current session label and row label to the stored default settings
`SA_SESSION.SET_ACCESS_PROFILE`	Sets the Oracle Label Security authorizations and privileges of the database session to those of the specified user

D

Oracle Label Security PL/SQL Packages

Oracle Label Security provides a set of PL/SQL packages.

- SA_AUDIT_ADMIN Oracle Label Security Auditing PL/SQL Package
  For a non-unified auditing environment, the `SA_AUDIT_ADMIN` PL/SQL package configures auditing that is specific to Oracle Label Security.
- SA_COMPONENTS Label Components PL/SQL Package
  The `SA_COMPONENTS` PL/SQL package manages the component definitions of an Oracle Label Security label.
- SA_LABEL_ADMIN Label Management PL/SQL Package
  The `SA_LABEL_ADMIN` PL/SQL package provides an administrative interface to manage the labels used by a policy.
- SA_POLICY_ADMIN Policy Administration PL/SQL Package
  The `SA_POLICY_ADMIN` PL/SQL package manages Oracle Label Security policies as a whole.
- SA_SESSION Session Management PL/SQL Package
  The `SA_SESSION` PL/SQL package manages session behavior for user authorizations.
- SA_SYSDBA Policy Management PL/SQL Package
  The `SA_SYSDBA` PL/SQL package manages Oracle Label Security policies.
- SA_USER_ADMIN PL/SQL Package
  The `SA_USER_ADMIN` PL/SQL package manages user labels by label component.
- SA_UTL PL/SQL Utility Functions and Procedures
  The `SA_UTL` PL/SQL package contains utility functions and procedures that are used in PL/SQL programs.

See Also:

Using Dominance Functions for additional standalone Oracle Label Security functions

D.1 SA_AUDIT_ADMIN Oracle Label Security Auditing PL/SQL Package

For a non-unified auditing environment, the `SA_AUDIT_ADMIN` PL/SQL package configures auditing that is specific to Oracle Label Security.

- About the SA_AUDIT_ADMIN PL/SQL Package
  The `SA_AUDIT_ADMIN` PL/SQL package configures traditional auditing for labels and policies, as well as creating an auditing-related view.

- SA_AUDIT_ADMIN.AUDIT_LABEL_ENABLED
  The `SA_AUDIT_ADMIN.AUDIT_LABEL_ENABLED` function shows whether labels are being recorded in audit records for the policy.
- SA_AUDIT_ADMIN.CREATE_VIEW
  The `SA_AUDIT_ADMIN.CREATE_VIEW` procedure creates an audit trail view named `DBA_`*`policyname`*`_AUDIT_TRAIL`.
- SA_AUDIT_ADMIN.DROP_VIEW
  The `SA_AUDIT_ADMIN.DROP_VIEW` procedure drops the audit trail view for the specified policy.
- SA_AUDIT_ADMIN.NOAUDIT
  The `SA_AUDIT_ADMIN.NOAUDIT` procedure disables Oracle Label Security policy-specific auditing.
- SA_AUDIT_ADMIN.NOAUDIT_LABEL
  The `SA_AUDIT_ADMIN.NOAUDIT_LABEL` procedure disables the auditing of policy labels.

D.1.1 About the SA_AUDIT_ADMIN PL/SQL Package

The `SA_AUDIT_ADMIN` PL/SQL package configures traditional auditing for labels and policies, as well as creating an auditing-related view.

Starting in Oracle Database release 23ai, traditional auditing is desupported. However, the `SA_AUDIT_ADMIN` package is retained so that you can disable any Oracle Label Security policies that still have traditional auditing enabled. The following procedures in this package have been disabled:

- `SA_AUDIT_ADMIN.AUDIT`
- `SA_AUDIT_ADMIN.AUDIT_LABEL`

Related Topics

- Auditing Oracle Label Security Using Unified Auditing
  Oracle recommends that you migrate all your Oracle Label Security audit policies to unified auditing.
- Duties of Oracle Label Security Administrators
  Oracle Label Security administrators have a set of package- and role-based privileges.

D.1.2 SA_AUDIT_ADMIN.AUDIT_LABEL_ENABLED

The `SA_AUDIT_ADMIN.AUDIT_LABEL_ENABLED` function shows whether labels are being recorded in audit records for the policy.

Syntax

```
SA_AUDIT_ADMIN.AUDIT_LABEL_ENABLED (
  policy_name IN VARCHAR2)
RETURN BOOLEAN;
```

Parameters

Table D-1 SA_AUDIT_ADMIN.AUDIT_LABEL_ENABLED Parameter

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.

Example

The following example writes output indicating whether the Oracle Label Security labels are being audited for the `hr_ols_pol` policy.

```
SET SERVEROUTPUT ON
BEGIN
 IF SA_AUDIT_ADMIN.AUDIT_LABEL_ENABLED('hr_ols_pol')
  THEN DBMS_OUTPUT.PUT_LINE('OLS hr_ols_pol labels are being audited.');
 ELSE
  DBMS_OUTPUT.PUT_LINE('OLS hr_ols_pol labels not being audited.');
 END IF;
END;
/
```

D.1.3 SA_AUDIT_ADMIN.CREATE_VIEW

The `SA_AUDIT_ADMIN.CREATE_VIEW` procedure creates an audit trail view named `DBA_`*`policyname`*`_AUDIT_TRAIL`.

This view contains the specified policy's label column as well as all the entries in the audit trail written on behalf of this policy. If the view name exceeds the database limit of 30 characters, then the user can optionally specify a shorter view name.

Oracle Label Security grants the `READ` privilege on the `DBA_`*`policyname`*`_AUDIT_TRAIL` view to the Oracle Label Security policy database administrator.

See Also:

Oracle Label Security User-Created Auditing View to find the columns that are contained in the `DBA_`*`policyname`*`_AUDIT_TRAIL` view

Syntax

```
SA_AUDIT_ADMIN.CREATE_VIEW (
     policy_name      IN VARCHAR2,
     view_name        IN VARCHAR2     DEFAULT NULL);
```

Parameters

Table D-2 SA_AUDIT_ADMIN.CREATE_VIEW Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`view_name`	Optional. Specifies the name of the view name. If you omit this setting, then the name defaults to `DBA_`*`policyname`*`_AUDIT_TRAIL`.

Examples

The following example creates a view called `hr_ols_pol_view` for the `hr_ols_pol` policy.

```
BEGIN
 SA_AUDIT_ADMIN.CREATE_VIEW(
  policy_name      => 'hr_ols_pol',
  view_name        => 'hr_ols_pol_view');
END;
/
```

D.1.4 SA_AUDIT_ADMIN.DROP_VIEW

The `SA_AUDIT_ADMIN.DROP_VIEW` procedure drops the audit trail view for the specified policy.

Syntax

```
SA_AUDIT_ADMIN.DROP_VIEW (
     policy_name      IN VARCHAR2,
     view_name        IN VARCHAR2     DEFAULT NULL);
```

Parameters

Table D-3 SA_AUDIT_ADMIN.DROP_VIEW Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`view_name`	Specifies an existing view's name. You can find this view by first querying the `ALL_SA_POLICIES` data dictionary view to find the name of the policy on which the view was based, and then querying `ALL_VIEWS` data dictionary view to find any views that have the name of the policy.

Example

The following example drops the view called `hr_ols_pol_view` from the `hr_ols_pol` policy.

```
BEGIN
 SA_AUDIT_ADMIN.DROP_VIEW(
  policy_name      => 'hr_ols_pol',
  view_name        => 'hr_ols_pol_view');
END;
/
```

D.1.5 SA_AUDIT_ADMIN.NOAUDIT

The `SA_AUDIT_ADMIN.NOAUDIT` procedure disables Oracle Label Security policy-specific auditing.

Syntax

```
SA_AUDIT_ADMIN.NOAUDIT (
     policy_name     IN VARCHAR2,
     users           IN VARCHAR2 DEFAULT NULL,
     audit_option    IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table D-4   SA_AUDIT_ADMIN.NO_AUDIT Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`users`	Optional. A comma-delimited list of users who were audited. If not specified, then auditing is disabled for all users. To find users who have privileges to modify Oracle Label Security policies, query the `USER_NAME` column of the `ALL_SA_AUDIT_OPTIONS` view.
`audit_option`	Optional. A comma-delimited list of options to be disabled. Options are as follows: • `APPLY`: Disables auditing of the application of specified Oracle Label Security policies to tables and schemas • `REMOVE`: Disables auditing of the removal of specified Oracle Label Security policies from tables and schemas • `SET`: Disables auditing of the setting of user authorizations, and user and program privileges • `PRIVILEGES`: Disables auditing of the use of all policy-specific privileges If not specified, then all default options are disabled. Privileges must be disabled explicitly.

Examples

The following example disables auditing for failed `APPLY` and `REMOVE` attempts by the users `psmith` and `rlayton`.

```
BEGIN
 SA_AUDIT_ADMIN.NOAUDIT(
  policy_name      => 'hr_ols_pol',
  users            => 'jjones',
  audit_option     => 'apply, remove');
```

```
END;
/
```

You can disable auditing for all enabled options, or only for a subset of enabled options. All auditing for the specified options is disabled for all specified users (or all users, if the *users* parameter is `NULL`). For example, the following statement disables auditing of the `APPLY` and `REMOVE` operations for users John, Mary, and Scott:

```
EXEC SA_AUDIT_ADMIN.NOAUDIT ('HR', 'JOHN, MARY, SCOTT', 'APPLY, REMOVE');
```

Consider also a case in which one `AUDIT` call enables auditing for a specific user, and a second call (with no user specified) enables auditing for all users. For example:

```
EXEC SA_AUDIT_ADMIN.AUDIT ('HR', 'SCOTT');
EXEC SA_AUDIT_ADMIN.AUDIT ('HR');
```

In this case, a subsequent call to `NOAUDIT` with no users specified (such as the following statement) does not reverse the auditing that was set for `SCOTT` explicitly in the first call. So, auditing continues to be performed on `SCOTT`.

```
EXEC SA_AUDIT_ADMIN.NOAUDIT ('HR');
```

In this way, even if `SA_AUDIT_ADMIN.NOAUDIT` is set for all users, Oracle Label Security still audits any users for whom auditing was explicitly set.

Auditing of privileged operations must be specified explicitly. If you run `SA_AUDIT_ADMIN.NOAUDIT` with no options, the Oracle Label Security will nonetheless continue to audit privileged operations. For example, if auditing is enabled and you enter

```
EXEC SA_AUDIT_ADMIN.NOAUDIT ('HR');
```

then auditing will continue to be performed on the privileged operations (such as `WRITEDOWN`).

`SA_AUDIT_ADMIN.NOAUDIT` parameters and options that you set apply only to subsequent sessions, not to current sessions.

If you try to enable an audit option that has already been set, or if you try to disable an audit option that has not been set, then Oracle Label Security processes the statement without indicating an error. An attempt to specify an invalid option results in an error message. You can find the status of audit options by querying the `ALL_SA_AUDIT_OPTIONS` data dictionary view.

D.1.6 SA_AUDIT_ADMIN.NOAUDIT_LABEL

The `SA_AUDIT_ADMIN.NOAUDIT_LABEL` procedure disables the auditing of policy labels.

Syntax

```
SA_AUDIT_ADMIN.NOAUDIT_LABEL (
   policy_name     IN VARCHAR2);
```

Parameters

Table D-5 SA_AUDIT_ADMIN.NO_AUDIT_LABEL Parameter

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.

Example

The following example disables auditing for the `hr_ols_pol` policy.

```
BEGIN
 SA_AUDIT_ADMIN.NOAUDIT_LABEL(
  policy_name       => 'hr_ols_pol');
END;
/
```

D.2 SA_COMPONENTS Label Components PL/SQL Package

The `SA_COMPONENTS` PL/SQL package manages the component definitions of an Oracle Label Security label.

- About the SA_COMPONENTS PL/SQL Package
  The `SA_COMPONENTS` PL/SQL package configures compartments, groups, parent groups, and levels.
- SA_COMPONENTS.ALTER_COMPARTMENT
  The `SA_COMPONENTS.ALTER_COMPARTMENT` procedure changes the short name and long name associated with a compartment.
- SA_COMPONENTS.ALTER_GROUP
  The `SA_COMPONENTS.ALTER_GROUP` procedure changes the short name and long name associated with a group.
- SA_COMPONENTS.ALTER_GROUP_PARENT
  The `SA_COMPONENTS.ALTER_GROUP_PARENT` procedure changes the parent group associated with a particular group.
- SA_COMPONENTS.ALTER_LEVEL
  The `SA_COMPONENTS.ALTER_LEVEL` procedure changes the short name and long name associated with a level.
- SA_COMPONENTS.CREATE_COMPARTMENT
  The `SA_COMPONENTS.CREATE_COMPARTMENT` procedure creates a compartment and specify its short name and long name.
- SA_COMPONENTS.CREATE_GROUP
  The `SA_COMPONENTS.CREATE_GROUP` procedure creates a group and specify its short name and long name, and optionally a parent group.
- SA_COMPONENTS.CREATE_LEVEL
  The `SA_COMPONENTS.CREATE_LEVEL` procedure creates a level and specify its short name and long name.
- SA_COMPONENTS.DROP_COMPARTMENT
  The `SA_COMPONENTS.DROP_COMPARTMENT` procedure removes a compartment.

- SA_COMPONENTS.DROP_GROUP
  The `SA_COMPONENTS.DROP_GROUP` procedure removes a group.
- SA_COMPONENTS.DROP_LEVEL
  The `SA_COMPONENTS.DROP_LEVEL` procedure removes a level.

D.2.1 About the SA_COMPONENTS PL/SQL Package

The `SA_COMPONENTS` PL/SQL package configures compartments, groups, parent groups, and levels.

To use this package, you must be granted the *`policy`*`_DBA` role (for example, `HR_OLS_POL_DBA` for a role for the `hr_ols_pol` policy) and the `EXECUTE` privilege on the `SA_COMPONENTS` package.

Related Topics

- Understanding Data Labels and User Labels
  You should understand fundamental concepts of data labels and user labels.

D.2.2 SA_COMPONENTS.ALTER_COMPARTMENT

The `SA_COMPONENTS.ALTER_COMPARTMENT` procedure changes the short name and long name associated with a compartment.

Once set, the `comp_num` parameter cannot be changed. If the `comp_num` parameter is used in any existing label, then its short name *cannot* be changed but its long name *can* be changed.

Syntax

```
SA_COMPONENTS.ALTER_COMPARTMENT (
   policy_name       IN VARCHAR2,
   comp_num          IN NUMBER(38),
   new_short_name    IN VARCHAR2,
   new_long_name     IN VARCHAR2);

SA_COMPONENTS.ALTER_COMPARTMENT (
   policy_name       IN VARCHAR2,
   short_name        IN VARCHAR2 DEFAULT NULL,
   new_long_name     IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table D-6 SA_COMPONENTS.ALTER_COMPARTMENT Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`comp_num`	Specifies the number of the compartment to be altered. To find a list of existing compartment numbers, query the `COMP_NUM` column of the `ALL_SA_COMPARTMENTS` view.
`short_name`	Specifies the short name of the compartment to be altered (up to 30 characters). To find the current compartment, query the `SHORT_NAME` column of the `ALL_SA_COMPARTMENTS` view.

Table D-6 (Cont.) SA_COMPONENTS.ALTER_COMPARTMENT Parameters

Parameter	Description
`new_short_name`	Specifies the new short name of the compartment (up to 30 characters)
`new_long_name`	Specifies the new long name of the compartment (up to 80 characters).

Example

The following example modifies the `hr_ols_pol` policy.

```
BEGIN
  SA_COMPONENTS.ALTER_COMPARTMENT (
   policy_name          => 'hr_ols_pol',
   comp_num             => '48',
   new_short_name       => 'FIN',
   new_long_name        => 'FINANCE');
END;
/
```

D.2.3 SA_COMPONENTS.ALTER_GROUP

The `SA_COMPONENTS.ALTER_GROUP` procedure changes the short name and long name associated with a group.

Once set, the `group_num` parameter cannot be changed. If the group is used in any existing label, then its short name *cannot* be changed, but its long name *can* be changed.

Syntax

```
SA_COMPONENTS.ALTER_GROUP (
   policy_name    IN VARCHAR2,
   group_num      IN NUMBER(38),
   new_short_name IN VARCHAR2 DEFAULT NULL,
   new_long_name  IN VARCHAR2 DEFAULT NULL);

SA_COMPONENTS.ALTER_GROUP (
   policy_name    IN VARCHAR2,
   short_name     IN VARCHAR2,
   new_long_name  IN VARCHAR2);
```

Parameters

Table D-7 SA_COMPONENTS.ALTER_GROUP Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`group_num`	Specifies the existing group number to be altered. To find existing group numbers, query the `GROUP_NUM` column of the `ALL_SA_GROUPS` view.

Table D-7   (Cont.) SA_COMPONENTS.ALTER_GROUP Parameters

Parameter	Description
`short_name`	Specifies the existing group short name to be altered. To find existing short names, query the `SHORT_NAME` column of the `ALL_SA_GROUPS` view.
`new_short_name`	Specifies the new short name for the group (up to 30 characters)
`new_long_name`	Specifies the new long name for the group (up to 80 characters)

Example

The following example modifies the `long_name` setting for the `hr_ols_pol` policy.

```
BEGIN
  SA_COMPONENTS.ALTER_GROUP (
   policy_name     => 'hr_ols_pol',
   short_name      => 'ER_FIN',
   new_long_name   => 'ER_FINANCES');
END;
/
```

D.2.4 SA_COMPONENTS.ALTER_GROUP_PARENT

The `SA_COMPONENTS.ALTER_GROUP_PARENT` procedure changes the parent group associated with a particular group.

Syntax

```
SA_COMPONENTS.ALTER_GROUP_PARENT (
   policy_name     IN VARCHAR2,
   group_num       IN NUMBER(38),
   new_parent_num  IN NUMBER(38));

SA_COMPONENTS.ALTER_GROUP_PARENT (
   policy_name     IN VARCHAR2,
   group_num       IN NUMBER(38),
   new_parent_name IN VARCHAR2);

SA_COMPONENTS.ALTER_GROUP_PARENT (
   policy_name     IN VARCHAR2,
   short_name      IN VARCHAR2,
   new_parent_name IN VARCHAR2);
```

Parameters

Table D-8   SA_COMPONENTS.ALTER_GROUP_PARENT Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.

Table D-8 (Cont.) SA_COMPONENTS.ALTER_GROUP_PARENT Parameters

Parameter	Description
`group_num`	Specifies the existing group number to be altered. To find existing group numbers, query the `GROUP_NUM` column of the `ALL_SA_GROUPS` view.
`short_name`	Specifies the existing group short name to be altered. To find existing short names, query the `SHORT_NAME` column of the `ALL_SA_GROUPS` view.
`new_parent_num`	Specifies the number of an existing group as the parent group. To find existing parent groups, query the `PARENT_NUM` column of the `ALL_SA_GROUPS` view.
`new_parent_name`	Specifies the short name of an existing group as the parent group. To find existing groups, query the `SHORT_NAME` column of the `ALL_SA_GROUPS` view.

Example

The following example modifies the parent name for the `hr_ols_pol` policy.

```
BEGIN
  SA_COMPONENTS.ALTER_GROUP_PARENT (
   policy_name         => 'hr_ols_pol',
   group_num           => 2100,
   new_parent_name     => 'ER');
END;
/
```

D.2.5 SA_COMPONENTS.ALTER_LEVEL

The `SA_COMPONENTS.ALTER_LEVEL` procedure changes the short name and long name associated with a level.

Once they are defined, level numbers cannot be changed. If a level is used in any existing label, then its short name *cannot* be changed, but its long name *can* be changed.

Syntax

```
SA_COMPONENTS.ALTER_LEVEL (
   policy_name     IN VARCHAR2,
   level_num       IN NUMBER(38),
   new_short_name  IN VARCHAR2 DEFAULT NULL,
   new_long_name   IN VARCHAR2 DEFAULT NULL);

SA_COMPONENTS.ALTER_LEVEL (
   policy_name     IN VARCHAR2,
   short_name      IN VARCHAR2,
   new_long_name   IN VARCHAR2);
```

Parameters

Table D-9 SA_COMPONENTS.ALTER_LEVEL Parameters

Parameter	Description
`policy_name`	Specifies the policy, which much exist. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`level_num`	Specifies the number of the level to be altered. To find existing levels, query the `LEVEL_NUM` column of the `ALL_SA_LEVELS` view.
`short_name`	Specifies the existing short name of the level. To find existing level short names, query the `SHORT_NAME` column of the `ALL_SA_LEVELS` view.
`new_short_name`	Specifies the new short name for the level (up to 30 characters)
`new_long_name`	Specifies the new long name for the level (up to 80 characters)

Example

The following example modifies the short and long names for the `hr_ols_pol` policy level.

```
BEGIN
 SA_COMPONENTS.ALTER_LEVEL (
   policy_name     => 'hr_ols_pol',
   level_num       => 40,
   new_short_name  => 'TS',
   new_long_name   => 'TOP_SECRET');
END;
/
```

D.2.6 SA_COMPONENTS.CREATE_COMPARTMENT

The `SA_COMPONENTS.CREATE_COMPARTMENT` procedure creates a compartment and specify its short name and long name.

The `comp_num` parameter determines the order in which compartments are listed in the character string representation of labels.

Syntax

```
SA_COMPONENTS.CREATE_COMPARTMENT (
   policy_name IN VARCHAR2,
   comp_num    IN NUMBER(38),
   short_name  IN VARCHAR2,
   long_name   IN VARCHAR2);
```

Parameters

Table D-10 SA_COMPONENTS.CREATE_COMPARTMENT Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`comp_num`	Specifies the compartment number (0-9999)
`short_name`	Specifies the short name for the compartment (up to 30 characters)
`long_name`	Specifies the long name for the compartment (up to 80 characters)

Example

The following example creates a compartment for the `hr_ols_pol` policy.

```
BEGIN
  SA_COMPONENTS.CREATE_COMPARTMENT (
   policy_name     => 'hr_ols_pol',
   comp_num        => '48',
   short_name      => 'FIN',
   long_name       => 'FINANCE');
END;
/
```

D.2.7 SA_COMPONENTS.CREATE_GROUP

The `SA_COMPONENTS.CREATE_GROUP` procedure creates a group and specify its short name and long name, and optionally a parent group.

Syntax

```
SA_COMPONENTS.CREATE_GROUP (
   policy_name IN VARCHAR2,
   group_num   IN NUMBER(38),
   short_name  IN VARCHAR2,
   long_name   IN VARCHAR2,
   parent_name IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table D-11 SA_COMPONENTS.CREATE_GROUP Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`group_num`	Specifies the group number (0-9999)
`short_name`	Specifies the short name for the group (up to 30 characters)
`long_name`	Specifies the long name for the group (up to 80 characters)
`parent_name`	Specifies the short name of an existing group as the parent group. If `NULL`, then the group is a top-level group.

Note that the group number affects the order in which groups will be displayed when labels are selected.

Examples

In the following examples, the first creates a parent group, `ER`, and the second creates a second group that is part of the parent group.

```
BEGIN
  SA_COMPONENTS.CREATE_GROUP (
   policy_name     => 'hr_ols_pol',
   group_num       => 2000,
   short_name      => 'ER',
   long_name       => 'EAST_REGION');
END;
/

BEGIN
  SA_COMPONENTS.CREATE_GROUP (
   policy_name     => 'hr_ols_pol',
   group_num       => 2100,
   short_name      => 'ER_FIN',
   long_name       => 'ER_FINANCES',
   parent_name     => 'ER');
END;
/
```

D.2.8 SA_COMPONENTS.CREATE_LEVEL

The `SA_COMPONENTS.CREATE_LEVEL` procedure creates a level and specify its short name and long name.

The numeric values assigned to the `level_num` parameter determine the sensitivity ranking (that is, a lower number indicates less sensitive data).

Syntax

```
SA_COMPONENTS.CREATE_LEVEL (
   policy_name       IN VARCHAR2,
   level_num         IN NUMBER(38),
   short_name        IN VARCHAR2,
   long_name         IN VARCHAR2);
```

Parameters

Table D-12 SA_COMPONENTS.CREATE_LEVEL Parameters

Parameter	Description
`policy_name`	Specifies the policy, which must exist. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`level_num`	Specifies the level number (0-9999)
`short_name`	Specifies the short name for the level (up to 30 characters)
`long_name`	Specifies the long name for the level (up to 80 characters)

Example

The following example creates a level for the `hr_ols_pol` policy.

```
BEGIN
 SA_COMPONENTS.CREATE_LEVEL (
   policy_name   => 'hr_ols_pol',
   level_num     => 40,
   short_name    => 'HS',
   long_name     => 'HIGHLY_SENSITIVE');
END;
/
```

D.2.9 SA_COMPONENTS.DROP_COMPARTMENT

The `SA_COMPONENTS.DROP_COMPARTMENT` procedure removes a compartment.

If the compartment is used in any existing label, then it *cannot* be dropped. You can find all existing labels by querying the `LABEL` column of the `ALL_SA_DATA_LABELS` data dictionary view.

Syntax

```
SA_COMPONENTS.DROP_COMPARTMENT (
   policy_name IN VARCHAR2,
   comp_num    IN INTEGER);

SA_COMPONENTS.DROP_COMPARTMENT (
   policy_name IN VARCHAR2,
   short_name  IN VARCHAR2);
```

Parameters

Table D-13   SA_COMPONENTS.DROP_COMPARTMENT Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`comp_num`	Specifies the number of an existing compartment for the policy. To find existing compartment numbers, query the `COMP_NUM` column of the `DBA_SA_COMPARTMENTS` view.
`short_name`	Specifies the short name of an existing compartment for the policy. To find existing compartment short names, query the `SHORT_NAME` column of the `DBA_SA_COMPARTMENTS` view.

Example

The following example removes the `FIN` compartment from the `hr_ols_pol` policy.

```
BEGIN
  SA_COMPONENTS.DROP_COMPARTMENT (
   policy_name     => 'hr_ols_pol',
   short_name      => 'FIN');
END;
/
```

D.2.10 SA_COMPONENTS.DROP_GROUP

The `SA_COMPONENTS.DROP_GROUP` procedure removes a group.

If the group is used in an existing label, then it cannot be dropped.

Syntax

```
SA_COMPONENTS.DROP_GROUP (
   policy_name IN VARCHAR2,
   group_num   IN NUMBER(38));

SA_COMPONENTS.DROP_GROUP (
   policy_name IN VARCHAR2,
   short_name  IN VARCHAR2);
```

Parameters

Table D-14 SA_COMPONENTS.DROP_GROUP Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`group_num`	Specifies the number of an existing group for the policy. To find existing group numbers, query the `GROUP_NUM` column of the `ALL_SA_GROUPS` view.
`short_name`	Specifies the short name of an existing group. To find existing group short names, query the `SHORT_NAME` column of the `ALL_SA_GROUPS` view.

Example

The following example removes a group based on the group number for the `hr_ols_pol` policy.

```
BEGIN
  SA_COMPONENTS.DROP_GROUP (
   policy_name     => 'hr_ols_pol',
   group_num       => 2000);
END;
/
```

D.2.11 SA_COMPONENTS.DROP_LEVEL

The `SA_COMPONENTS.DROP_LEVEL` procedure removes a level.

If the level is used in any existing label, then it cannot be dropped.

Syntax

```
SA_COMPONENTS.DROP_LEVEL (
   policy_name IN VARCHAR2,
   level_num   IN NUMBER(38));
```

```
SA_COMPONENTS.DROP_LEVEL (
   policy_name IN VARCHAR2,
   short_name  IN VARCHAR2);
```

Parameters

Table D-15 SA_COMPONENTS.DROP_LEVEL Parameters

Parameter	Description
`policy_name`	Specifies the policy, which much exist. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`level_num`	Specifies the number of an existing level for the policy. To find existing level numbers, query the `LEVEL_NUM` column of the `ALL_SA_LEVELS` view.
`short_name`	Specifies the short name for the level (up to 30 characters). To find existing level short names, query the `SHORT_NAME` column of the `ALL_SA_LEVELS` view.

Example

The following example drops the level `40` from the `hr_ols_pol` policy.

```
BEGIN
 SA_COMPONENTS.DROP_LEVEL (
   policy_name     => 'hr_ols_pol',
   level_num       => 40);
END;
/
```

D.3 SA_LABEL_ADMIN Label Management PL/SQL Package

The `SA_LABEL_ADMIN` PL/SQL package provides an administrative interface to manage the labels used by a policy.

- About the SA_LABEL_ADMIN PL/SQL Package
  The `SA_LABEL_ADMIN` PL/SQL package creates, alters, and deletes labels.
- SA_LABEL_ADMIN.ALTER_LABEL
  The `SA_LABEL_ADMIN.ALTER_LABEL` procedure changes the character string label definition associated with a label tag.
- SA_LABEL_ADMIN.CREATE_LABEL
  The `SA_LABEL_ADMIN.CREATE_LABEL` procedure creates data labels.
- SA_LABEL_ADMIN.DROP_LABEL
  The `SA_LABEL_ADMIN.DROP_LABEL` procedure deletes a specified policy label.

D.3.1 About the SA_LABEL_ADMIN PL/SQL Package

The `SA_LABEL_ADMIN` PL/SQL package creates, alters, and deletes labels.

D.3.2 SA_LABEL_ADMIN.ALTER_LABEL

The `SA_LABEL_ADMIN.ALTER_LABEL` procedure changes the character string label definition associated with a label tag.

The label tag itself cannot be changed.

If you change the character string associated with a label tag, then the sensitivity of the data in the rows changes accordingly. For example, if the label character string `TS:A` with an associated label tag value of `4001` is changed to the label `TS:B`, then access to the data changes accordingly. This is true even when the label tag value (`4001`) has not changed. In this way, you can change the data's sensitivity without the need to update all the rows.

Ensure that when you specify a label to alter, you can refer to it either by its label tag or by its character string value.

Syntax

```
SA_LABEL_ADMIN.ALTER_LABEL (
   policy_name       IN VARCHAR2,
   label_tag         IN BINARY_INTEGER,
   new_label_value   IN VARCHAR2 DEFAULT NULL,
   new_data_label    IN BOOLEAN  DEFAULT NULL);

SA_LABEL_ADMIN.ALTER_LABEL (
   policy_name       IN VARCHAR2,
   label_value       IN VARCHAR2,
   new_label_value   IN VARCHAR2 DEFAULT NULL,
   new_data_label    IN BOOLEAN  DEFAULT NULL);
```

Parameters

Table D-16 SA_LABEL_ADMIN.ALTER_LABEL Parameters

Parameter	Description
`policy_name`	Specifies the name of an existing policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`label_tag`	Identifies the integer tag assigned to the label to be altered. To find existing label tags, query the `LABEL_TAG` column of the `ALL_SA_LABELS` view.
`label_value`	Identifies the existing character string representation of the label to be altered. To find the existing label values, query the `LABEL` column of the `ALL_SA_LABELS` view.
`new_label_value`	Specifies the new character string representation of the label value. If `NULL`, the existing value is not changed.
`new_data_label`	`TRUE` if the label can be used to label row data. If `NULL`, the existing value is not changed.

Example

The following example modifies the `label_tag` and `label_value` settings of `hr_ols_pol` policy.

```
BEGIN
  SA_LABEL_ADMIN.ALTER_LABEL (
   policy_name       => 'hr_ols_pol',
   label_tag         => 1111,
   new_label_value   => 'HS',
   new_data_label    => TRUE);
END;
/
```

D.3.3 SA_LABEL_ADMIN.CREATE_LABEL

The `SA_LABEL_ADMIN.CREATE_LABEL` procedure creates data labels.

Syntax

```
SA_LABEL_ADMIN.CREATE_LABEL (
   policy_name IN VARCHAR2,
   label_tag   IN BINARY_INTEGER,
   label_value IN VARCHAR2,
   data_label  IN BOOLEAN DEFAULT TRUE);
```

Parameters

Table D-17 SA_LABEL_ADMIN.CREATE_LABEL Parameters

Parameter	Description
`policy_name`	Specifies the name of an existing policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`label_tag`	Specifies a unique integer value representing the sort order of the label, relative to other policy labels (0-99999999). This value must be 1 to 8 digits long.
`label_value`	Specifies the character string representation of the label to be created. Use the short name of the level, compartment, and group. You can find these values by querying the `SHORT_NAME` column of the `ALL_SA_LEVELS`, `ALL_SA_COMPARTMENTS`, and `ALL_SA_GROUPS` views.
`data_label`	`TRUE` if the label can be used to label row data. Use this to define the label as valid for data.

When you identify valid labels, you specify which of all the possible combinations of levels, compartments, and groups can potentially be used to label data in tables.

Example

The following example creates a label for the `hr_ols_pol` policy.

```
BEGIN
  SA_LABEL_ADMIN.CREATE_LABEL (
   policy_name     => 'hr_ols_pol',
   label_tag       => 1111,
   label_value     => 'HS:FIN',
   data_label      => TRUE);
END;
/
```

Note:

If you create a new label by using the `TO_DATA_LABEL` procedure, then a system-generated label tag of 10 digits is generated automatically.

D.3.4 SA_LABEL_ADMIN.DROP_LABEL

The `SA_LABEL_ADMIN.DROP_LABEL` procedure deletes a specified policy label.

Any subsequent reference to the label (in data rows, or in user or program unit labels) will raise an invalid label error.

Use this procedure only while setting up labels, prior to data population. If you should inadvertently drop a label that is being used, you can recover it by disabling the policy, fixing the problem, and then re-enabling the policy.

Syntax

```
SA_LABEL_ADMIN.DROP_LABEL (
   policy_name        IN VARCHAR2,
   label_tag          IN BINARY_INTEGER);

SA_LABEL_ADMIN.DROP_LABEL (
   policy_name        IN VARCHAR2,
   label_value        IN VARCHAR2);
```

Parameters

Table D-18 SA_LABEL_ADMIN.DROP_LABEL Parameters

Parameter	Description
`policy_name`	Specifies the name of an existing policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`label_tag`	Specifies the integer tag assigned to the label to be dropped. To find existing label tags, query the `LABEL_TAG` column of the `ALL_SA_LABELS` view.
`label_value`	Specifies the string value of the label to be dropped. To find existing label values, query the `LABEL` column of the `ALL_SA_LABELS` view.

WARNING:

Do not drop a label that is in use anywhere in the database. You can find labels by querying the `ALL_SA_LABELS` data dictionary view.

Example

The following example drops the `hr_ols_pol` policy label based on its `label_tag` setting.

```
BEGIN
  SA_LABEL_ADMIN.DROP_LABEL (
   policy_name     => 'hr_ols_pol',
   label_tag       => 1111);
END;
/
```

D.4 SA_POLICY_ADMIN Policy Administration PL/SQL Package

The `SA_POLICY_ADMIN` PL/SQL package manages Oracle Label Security policies as a whole.

- About the SA_POLICY_ADMIN PL/SQL Package
  The `SA_POLICY_ADMIN` PL/SQL package configures schema and table policies, and performs subscribe and unsubscribe actions.
- SA_POLICY_ADMIN.ALTER_SCHEMA_POLICY
  The `SA_POLICY_ADMIN.ALTER_SCHEMA_POLICY` procedure changes the default enforcement options for the policy.
- SA_POLICY_ADMIN.APPLY_SCHEMA_POLICY
  The `SA_POLICY_ADMIN.APPLY_SCHEMA_POLICY` procedure applies a policy to all of the tables in a schema and enables the policy for these tables.
- SA_POLICY_ADMIN.APPLY_TABLE_POLICY
  The `SA_POLICY_ADMIN.APPLY_TABLE_POLICY` procedure adds the specified policy to a table.
- SA_POLICY_ADMIN.DISABLE_SCHEMA_POLICY
  The `SA_POLICY_ADMIN.DISABLE_SCHEMA_POLICY` procedure disables the enforcement of the policy for all tables in a schema.
- SA_POLICY_ADMIN.DISABLE_TABLE_POLICY
  The `SA_POLICY_ADMIN.DISABLE_TABLE_POLICY` procedure disables the enforcement of the policy for a table without changing the enforcement options, labeling function, or predicate values.
- SA_POLICY_ADMIN.ENABLE_SCHEMA_POLICY
  The `SA_POLICY_ADMIN.ENABLE_SCHEMA_POLICY` procedure reenables the current enforcement options, labeling function, and predicate for the tables in the specified schema.
- SA_POLICY_ADMIN.ENABLE_TABLE_POLICY
  The `SA_POLICY_ADMIN.ENABLE_TABLE_POLICY` procedure reenables the current enforcement options, labeling function, and predicate for the specified table.
- SA_POLICY_ADMIN.REMOVE_SCHEMA_POLICY
  The `SA_POLICY_ADMIN.REMOVE_SCHEMA_POLICY` procedure removes the specified policy from a schema.
- SA_POLICY_ADMIN.REMOVE_TABLE_POLICY
  The `SA_POLICY_ADMIN.REMOVE_TABLE_POLICY` procedure removes the specified policy from a table.

D.4.1 About the SA_POLICY_ADMIN PL/SQL Package

The `SA_POLICY_ADMIN` PL/SQL package configures schema and table policies, and performs subscribe and unsubscribe actions.

To use this package, you must be granted the *`policy`*`_DBA` role (for example, `HR_OLS_POL_DBA` for a role for the `hr_ols_pol` policy) and the `EXECUTE` privilege for the `SA_POLICY_ADMIN` package.

D.4.2 SA_POLICY_ADMIN.ALTER_SCHEMA_POLICY

The `SA_POLICY_ADMIN.ALTER_SCHEMA_POLICY` procedure changes the default enforcement options for the policy.

Any new tables created in the schema will automatically have the new enforcement options applied. The existing tables in the schema are not affected.

To change enforcement options on a table (rather than a schema), you must first drop the policy from the table, make the change, and then reapply the policy.

If you alter the enforcement options on a schema, then this will take effect the next time a table is created in the schema. As a result, different tables within a schema may have different policy enforcement options in force.

Syntax

```
SA_POLICY_ADMIN.ALTER_SCHEMA_POLICY (
  policy_name          IN VARCHAR2,
  schema_name          IN VARCHAR2,
  default_options      IN VARCHAR2);
```

Parameters

Table D-19   SA_POLICY_ADMIN.ALTER_SCHEMA Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`schema_name`	The schema that contains the table. To find existing schemas associated with this policy, query the `POLICY_NAME` and `SCHEMA_NAME` columns of the `ALL_SA_TABLE_POLICIES` view.
`default_options`	The default options to be used for new tables in the schema. Separate each option with a comma. See Table 10-2 for a listing of the default enforcement options.

Example

The following example adds the `UPDATE_CONTROL` default option to the `HR` schema.

```
BEGIN
 SA_POLICY_ADMIN.ALTER_SCHEMA_POLICY(
  policy_name      => 'hr_ols_pol',
  schema_name      => 'HR',
  default_options  => 'read_control, write_control, update_control');
END;
/
```

D.4.3 SA_POLICY_ADMIN.APPLY_SCHEMA_POLICY

The `SA_POLICY_ADMIN.APPLY_SCHEMA_POLICY` procedure applies a policy to all of the tables in a schema and enables the policy for these tables.

That is, it applies to those tables that do not already have the policy applied. Then, whenever a new table is created in the schema, the policy is automatically applied to that table, using the schema's default options. No changes are made to existing tables in the schema that already have the policy applied.

Syntax

```
SA_POLICY_ADMIN.APPLY_SCHEMA_POLICY (
  policy_name        IN VARCHAR2,
  schema_name        IN VARCHAR2,
  default_options    IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table D-20 SA_POLICY_ADMIN.APPLY_SCHEMA_POLICY Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`schema_name`	The schema that contains the table to protect
`default_options`	The default options to be used for tables in the schema. Separate each option with a comma. If the `default_options` parameter is `NULL`, then the policy's default options will be used to apply the policy to the tables in the schema. See Table 10-2 for a listing of the default enforcement options.

Example

The following example applies the `READ_CONTROL` and `WRITE_CONTROL` options to the `HR` schema.

```
BEGIN
 SA_POLICY_ADMIN.APPLY_SCHEMA_POLICY(
  policy_name      => 'hr_ols_pol',
  schema_name      => 'HR',
  default_options  => 'read_control, write_control');
END;
/
```

D.4.4 SA_POLICY_ADMIN.APPLY_TABLE_POLICY

The `SA_POLICY_ADMIN.APPLY_TABLE_POLICY` procedure adds the specified policy to a table.

A policy label column is added to the table if it does not exist, and is set to `NULL`. When a policy is applied, it is automatically enabled. To change the table options, labeling function, or predicate, you must first remove the policy, and then reapply it.

Syntax

```
SA_POLICY_ADMIN.APPLY_TABLE_POLICY (
  policy_name       IN VARCHAR2,
  schema_name       IN VARCHAR2,
  table_name        IN VARCHAR2,
  table_options     IN VARCHAR2 DEFAULT NULL,
  label_function    IN VARCHAR2 DEFAULT NULL,
  predicate         IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table D-21 SA_POLICY_ADMIN.APPLY_TABLE_POLICY Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`schema_name`	The schema that contains the table that the policy protects
`table_name`	The table to be protected by the policy
`table_options`	A comma-delimited list of policy enforcement options to be used for the table. If `NULL`, then the policy's default options are used. See Table 10-2 for a listing of the default enforcement options.
`label_function`	A string calling a function to return a label value to use as the default. For example, `my_label(:new.dept,:new.status)` computes the label based on the new values of the `DEPT` and `STATUS` columns in the row.
`predicate`	An additional predicate to combine (using `AND` or `OR`) with the label-based predicate for `READ_CONTROL`

Example

The following statement applies the `hr_ols_pol` policy to the `EMPLOYEES` table in the `HR` schema.

```
BEGIN
 SA_POLICY_ADMIN.APPLY_TABLE_POLICY(
  policy_name    => 'hr_ols_pol',
  schema_name    => 'HR',
  table_name     => 'EMPLOYEES',
  table_options  => NULL,
  label_function => 'hs(:new.dept,:new.status)',
  predicate      => 'no_control');
END;
/
```

D.4.5 SA_POLICY_ADMIN.DISABLE_SCHEMA_POLICY

The `SA_POLICY_ADMIN.DISABLE_SCHEMA_POLICY` procedure disables the enforcement of the policy for all tables in a schema.

However, it does not change the enforcement options, labeling function, or predicate values.

This procedure removes the row level security predicate and DML triggers from all the tables in the schema.

Syntax

```
SA_POLICY_ADMIN.DISABLE_SCHEMA_POLICY (
  policy_name     IN VARCHAR2,
  schema_name     IN VARCHAR2);
```

Parameters

Table D-22 SA_POLICY_ADMIN.DISABLE_SCHEMA_POLICY Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`schema_name`	The schema that contains the table for this policy. To find this schema, query the `POLICY_NAME` and `SCHEMA_NAME` columns of the `ALL_SA_TABLE_POLICIES` view.

Example

The following example disables the `hr_ols_pol` policy for the `HR` schema.

```
BEGIN
 SA_POLICY_ADMIN.DISABLE_SCHEMA_POLICY(
  policy_name      => 'hr_ols_pol',
  schema_name      => 'HR');
END;
/
```

D.4.6 SA_POLICY_ADMIN.DISABLE_TABLE_POLICY

The `SA_POLICY_ADMIN.DISABLE_TABLE_POLICY` procedure disables the enforcement of the policy for a table without changing the enforcement options, labeling function, or predicate values.

This procedure removes the row level security predicate and DML triggers from the table.

Syntax

```
SA_POLICY_ADMIN.DISABLE_TABLE_POLICY (
  policy_name      IN VARCHAR2,
  schema_name      IN VARCHAR2,
  table_name       IN VARCHAR2);
```

Parameters

Table D-23 SA_POLICY_ADMIN.DISABLE_TABLE_POLICY Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.

Table D-23 (Cont.) SA_POLICY_ADMIN.DISABLE_TABLE_POLICY Parameters

Parameter	Description
`schema_name`	The schema that contains the table. To find this schema, query the `POLICY_NAME` and `SCHEMA_NAME` columns of the `ALL_SA_TABLE_POLICIES` view.
`table_name`	The table in the schema specified by `schema_name`. To find this table, query the `POLICY_NAME`, `SCHEMA_NAME`, and `TABLE_NAME` columns of the `ALL_SA_TABLE_POLICIES` view.

Example

The following statement disables the `hr_ols_pos` policy on the `EMPLOYEES` table in the `HR` schema:

```
BEGIN
 SA_POLICY_ADMIN.DISABLE_TABLE_POLICY(
  policy_name   => 'hr_ols_pol',
  schema_name   => 'HR',
  table_name    => 'EMPLOYEES');
END;
/
```

D.4.7 SA_POLICY_ADMIN.ENABLE_SCHEMA_POLICY

The `SA_POLICY_ADMIN.ENABLE_SCHEMA_POLICY` procedure reenables the current enforcement options, labeling function, and predicate for the tables in the specified schema.

It accomplishes this by re-applying the row level security predicate and DML triggers. The result is similar to enabling a policy for a table, but it covers all the tables in the schema.

Syntax

```
SA_POLICY_ADMIN.ENABLE_SCHEMA_POLICY (
  policy_name     IN VARCHAR2,
  schema_name     IN VARCHAR2);
```

Parameters

Table D-24 SA_POLICY_ADMIN.ENABLE_SCHEMA_POLICY Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies and their status, query the `POLICY_NAME` and `STATUS` columns of the `ALL_SA_POLICIES` data dictionary view.
`schema_name`	The schema that contains the table. To find this schema, query the `POLICY_NAME` and `SCHEMA_NAME` columns of the `ALL_SA_TABLE_POLICIES` view.

Example

The following example enables the `hr_ols_pol` policy for the `HR` schema.

```
BEGIN
 SA_POLICY_ADMIN.ENABLE_SCHEMA_POLICY(
  policy_name      => 'hr_ols_pol',
  schema_name      => 'HR');
END;
/
```

D.4.8 SA_POLICY_ADMIN.ENABLE_TABLE_POLICY

The `SA_POLICY_ADMIN.ENABLE_TABLE_POLICY` procedure reenables the current enforcement options, labeling function, and predicate for the specified table.

It accomplishes this by reapplying the row level security predicate and DML triggers.

Syntax

```
SA_POLICY_ADMIN.ENABLE_TABLE_POLICY (
  policy_name     IN VARCHAR2,
  schema_name     IN VARCHAR2,
  table_name      IN VARCHAR2);
```

Parameters

Table D-25 SA_POLICY_ADMIN.ENABLE_TABLE_POLICY Parameters

Parameter	Description
`policy_name`	Specifies the policy. `POLICY_NAME` and `STATUS` columns of the `ALL_SA_POLICIES` data dictionary view.
`schema_name`	The schema that contains the table. To find this schema, query the `POLICY_NAME` and `SCHEMA_NAME` columns of the `ALL_SA_TABLE_POLICIES` view.
`table_name`	The table in the schema specified by `schema_name`. To find this table, query the `POLICY_NAME`, `SCHEMA_NAME`, and `TABLE_NAME` columns of the `ALL_SA_TABLE_POLICIES` view.

Example

The following statement reenables the `hr_ols_pol` policy on the `EMPLOYEES` table in the `HR` schema:

```
BEGIN
 SA_POLICY_ADMIN.ENABLE_TABLE_POLICY(
  policy_name   => 'hr_ols_pol',
  schema_name   => 'HR',
  table_name    => 'EMPLOYEES');
END;
/
```

D.4.9 SA_POLICY_ADMIN.REMOVE_SCHEMA_POLICY

The `SA_POLICY_ADMIN.REMOVE_SCHEMA_POLICY` procedure removes the specified policy from a schema.

The policy will be removed from all the tables in the schema and, optionally, the label column for the policy will be dropped from all the tables.

Syntax

```
SA_POLICY_ADMIN.REMOVE_SCHEMA_POLICY (
  policy_name     IN VARCHAR2,
  schema_name     IN VARCHAR2,
  drop_column     IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table D-26 SA_POLICY_ADMIN.REMOVE_SCHEMA_POLICY Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`schema_name`	The schema that contains the table associated with this policy. To find this schema, query the `SCHEMA_NAME` of the `ALL_SA_SCHEMA_POLICIES` view.
`drop_column`	If `TRUE`, then the policy's column will be dropped from the tables, otherwise, the column will remain.

Example

The following example drops the `human_resource` policy's column from the `HR` schema.

```
BEGIN
 SA_POLICY_ADMIN.REMOVE_SCHEMA_POLICY(
  policy_name      => 'hr_ols_pol',
  schema_name      => 'HR',
  drop_column      => TRUE);
END;
/
```

D.4.10 SA_POLICY_ADMIN.REMOVE_TABLE_POLICY

The `SA_POLICY_ADMIN.REMOVE_TABLE_POLICY` procedure removes the specified policy from a table.

The policy predicate and any DML triggers will be removed from the table, and the policy label column can optionally be dropped. Policies can be removed from tables belonging to a schema that is protected by the policy.

Syntax

```
SA_POLICY_ADMIN.REMOVE_TABLE_POLICY (
policy_name        IN VARCHAR2,
schema_name        IN VARCHAR2,
table_name          IN VARCHAR2,
drop_column         IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table D-27   SA_POLICY_ADMIN.REMOVE_TABLE_POLICY Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`schema_name`	The schema that contains the table associated with this policy. To find this schema, query the `SCHEMA_NAME` of the `ALL_SA_SCHEMA_POLICIES` view.
`table_name`	The table in the schema specified by `schema_name`. To find this table query the `POLICY_NAME`, `SCHEMA_NAME`, and `TABLE_NAME` columns of the `ALL_SA_TABLE_POLICIES` view.
`drop_column`	Whether the column is to be dropped: if `TRUE`, then the policy's column will be dropped from the table, otherwise, it will remain

Example

The following statement removes the `hr_ols_pol` policy from the `EMPLOYEES` table in the `HR` schema:

```
BEGIN
 SA_POLICY_ADMIN.REMOVE_TABLE_POLICY(
  policy_name     => 'hr_ols_pol',
  schema_name     => 'HR',
  table_name      => 'EMPLOYEES',
  drop_column     => TRUE);
END;
/
```

D.5 SA_SESSION Session Management PL/SQL Package

The `SA_SESSION` PL/SQL package manages session behavior for user authorizations.

- About the SA_SESSION PL/SQL Package
  The `SA_SESSION` PL/SQL package manages user name, levels, labels, and read and write permissions for a user session.
- SA_SESSION.COMP_READ
  The `SA_SESSION.COMP_READ` function returns a comma-delimited list of compartments that the user is authorized to read.
- SA_SESSION.COMP_WRITE
  The `SA_SESSION.COMP_WRITE` function returns a comma-delimited list of compartments to which the user is authorized to write.
- SA_SESSION.GROUP_READ
  The `SA_SESSION.GROUP_READ` function returns a comma-delimited list of groups that the user is authorized to read.
- SA_SESSION.GROUP_WRITE
  The `SA_SESSION.GROUP_WRITE` function returns a comma-delimited list of groups that the user is authorized to write.

- SA_SESSION.LABEL
  The `SA_SESSION.LABEL` function returns the label that is associated with the specified policy for the current session.
- SA_SESSION.MAX_LEVEL
  The `SA_SESSION.MAX_LEVEL` function returns the maximum Oracle Label Security level authorized for the session.
- SA_SESSION.MAX_READ_LABEL
  The `SA_SESSION.MAX_READ_LABEL` function returns the label string that was used to initialize the user's maximum authorized read label.
- SA_SESSION.MAX_WRITE_LABEL
  The `SA_SESSION.MAX_WRITE_LABEL` function returns the label string that was used to initialize the user's maximum authorized write label.
- SA_SESSION.MIN_LEVEL
  The `SA_SESSION.MIN_LEVEL` function returns the minimum Oracle Label Security level authorized for the session.
- SA_SESSION.MIN_WRITE_LABEL
  The `SA_SESSION.MIN_WRITE_LABEL` function retrieves the label string that was used to initialize the user's minimum authorized write label.
- SA_SESSION.PRIVS
  The `SA_SESSION.PRIVS` function returns the set of current session privileges, in a comma-delimited list.
- SA_SESSION.RESTORE_DEFAULT_LABELS
  The `SA_SESSION.RESTORE_DEFAULT_LABELS` procedure restores the session label and row label to those stored in the data dictionary.
- SA_SESSION.ROW_LABEL
  The `SA_SESSION.ROW_LABEL` function returns the name of the row label that is associated with the policy for the current session.
- SA_SESSION.SET_LABEL
  The `SA_SESSION.SET_LABEL` procedure sets the label of the current database session.
- SA_SESSION.SA_USER_NAME
  The `SA_SESSION.SA_USER_NAME` function returns the name of the current Oracle Label Security user, as set by the `SA_SESSION.SET_ACCESS_PROFILE` procedure (or as established at login).
- SA_SESSION.SAVE_DEFAULT_LABELS
  The `SA_SESSION.SAVE_DEFAULT_LABELS` procedure stores the current session label and row label as your initial session label and default row label.
- SA_SESSION.SET_ACCESS_PROFILE
  The `SA_SESSION.SET_ACCESS_PROFILE` procedure sets the Oracle Label Security authorizations and privileges of the database session to those of the specified user.
- SA_SESSION.SET_ROW_LABEL
  The `SA_SESSION.SET_ROW_LABEL` procedure sets the default row label value for the current database session.

D.5.1 About the SA_SESSION PL/SQL Package

The `SA_SESSION` PL/SQL package manages user name, levels, labels, and read and write permissions for a user session.

Users can change labels during a session within the authorizations set by the administrator.

You do not need special privileges to use this package.

See Also:

SA_UTL PL/SQL Utility Functions and Procedures for additional functions that return numeric label tags and `BOOLEAN` values

D.5.2 SA_SESSION.COMP_READ

The `SA_SESSION.COMP_READ` function returns a comma-delimited list of compartments that the user is authorized to read.

Syntax

```
SA_SESSION.COMP_READ (
  policy_name IN VARCHAR2)
RETURN VARCHAR2;
```

Parameter

Table D-28 SA_SESSION.COMP_READ Parameter

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.

Example

The following example returns the compartments that the user can read for the `hr_ols_pol` policy.

```
SELECT SA_SESSION.COMP_READ ('hr_ols_pol') FROM DUAL;
```

D.5.3 SA_SESSION.COMP_WRITE

The `SA_SESSION.COMP_WRITE` function returns a comma-delimited list of compartments to which the user is authorized to write.

This function is a subset of `SA_SESSION.COMP_READ`.

Syntax

```
SA_SESSION.COMP_WRITE (
  policy_name IN VARCHAR2)
RETURN VARCHAR2;
```

Parameter

Table D-29   SA_SESSION.COMP_WRITE Parameter

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.

Example

The following example returns the compartments that the user can modify for the `hr_ols_pol` policy.

```
SELECT SA_SESSION.COMP_WRITE ('hr_ols_pol') FROM DUAL;
```

D.5.4 SA_SESSION.GROUP_READ

The `SA_SESSION.GROUP_READ` function returns a comma-delimited list of groups that the user is authorized to read.

Syntax

```
SA_SESSION.GROUP_READ (
  policy_name IN VARCHAR2)
RETURN VARCHAR2;
```

Parameter

Table D-30   SA_SESSION.GROUP_READ Parameter

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.

Example

The following example returns the list of groups that a user can read for the `hr_ols_pol` policy.

```
SELECT SA_SESSION.GROUP_READ ('hr_ols_pol') FROM DUAL;
```

D.5.5 SA_SESSION.GROUP_WRITE

The `SA_SESSION.GROUP_WRITE` function returns a comma-delimited list of groups that the user is authorized to write.

This function is a subset of `SA_SESSION.GROUP_READ`.

Syntax

```
SA_SESSION.GROUP_WRITE (
  policy_name IN VARCHAR2)
RETURN VARCHAR2;
```

Parameter

Table D-31 SA_SESSION.GROUP_WRITE Parameter

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.

Example

The following example returns the groups the user is authorized to modify for the `hr_ols_pol` policy.

```
SELECT SA_SESSION.GROUP_WRITE ('hr_ols_pol') FROM DUAL;
```

D.5.6 SA_SESSION.LABEL

The `SA_SESSION.LABEL` function returns the label that is associated with the specified policy for the current session.

Syntax

```
SA_SESSION.LABEL (
  policy_name IN VARCHAR2)
RETURN VARCHAR2;
```

Parameter

Table D-32 SA_SESSION.LABEL Parameter

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.

Example

The following example returns the label that is associated with the `hr_ols_pol` policy.

```
SELECT SA_SESSION.LABEL ('hr_ols_pol') FROM DUAL;
```

D.5.7 SA_SESSION.MAX_LEVEL

The `SA_SESSION.MAX_LEVEL` function returns the maximum Oracle Label Security level authorized for the session.

Syntax

```
SA_SESSION.MAX_LEVEL (
  policy_name IN VARCHAR2)
RETURN VARCHAR2;
```

Parameter

Table D-33 SA_SESSION.MAX_LEVEL Parameter

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.

Example

The following example returns the maximum Oracle Label Security level that is authorized for the `hr_ols_pol` policy.

```
SELECT SA_SESSION.MAX_LEVEL ('hr_ols_pol') FROM DUAL;
```

D.5.8 SA_SESSION.MAX_READ_LABEL

The `SA_SESSION.MAX_READ_LABEL` function returns the label string that was used to initialize the user's maximum authorized read label.

The return string is composed of the user's maximum level, compartments authorized for read access, and groups authorized for read access.

Syntax

```
SA_SESSION.MAX_READ_LABEL (
  policy_name IN VARCHAR2)
RETURN VARCHAR2;
```

Parameter

Table D-34 SA_SESSION.MAX_READ_LABEL Parameter

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.

Example

The following example returns the maximum read label privileges for the `hr_ols_pol` policy.

```
SELECT SA_SESSION.MAX_READ_LABEL ('hr_ols_pol') FROM DUAL;
```

D.5.9 SA_SESSION.MAX_WRITE_LABEL

The `SA_SESSION.MAX_WRITE_LABEL` function returns the label string that was used to initialize the user's maximum authorized write label.

This return string is composed of the user's maximum level, compartments authorized for write access, and groups authorized for write access.

Syntax

```
SA_SESSION.MAX_WRITE_LABEL (
  policy_name IN VARCHAR2)
RETURN VARCHAR2;
```

Parameter

Table D-35 SA_SESSION.MAX_WRITE_LABEL Parameter

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.

Example

The following example returns the maximum write label privileges for the `hr_ols_pol` policy.

```
SELECT SA_SESSION.MAX_WRITE_LABEL ('hr_ols_pol') FROM DUAL;
```

D.5.10 SA_SESSION.MIN_LEVEL

The `SA_SESSION.MIN_LEVEL` function returns the minimum Oracle Label Security level authorized for the session.

Syntax

```
SA_SESSION.MIN_LEVEL (
  policy_name IN VARCHAR2)
RETURN VARCHAR2;
```

Parameter

Table D-36 SA_SESSION.MIN_LEVEL Parameter

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.

Example

The following example returns the current minimum level for the `hr_ols_pol` policy.

```
SELECT SA_SESSION.MIN_LEVEL ('hr_ols_pol') FROM DUAL;
```

D.5.11 SA_SESSION.MIN_WRITE_LABEL

The `SA_SESSION.MIN_WRITE_LABEL` function retrieves the label string that was used to initialize the user's minimum authorized write label.

The return string contains only the level, with no compartments or groups.

Syntax

```
SA_SESSION.MIN_WRITE_LABEL (
  policy_name IN VARCHAR2)
RETURN VARCHAR2;
```

Parameter

Table D-37 SA_SESSION.MIN_WRITE_LABEL Parameter

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.

Example

The following example returns the maximum write label privileges for the `hr_ols_pol` policy.

```
SELECT SA_SESSION.MIN_WRITE_LABEL ('hr_ols_pol') FROM DUAL;
```

D.5.12 SA_SESSION.PRIVS

The `SA_SESSION.PRIVS` function returns the set of current session privileges, in a comma-delimited list.

Syntax

```
SA_SESSION.PRIVS (
  policy_name IN VARCHAR2)
RETURN VARCHAR2;
```

Parameter

Table D-38 SA_SESSION.Privs Parameter

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.

Example

The following example returns the current session privileges for the `hr_ols_pol` policy.

```
SELECT SA_SESSION.PRIVS ('hr_ols_pol') FROM DUAL;
```

D.5.13 SA_SESSION.RESTORE_DEFAULT_LABELS

The `SA_SESSION.RESTORE_DEFAULT_LABELS` procedure restores the session label and row label to those stored in the data dictionary.

This command is useful to reset values after a `SA_SESSION.SET_LABEL` command has been processed.

Syntax

```
SA_SESSION.RESTORE_DEFAULT_LABELS (
 policy_name in VARCHAR2);
```

Parameter

Table D-39 SA_SESSION.RESTORE_DEFAULT_LABEL Parameter

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.

Example

The following example restores the default labels for the `hr_ols_pol` policy.

```
BEGIN
 SA_SESSION.RESTORE_DEFAULT_LABELS (
  policy_name         => 'hr_ols_pol');
END;
/
```

D.5.14 SA_SESSION.ROW_LABEL

The `SA_SESSION.ROW_LABEL` function returns the name of the row label that is associated with the policy for the current session.

Syntax

```
SA_SESSION.ROW_LABEL (
  policy_name IN VARCHAR2)
RETURN VARCHAR2;
```

Parameter

Table D-40 SA_SESSION.ROW_LABEL Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.

Example

The following example returns the row label that is associated with the `hr_ols_pol` policy.

```
SELECT SA_SESSION.ROW_LABEL ('hr_ols_pol') FROM DUAL;
```

D.5.15 SA_SESSION.SET_LABEL

The `SA_SESSION.SET_LABEL` procedure sets the label of the current database session.

You can set the session label to:

- Any level equal to or less than the maximum, and equal to or greater than the minimum level
- Include any compartments in the authorized compartment list
- Include any groups in the authorized group list. (Subgroups of authorized groups are implicitly included in the authorized list.)

Note that if you change the session label, this change may affect the value of the session's row label. The session's row label contains the subset of compartments and groups for which the user has write access. This may or may not be equivalent to the session label. For example, if you use the `SA_SESSION.SET_LABEL` procedure to set your current session label to `C:A,B:US` and you have write access only on the `A` compartment, then your row label would be set to `C:A`.

Syntax

```
SA_SESSION.SET_LABEL (
 policy_name IN VARCHAR2,
 label       IN VARCHAR2);
```

Parameters

Table D-41 SA_SESSION.SET_LABEL Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`label`	The value to set as the label

Example

The following example sets the label for the `hr_ols_pol` policy.

```
BEGIN
 SA_SESSION.SET_LABEL (
  policy_name          => 'hr_ols_pol',
  label                => 'C:A,B:US');
END;
/
```

Related Topics

- SA_USER_ADMIN.SET_DEFAULT_LABEL
  The `SA_USER_ADMIN.SET_DEFAULT_LABEL` procedure sets the user's initial session label to the one specified.

D.5.16 SA_SESSION.SA_USER_NAME

The `SA_SESSION.SA_USER_NAME` function returns the name of the current Oracle Label Security user, as set by the `SA_SESSION.SET_ACCESS_PROFILE` procedure (or as established at login).

This is how you can determine the identity of the current user in relation to Oracle Label Security, rather than in relation to your Oracle login name.

Syntax

```
SA_SESSION.SA_USER_NAME (
  policy_name IN VARCHAR2)
RETURN VARCHAR2;
```

Parameter

Table D-42   SA_SESSION.SA_USER_NAME Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.

Example

The following example finds the name of the Oracle Label Security user for the `hr_ols_pol` policy.

```
SELECT SA_SESSION.SA_USER_NAME ('hr_ols_pol') FROM DUAL;
```

D.5.17 SA_SESSION.SAVE_DEFAULT_LABELS

The `SA_SESSION.SAVE_DEFAULT_LABELS` procedure stores the current session label and row label as your initial session label and default row label.

This procedure permits you to change your defaults to reflect your current session label and row label. The saved labels will be used as the initial default settings for future sessions.

When you log in to a database, your default session label and row label are used to initialize the session label and row label. When the administrator originally authorized your Oracle Label Security labels, they also defined your default level, default compartments, and default groups. If you change your session label and row label, and want to save these values as the default labels, you can use the `SA_SESSION.SAVE_DEFAULT_LABELS` procedure.

This procedure is useful if you have multiple sessions and want to be sure that all additional sessions have the same labels. You can save the current labels as the default, and all future sessions will have these as the initial labels.

Consider a situation in which you connect to the database through Oracle Forms and want to run a report. By saving the current session labels as the default before you call Oracle Reports, you ensure that Oracle Reports will initialize at the same labels as are being used by Oracle Forms.

Syntax

```
SA_SESSION.SAVE_DEFAULT_LABELS (
  policy_name IN VARCHAR2);
```

Parameter

Table D-43 SA_SESSION.SAVE_DEFAULT_LABELS Parameter

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.

Example

The following example saves the label defaults for the `hr_ols_pol` policy.

```
BEGIN
 SA_SESSION.SAVE_DEFAULT_LABELS (
  policy_name       => 'hr_ols_pol');
END;
/
```

Note:

The `SA_SESSION.SAVE_DEFAULT_LABELS` procedure overrides the settings established by the administrator.

D.5.18 SA_SESSION.SET_ACCESS_PROFILE

The `SA_SESSION.SET_ACCESS_PROFILE` procedure sets the Oracle Label Security authorizations and privileges of the database session to those of the specified user.

Note that the originating user retains the `PROFILE_ACCESS` privilege.

The user who runs the SA_SESSION.SET_ACCESS_PROFILE procedure must have the PROFILE_ACCESS privilege. The logged-in database user (the Oracle user ID) does not change. That user assumes only the authorizations and privileges of the specified user. By contrast, the Oracle Label Security user name *is* changed.

This administrative procedure is useful for various tasks:

- With SA_SESSION.SET_ACCESS_PROFILE, you can see the result of the authorization and privilege settings for a particular user.
- Applications need to have proxy accounts connect as (and assume the identity of) application users, for purposes of accessing labeled data. With the SA_SESSION.SET_ACCESS_PROFILE privilege, the proxy account can act on behalf of the application users.

Syntax

```
SA_SESSION.SET_ACCESS_PROFILE (
  policy_name IN VARCHAR2
  user_name   IN VARCHAR2);
```

Parameters

Table D-44 SA_SESSION.SET_ACCESS_PROFILE Parameters

Parameter	Description
policy_name	Specifies the policy. To find existing policies, query the POLICY_NAME column of the ALL_SA_POLICIES data dictionary view.
user_name	Name of the user whose authorizations and privileges should be assumed (typically, the user associated with this policy). To find this user, query the USER_NAME and POLICY_NAME columns of the DBA_SA_USERS view.

Example

The following example enables user psmith to have Oracle Label Security authorizations and privileges for the database session.

```
BEGIN
 SA_SESSION.SET_ACCESS_PROFILE (
  policy_name       => 'hr_ols_pol',
  user_name         => 'jjones');
END;
/
```

D.5.19 SA_SESSION.SET_ROW_LABEL

The SA_SESSION.SET_ROW_LABEL procedure sets the default row label value for the current database session.

The compartments and groups in the label must be a subset of the compartments and groups in the session label to which the user has write access. When the LABEL_DEFAULT option is set, this row label value is used on insert if the user does not explicitly specify the label.

If the SA_SESSION.SET_ROW_LABEL procedure is not used to set the default row label value, then this value is automatically derived from the session label. It contains the level of the session label and the subset of the compartments and groups in the session label for which the user has write authorization.

The row label is automatically reset if the session label changes. For example, if you change your session level from `HIGHLY_SENSITIVE` to `SENSITIVE`, then the level component of the row label automatically changes to `SENSITIVE`.

The user can set the row label independently, but only to include:

- A level that is less than or equal to the level of the session label, and greater than or equal to the user's minimum level
- A subset of the compartments and groups from the session label, for which the user is authorized to have write access

If the user tries to set the row label to an invalid value, then the operation is not permitted and the row label value is unchanged.

Syntax

```
SA_SESSION.SET_ROW_LABEL (
 policy_name    IN VARCHAR2,
 row_label      IN VARCHAR2);
```

Parameters

Table D-45   SA_SESSION.SET_ROW_LABEL Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`label`	The value to set as the default row label

Example

The following example sets the row label for the `hr_ols_pol` policy.

```
BEGIN
 SA_SESSION.SET_ROW_LABEL (
  policy_name     => 'hr_ols_pol',
  label           => 'HR');
END;
/
```

Related Topics

- SA_USER_ADMIN.SET_ROW_LABEL
  The `SA_USER_ADMIN.SET_ROW_LABEL` procedure sets a user's initial row label to the one specified.

D.6 SA_SYSDBA Policy Management PL/SQL Package

The `SA_SYSDBA` PL/SQL package manages Oracle Label Security policies.

- About the SA_SYSDBA PL/SQL Package
  The `SA_SYSDBA` PL/SQL package creates, modifies, enables or disables, and drops Oracle Label Security policies.
- SA_SYSDBA.ALTER_POLICY
  The `SA_SYSDBA.ALTER_POLICY` procedure sets and modifies column names that are associated with the policy.
- SA_SYSDBA.CREATE_POLICY
  The `SA_SYSDBA.CREATE_POLICY` procedure creates a new Oracle Label Security policy, defines a policy-specific column name, and specifies default policy options.
- SA_SYSDBA.DISABLE_POLICY
  The `SA_SYSDBA.DISABLE_POLICY` procedure turns off enforcement of a policy, without removing it from the database.
- SA_SYSDBA.DROP_POLICY
  The `SA_SYSDBA.DROP_POLICY` procedure deletes the policy and its associated user labels and data labels from the database.
- SA_SYSDBA.ENABLE_POLICY
  The `SA_SYSDBA.ENABLE_POLICY` procedure enforces access control on the tables and schemas protected by the policy.

D.6.1 About the SA_SYSDBA PL/SQL Package

The `SA_SYSDBA` PL/SQL package creates, modifies, enables or disables, and drops Oracle Label Security policies.

To use this package, you must be granted the `LBAC_DBA` role and the `EXECUTE` privilege on the `SA_SYSDBA` package. The `SA_SYSDBA` package is an invoker's rights package, so you must provide the following `INHERIT PRIVILEGES` grant to the user `SYS` before you can use this package:

```
GRANT INHERIT PRIVILEGES ON USER SYS TO LBACSYS;
```

You only need to grant this privilege on user `SYS.` You do not need to grant it on other users.

D.6.2 SA_SYSDBA.ALTER_POLICY

The `SA_SYSDBA.ALTER_POLICY` procedure sets and modifies column names that are associated with the policy.

`SA_SYSDBA.ALTER_POLICY` can only be used to change column name for policies that are not applied on any user tables or schemas. Otherwise, this error appears:

```
12474, 00000, "cannot change column name for a policy in use"
```

Syntax

```
SA_SYSDBA.ALTER_POLICY (
   policy_name       IN  VARCHAR2,
   default_options   IN  VARCHAR2 DEFAULT NULL,
   column_name       IN  VARCHAR2 DEFAULT NULL);
```

Parameters

Table D-46 SA_SYSDBA.ALTER_POLICY Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`default_options`	Specifies the default enforcement options to be used when the policy is applied and no table- or schema-specific options are specified. Includes enforcement options and the option to hide the label column. Separate each option with a comma. See Categories of Policy Enforcement Options for a listing of the default enforcement options.
`column_name`	Specifies the column name associated with the policy. To find this column name, query the `COLUMN_NAME` column of the `ALL_SA_POLICIES` view.

Example

The following example updates the `hr_ols_pol` policy to use a different set of default options. Because the name of the column does not need to change, the `column_name` parameter is omitted.

```
BEGIN
 SA_SYSDBA.ALTER_POLICY (
  policy_name      => 'hr_ols_pol',
  default_options  => 'read_control, delete_control');
END;
/
```

D.6.3 SA_SYSDBA.CREATE_POLICY

The `SA_SYSDBA.CREATE_POLICY` procedure creates a new Oracle Label Security policy, defines a policy-specific column name, and specifies default policy options.

After you create the policy, a role for it is created and granted to you. The format of the role name is *`policy`*`_DBA` (for example, `my_ols_pol_DBA`).

Syntax

```
SA_SYSDBA.CREATE_POLICY (
   policy_name       IN VARCHAR2,
   column_name       IN VARCHAR2 DEFAULT NULL,
   default_options   IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table D-47   SA_SYSDBA.CREATE_POLICY Parameters

Parameter	Description
`policy_name`	Specifies the policy name, which must be unique within the database. It can have a maximum of 30 characters, but only the first 26 characters in the `policy_name` are significant. Two policies may not have the same first 26 characters in the `policy_name`. To find a list of existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`column_name`	Specifies the name of the column to be added to tables protected by the policy. If `NULL`, then the name *`policy_name`*`_COL` is used. Two Oracle Label Security policies cannot share the same column name.
`default_options`	Specifies the default options to be used when the policy is applied and no table- or schema-specific options are specified. Includes enforcement options and the option to hide the label column. Separate each option with a comma. See Categories of Policy Enforcement Options for a listing of the default enforcement options.

Example

The following example creates a policy container whose default options are `READ_CONTROL` and `WRITE_CONTROL`. The `WRITE_CONTROL` option encompasses the `INSERT_CONTROL`, `UPDATE_CONTROL`, and `DELETE_CONTROL` options.

```
BEGIN
 SA_SYSDBA.CREATE_POLICY (
  policy_name      => 'hr_ols_pol',
  column_name      => 'ols_col',
  default_options  => 'read_control, write_control');
END;
/
```

D.6.4 SA_SYSDBA.DISABLE_POLICY

The `SA_SYSDBA.DISABLE_POLICY` procedure turns off enforcement of a policy, without removing it from the database.

The policy is not enforced for all subsequent access to the database.

To disable a policy means that no access control is enforced on the tables and schemas protected by the policy. The administrator can continue to perform administrative operations while the policy is disabled.

Note:

This feature is extremely powerful, and should be used with caution. When a policy is disabled, anyone who connects to the database can access all the data normally protected by the policy. So, your site should establish guidelines for use of this feature.

Normally, a policy should not be disabled in order to manage data. At times, however, an administrator may need to disable a policy to perform application debugging tasks. In this case, the database should be run in single-user mode. In a development environment, for example, you may need to observe data processing operations without the policy turned on. When you reenable the policy, all of the selected enforcement options become effective again.

Syntax

```
SA_SYSDBA.DISABLE_POLICY (
 policy_name IN VARCHAR2);
```

Parameters

Table D-48 SA_SYSDBA.DISABLE_POLICY Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies and their status, query the `POLICY_NAME` and `STATUS` columns of the `ALL_SA_POLICIES` data dictionary view.

Example

The following example disables the `hr_ols_pol` policy:

```
EXEC SA_SYSDBA.DISABLE_POLICY ('hr_ols_pol');
```

D.6.5 SA_SYSDBA.DROP_POLICY

The `SA_SYSDBA.DROP_POLICY` procedure deletes the policy and its associated user labels and data labels from the database.

This procedure purges the policy and these associations from the system entirely. You can optionally drop the label column from all tables controlled by the policy. The policy does not need to be disabled before you drop it.

Syntax

```
SA_SYSDBA.DROP_POLICY (
   policy_name IN VARCHAR2,
   drop_column  BOOLEAN DEFAULT FALSE);
```

Parameters

Table D-49 SA_SYSDBA.DROP_POLICY Parameters

Parameter	Description
`policy_name`	Specifies the policy to be dropped. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`drop_column`	Indicates that the policy column should be dropped from protected tables (`TRUE`)

Example

The following example deletes the `hr_ols_pol` policy.

```
EXEC SA_SYSDBA.DROP_POLICY ('hr_ols_pol');
```

D.6.6 SA_SYSDBA.ENABLE_POLICY

The `SA_SYSDBA.ENABLE_POLICY` procedure enforces access control on the tables and schemas protected by the policy.

A policy is automatically enabled when it is created. After creation or enablement, the policy is enforced for all subsequent access to tables protected by the policy.

Syntax

```
SA_SYSDBA.ENABLE_POLICY (policy_name IN VARCHAR2);
```

Parameters

Table D-50 SA_SYSDBA.ENABLE_POLICY Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies and their status, query the `POLICY_NAME` and `STATUS` columns of the `ALL_SA_POLICIES` data dictionary view.

Example

The following example enables the `hr_ols_pol` policy.

```
EXEC SA_SYSDBA.ENABLE_POLICY('hr_ols_pol');
```

D.7 SA_USER_ADMIN PL/SQL Package

The `SA_USER_ADMIN` PL/SQL package manages user labels by label component.

- About the SA_USER_ADMIN PL/SQL Package
  The `SA_USER_ADMIN` PL/SQL package configures compartments, groups, user access, labels, levels, and privileges.

- SA_USER_ADMIN.ADD_COMPARTMENTS
  The `SA_USER_ADMIN.ADD_COMPARTMENTS` procedure adds (assigns) compartments to a user's authorizations, indicating if the compartments are authorized for write and read privileges.
- SA_USER_ADMIN.ADD_GROUPS
  The `SA_USER_ADMIN.ADD_GROUPS` procedure adds (assigns) groups to a user, indicating if the groups are authorized for write and read privileges.
- SA_USER_ADMIN.ALTER_COMPARTMENTS
  The `SA_USER_ADMIN.ALTER_COMPARTMENTS` procedure changes the write access, default label indicator, and row label indicator for the specified compartments.
- SA_USER_ADMIN.ALTER_GROUPS
  The `SA_USER_ADMIN.ALTER_GROUPS` procedure changes the write access, default label indicator, and row label indicator for the specified groups.
- SA_USER_ADMIN.DROP_ALL_COMPARTMENTS
  The `SA_USER_ADMIN.DROP_ALL_COMPARTMENTS` procedure drops all compartments from a user's authorizations.
- SA_USER_ADMIN.DROP_ALL_GROUPS
  The `SA_USER_ADMIN.DROP_ALL_GROUPS` procedure drops all groups from a user's authorizations.
- SA_USER_ADMIN.DROP_COMPARTMENTS
  The `SA_USER_ADMIN.DROP_COMPARTMENTS` procedure drops the specified compartments from a user's authorizations.
- SA_USER_ADMIN.DROP_GROUPS
  The `SA_USER_ADMIN.DROP_GROUPS` procedure drops the specified groups from a user's authorizations.
- SA_USER_ADMIN.DROP_USER_ACCESS
  The `SA_USER_ADMIN.DROP_USER_ACCESS` procedure removes all Oracle Label Security authorizations and privileges from the specified user.
- SA_USER_ADMIN.SET_COMPARTMENTS
  The `SA_USER_ADMIN.SET_COMPARTMENTS` procedure assigns compartments to a user and identifies default values for the user's session label and row label.
- SA_USER_ADMIN.SET_DEFAULT_LABEL
  The `SA_USER_ADMIN.SET_DEFAULT_LABEL` procedure sets the user's initial session label to the one specified.
- SA_USER_ADMIN.SET_GROUPS
  The `SA_USER_ADMIN.SET_GROUPS` procedure assigns groups to a user and identifies default values for the user's session label and row label.
- SA_USER_ADMIN.SET_LEVELS
  The `SA_USER_ADMIN.SET_LEVELS` procedure assigns a user minimum and maximum levels and identifies default values for the user's session label and row label.
- SA_USER_ADMIN.SET_PROG_PRIVS
  The `SA_USER_ADMIN.SET_PROG_PRIVS` procedure sets policy-specific privileges for program units.
- SA_USER_ADMIN.SET_ROW_LABEL
  The `SA_USER_ADMIN.SET_ROW_LABEL` procedure sets a user's initial row label to the one specified.

- SA_USER_ADMIN.SET_USER_LABELS
  The SA_USER_ADMIN.SET_USER_LABELS procedure sets the user's levels, compartments, and groups using a set of labels, instead of the individual components.
- SA_USER_ADMIN.SET_USER_PRIVS
  The SA_USER_ADMIN.SET_USER_PRIVS procedure sets policy-specific privileges for users.

D.7.1 About the SA_USER_ADMIN PL/SQL Package

The SA_USER_ADMIN PL/SQL package configures compartments, groups, user access, labels, levels, and privileges.

To use this package, you must be granted the *policy*_DBA role (for example, HR_OLS_POL_DBA for a role for the hr_ols_pol policy) and the EXECUTE privilege on the SA_USER_ADMIN package.

D.7.2 SA_USER_ADMIN.ADD_COMPARTMENTS

The SA_USER_ADMIN.ADD_COMPARTMENTS procedure adds (assigns) compartments to a user's authorizations, indicating if the compartments are authorized for write and read privileges.

This procedure is useful if you have already used the SA_USER_ADMIN.SET_COMPARTMENTS procedure for the user but then decide that you want to grant this user authorization for additional compartments, or to update the current set of compartments. You also can use it in place of SA_USER_ADMIN.SET_COMPARTMENTS.

Syntax

```
SA_USER_ADMIN.ADD_COMPARTMENTS (
policy_name     IN VARCHAR2,
user_name       IN VARCHAR2,
comps           IN VARCHAR2,
access_mode     IN VARCHAR2 DEFAULT NULL,
in_def          IN VARCHAR2 DEFAULT NULL,
in_row          IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table D-51 SA_USER_ADMIN.ADD_COMPARTMENTS Parameters

Parameter	Description
policy_name	Specifies the policy. To find existing policies, query the POLICY_NAME column of the ALL_SA_POLICIES data dictionary view.
user_name	Specifies the user name. This user can be either a new user or a user who has already been authorized for this policy's compartments. To find an existing user, query the USER_NAME column of the DBA_SA_USER_COMPARTMENTS view.
comps	A comma-delimited list of compartments to add, by short name only. To find existing compartments, query the SHORT_NAME column of the ALL_SA_COMPARTMENTS view.

Table D-51 (Cont.) SA_USER_ADMIN.ADD_COMPARTMENTS Parameters

Parameter	Description
`access_mode`	One of two public variables that contain string values that can specify the type of access authorized. The variable names, values, and meaning are as follows: • `SA_UTL.READ_ONLY` indicates no write access • `SA_UTL.READ_WRITE` indicates that write is authorized • If `access_mode` is `NULL`, then it is set to `SA_UTL.READ_ONLY`.
`in_def`	Specifies whether these compartments should be in the default compartments (`Y`/`N`) If `in_def` is `NULL`, then it is set to `Y`.
`in_row`	Specifies whether these compartments should be in the row label (`Y`/`N`) If `in_row` is `NULL`, then it is set to `N`.

Example

The following example adds compartments to the `hr_ols_pol` policy.

```
BEGIN
 SA_USER_ADMIN.ADD_COMPARTMENTS (
  policy_name     => 'hr_ols_pol',
  user_name       => 'jjones',
  comps           => 'FIN',
  access_mode     => SA_UTL.READ_ONLY,
  in_def          => 'y',
  in_row          => 'y');
END;
/
```

D.7.3 SA_USER_ADMIN.ADD_GROUPS

The `SA_USER_ADMIN.ADD_GROUPS` procedure adds (assigns) groups to a user, indicating if the groups are authorized for write and read privileges.

This procedure is useful if you have already used the `SA_USER_ADMIN.SET_GROUPS` procedure for the user but then decide that you want to grant this user authorization for additional groups or to update the current set of groups. You also can use it in place of `SA_USER_ADMIN.SET_GROUPS`.

Syntax

```
SA_USER_ADMIN.ADD_GROUPS (
  policy_name        IN VARCHAR2,
  user_name          IN VARCHAR2,
  groups             IN VARCHAR2,
  access_mode        IN VARCHAR2 DEFAULT NULL,
  in_def             IN VARCHAR2 DEFAULT NULL,
  in_row             IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table D-52 SA_USER_ADMIN.ADD_GROUPS Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`user_name`	Specifies the user. This user can be either a new user or a user who has already been authorized for this policy's groups. To find an existing user, query the `USER_NAME` column of the `DBA_SA_USER_GROUPS` view.
`groups`	A comma-delimited list of groups to add, by short name only. To find a list of existing groups, query the `SHORT_NAME` column of the `ALL_SA_GROUPS` view.
`access_mode`	One of two public variables that contain string values that can specify the type of access authorized. The variable names, values, and meaning are as follows: • `SA_UTL.READ_ONLY` indicates no write access • `SA_UTL.READ_WRITE` indicates that write is authorized • If `access_mode` is `NULL`, then `access_mode` is set to `SA_UTL.READ_ONLY`.
`in_def`	Specifies whether these groups should be in the default groups (`Y`/`N`) If `in_def` is `NULL`, then it is set to `Y`.
`in_row`	Specifies whether these groups should be in the row label (`Y`/`N`) If `in_row` is `NULL`, then it is set to `N`.

Example

The following example adds several groups to the `hr_ols_pol` policy.

```
BEGIN
 SA_USER_ADMIN.ADD_GROUPS (
  policy_name     => 'hr_ols_pol',
  user_name       => 'jjones',
  groups          => 'ER_FIN, SR_FIN, NR_FIN, WR_FIN',
  access_mode     => SA_UTL.READ_WRITE,
  in_def          => 'y',
  in_row          => 'y');
END;
/
```

D.7.4 SA_USER_ADMIN.ALTER_COMPARTMENTS

The `SA_USER_ADMIN.ALTER_COMPARTMENTS` procedure changes the write access, default label indicator, and row label indicator for the specified compartments.

Syntax

```
SA_USER_ADMIN.ALTER_COMPARTMENTS (
  policy_name  IN VARCHAR2,
  user_name     IN VARCHAR2,
  comps         IN VARCHAR2,
  access_mode   IN VARCHAR2 DEFAULT NULL,
  in_def        IN VARCHAR2 DEFAULT NULL,
  in_row        IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table D-53 SA_USER_ADMIN.ALTER_COMPARTMENTS Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`user_name`	Specifies the user who has been authorized for the compartment. To find authorized users, query the `USER_NAME` column of the `DBA_SA_USER_COMPARTMENTS` view.
`comps`	A comma-delimited list of compartments to modify, using the short name only. To find existing compartments, query the `SHORT_NAME` column of the `ALL_SA_COMPARTMENTS` view.
`access_mode`	One of two public variables that contain string values that can specify the type of access authorized. The variable names, values, and meaning are as follows: `SA_UTL.READ_ONLY` indicates no write access `SA_UTL.READ_WRITE` indicates that write is authorized If `access_mode` is `NULL`, then `access_mode` for the compartment is unaltered.
`in_def`	Specifies whether these compartments should be in the default compartments (`Y`/`N`) If `in_def` is `NULL`, then `in_def` for the compartment is unaltered.
`in_row`	Specifies whether these compartments should be in the row label (`Y`/`N`) If `in_row` is `NULL`, then `in_row` for the compartment is unaltered. If `in_def` is `N`, then `in_row` cannot be `Y`. This is because the row label compartments must be a subset of the session label compartments.

Example

The following example modifies compartments for the `hr_ols_pol` policy.

```
BEGIN
 SA_USER_ADMIN.ALTER_COMPARTMENTS (
  policy_name     => 'hr_ols_pol',
  user_name       => 'jjones',
  comps           => 'FIN',
  access_mode     => SA_UTL.READ_ONLY,
  in_def          => 'y',
  in_row          => 'y');
END;
/
```

D.7.5 SA_USER_ADMIN.ALTER_GROUPS

The `SA_USER_ADMIN.ALTER_GROUPS` procedure changes the write access, default label indicator, and row label indicator for the specified groups.

Syntax

```
SA_USER_ADMIN.ALTER_GROUPS (
  policy_name      IN VARCHAR2,
  user_name        IN VARCHAR2,
  groups           IN VARCHAR2,
  access_mode      IN VARCHAR2 DEFAULT NULL,
  in_def           IN VARCHAR2 DEFAULT NULL,
  in_row           IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table D-54 SA_USER_ADMIN.ALTER_GROUPS Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`user_name`	Specifies the user who has been authorized for the group. To find existing users, query the `USER_NAME` and `GRP` columns of the `DBA_SA_USER_GROUPS` view.
`groups`	A comma-delimited list of groups to alter, by short name only. To find existing groups, query the `SHORT_NAME` column of the `ALL_SA_GROUPS` view.
`access_mode`	Two public variables contain string values that can specify the type of access authorized. The variable names, values, and meaning are as follows: `SA_UTL.READ_ONLY` indicates no write access `SA_UTL.READ_WRITE` indicates that write is authorized If `access_mode` is `NULL`, then `access_mode` for the group is unaltered.
`in_def`	Specifies whether these groups should be in the default groups (`Y`/`N`) If `in_def` is `NULL`, then `in_def` for the group is unaltered.
`in_row`	Specifies whether these groups should be in the row label ((`Y`/`N`) If `in_row` is `NULL`, then `in_row` for the group is unaltered. If `in_def` is `N`, then `in_row` cannot be `Y`. This is because the row label groups must be a subset of the session label groups.

Example

The following example sets the access mode for the existing groups to be read only.

```
BEGIN
 SA_USER_ADMIN.ALTER_GROUPS (
  policy_name     => 'hr_ols_pol',
  user_name       => 'jjones',
  groups          => 'ER',
  access_mode     => SA_UTL.READ_ONLY);
```

```
END;
/
```

D.7.6 SA_USER_ADMIN.DROP_ALL_COMPARTMENTS

The `SA_USER_ADMIN.DROP_ALL_COMPARTMENTS` procedure drops all compartments from a user's authorizations.

Syntax

```
SA_USER_ADMIN.DROP_ALL_COMPARTMENTS (
 policy_name  IN VARCHAR2,
 user_name    IN VARCHAR2);
```

Parameters

Table D-55 SA_USER_ADMIN.DROP_ALL_COMPARTMENTS Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`user_name`	Specifies the user who has been authorized for the compartment. To find existing users, query the `USER_NAME` column of the `DBA_SA_USER_COMPARTMENTS` view.

Example

The following example drops all compartments for the `hr_ols_pol` policy for user `jjones`.

```
BEGIN
 SA_USER_ADMIN.DROP_ALL_COMPARTMENTS (
  policy_name    => 'hr_ols_pol',
  user_name      => 'jjones');
END;
/
```

D.7.7 SA_USER_ADMIN.DROP_ALL_GROUPS

The `SA_USER_ADMIN.DROP_ALL_GROUPS` procedure drops all groups from a user's authorizations.

Syntax

```
SA_USER_ADMIN.DROP_ALL_GROUPS (
  policy_name IN VARCHAR2,
  user_name   IN VARCHAR2);
```

Parameters

Table D-56 SA_USER_ADMIN.DROP_ALL_GROUPS Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`user_name`	Specifies the user who has been authorized for the group. To find existing users, query the `USER_NAME` and `GRP` columns of the `DBA_SA_USER_GROUPS` view.

Example

The following example drops all groups from the `hr_ols_pol` policy for user `jjones`.

```
BEGIN
 SA_USER_ADMIN.DROP_ALL_GROUPS (
  policy_name     => 'hr_ols_pol',
  user_name       => 'jjones');
END;
/
```

D.7.8 SA_USER_ADMIN.DROP_COMPARTMENTS

The `SA_USER_ADMIN.DROP_COMPARTMENTS` procedure drops the specified compartments from a user's authorizations.

Syntax

```
SA_USER_ADMIN.DROP_COMPARTMENTS (
  policy_name      IN VARCHAR2,
  user_name        IN VARCHAR2,
  comps            IN VARCHAR2);
```

Parameters

Table D-57 SA_USER_ADMIN.DROP_COMPARTMENTS Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`user_name`	Specifies the user who has been authorized for the compartment. To find existing users, query the `USER_NAME` column of the `DBA_SA_USER_COMPARTMENTS` view.
`comps`	A comma-delimited list of compartments to drop. To find all comps for this policy, query the `POLICY_NAME` and `COMP` columns of the `DBA_SA_USER_COMPARTMENTS` view.

Example

The following example drops the `FINANCIAL` compartment from the `hr_ols_pol` policy.

```
BEGIN
 SA_USER_ADMIN.DROP_COMPARTMENTS (
  policy_name     => 'hr_ols_pol',
  user_name       => 'jjones',
  comps           => 'HR');
END;
/
```

D.7.9 SA_USER_ADMIN.DROP_GROUPS

The SA_USER_ADMIN.DROP_GROUPS procedure drops the specified groups from a user's authorizations.

Syntax

```
SA_USER_ADMIN.DROP_GROUPS (
  policy_name IN VARCHAR2,
  user_name   IN VARCHAR2,
  groups      IN VARCHAR2);
```

Parameters

Table D-58 SA_USER_ADMIN.DROP_GROUPS Parameters

Parameter	Description
policy_name	Specifies the policy. To find existing policies, query the POLICY_NAME column of the ALL_SA_POLICIES data dictionary view.
user_name	Specifies the user who has been authorized for the group. To find existing users, query the USER_NAME and GRP columns of the DBA_SA_USER_GROUPS view.
groups	A comma-delimited list of groups to drop, by short name only. To find a list of groups, query the SHORT_NAME column of the ALL_SA_GROUPS view.

Example

The following example drops the NR_FIN group from the hr_ols_pol policy.

```
BEGIN
 SA_USER_ADMIN.DROP_GROUPS (
  policy_name     => 'hr_ols_pol',
  user_name       => 'jjones',
  groups          => 'ER');
END;
/
```

D.7.10 SA_USER_ADMIN.DROP_USER_ACCESS

The `SA_USER_ADMIN.DROP_USER_ACCESS` procedure removes all Oracle Label Security authorizations and privileges from the specified user.

Syntax

```
SA_USER_ADMIN.DROP_USER_ACCESS (
  policy_name       IN VARCHAR2,
  user_name         IN VARCHAR2);
```

Parameters

Table D-59 SA_USER_ADMIN.DROP_USER_ACCESS Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`user_name`	Specifies the user name. To find all users associated with this policy, query the `USER_NAME` and `POLICY_NAME` columns of the `DBA_SA_USER_PRIVS` view.

Examples

The following example removes user `jjones`'s authorization for the `hr_ols_pol` policy.

```
BEGIN
 SA_USER_ADMIN.DROP_USER_ACCESS (
  policy_name        => 'hr_ols_pol',
  user_name          => 'jjones');
END;
/
```

D.7.11 SA_USER_ADMIN.SET_COMPARTMENTS

The `SA_USER_ADMIN.SET_COMPARTMENTS` procedure assigns compartments to a user and identifies default values for the user's session label and row label.

After you have set the compartment, you can configure additional compartments by using the `SA_USER_ADMIN.ADD_COMPARTMENTS` procedure. (See SA_USER_ADMIN.ADD_COMPARTMENTS.)

All users must have their levels set before their authorized compartments can be established.

The write compartments, if specified, must be a subset of the read compartments. (The write compartments are those to which the user should have write access.)

Syntax

```
SA_USER_ADMIN.SET_COMPARTMENTS (
  policy_name    IN VARCHAR2,
  user_name      IN VARCHAR2,
  read_comps     IN VARCHAR2,
  write_comps    IN VARCHAR2 DEFAULT NULL,
  def_comps      IN VARCHAR2 DEFAULT NULL,
  row_comps      IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table D-60 SA_USER_ADMIN.SET_COMPARTMENTS Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`user_name`	Specifies the user name to assign compartments
`read_comps`	A comma-delimited list of compartments authorized for read access, by short name only To find all compartments, query the `SHORT_NAME` column of the `ALL_SA_COMPARTMENTS` view.
`write_comps`	A comma-delimited list of compartments authorized for write access (subset of `read_comps`), by short name only. If `write_comps` are `NULL`, then they are set to the `read_comps`.
`def_comps`	Specifies the default compartments, by short name only. This must be a subset of `read_comps`. If the `def_comps` are `NULL`, then they are set to the `read_comps`.
`row_comps`	Specifies the row compartments, by short name only. This must be a subset of `write_comps` and `def_comps`. If the `row_comps` are `NULL`, then they are set to the components in `def_comps` that are authorized for write access.

Example

The following example sets compartments for the `hr_ols_pol` policy.

```
BEGIN
 SA_USER_ADMIN.SET_COMPARTMENTS (
  policy_name    => 'hr_ols_pol',
  user_name      => 'jjones',
  read_comps     => 'FIN',
  write_comps    => 'FIN',
  def_comps      => 'FIN',
  row_comps      => 'FIN');
END;
/
```

D.7.12 SA_USER_ADMIN.SET_DEFAULT_LABEL

The `SA_USER_ADMIN.SET_DEFAULT_LABEL` procedure sets the user's initial session label to the one specified.

As long as the row label will still be dominated by the new write label, you can set the session label to:

- Any level equal to or less than their maximum, and equal to or greater than their minimum label
- Include any compartments in the authorized compartment list

- Include any groups in the authorized group list. (Subgroups of authorized groups are implicitly included in the authorized list.)

The row label must be dominated by the new write label that will result from resetting the session label. If this condition is not true, then the `SET_DEFAULT_LABEL` procedure will fail.

For example, suppose the current row label is `S:A,B`, and that you have write access to both compartments. If you attempt to set the new default label to `C:A,B`, then the `SET_LABEL` procedure will fail. This is because the new write label would be `C:A,B`, which does not dominate the current row label.

To successfully reset the session label in this case, you must first lower the row label to a value that will be dominated by the resulting session label.

Syntax

```
SA_USER_ADMIN.SET_DEFAULT_LABELS (
  policy_name  IN VARCHAR2,
  user_name    IN VARCHAR2,
  def_label    IN VARCHAR2);
```

Parameters

Table D-61 SA_USER_ADMIN.SET_DEFAULT_LABEL Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`user_name`	Specifies the user who has been authorized with label components. To find this user, query the `USER_NAME` column of the `ALL_SA_USER_LABELS` view.
`def_label`	Specifies the label string to be used to initialize the user's default labels. This label may contain any compartments and groups that are authorized for read access. To find existing labels, query the `LABEL` column of the `ALL_SA_LABELS` view.

Example

The following example sets the default label for `hr_ols_pol` for user `jjones`.

```
BEGIN
 SA_USER_ADMIN.SET_DEFAULT_LABEL (
  policy_name       => 'hr_ols_pol',
  user_name         => 'jjones',
  def_label         => 'HS');
END;
/
```

Related Topics

- SA_SESSION Session Management PL/SQL Package
  The `SA_SESSION` PL/SQL package manages session behavior for user authorizations.

D.7.13 SA_USER_ADMIN.SET_GROUPS

The `SA_USER_ADMIN.SET_GROUPS` procedure assigns groups to a user and identifies default values for the user's session label and row label.

All users must have their levels set before their authorized groups can be established. You can find information about a user's level authorization by querying the `DBA_SA_USER_LEVELS` data dictionary view.

Syntax

```
SA_USER_ADMIN.SET_GROUPS (policy_name IN VARCHAR2,
  user_name        IN VARCHAR2,
  read_groups      IN VARCHAR2,
  write_groups     IN VARCHAR2 DEFAULT NULL,
  def_group        IN VARCHAR2 DEFAULT NULL,
  row_groups       IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table D-62   SA_USER_ADMIN.SET_GROUPS Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`user_name`	Specifies the user name. This user is a first-time user for group authorization, but the user must already be authorized for levels. To find users who have been authorized for levels, query the `USER_NAME` column of the `DBA_SA_USER_LEVELS` view.
`read_groups`	A comma-delimited list of groups authorized for read, by short name only. To find existing groups, query the `SHORT_NAME` column of the `ALL_SA_GROUPS` view.
`write_groups`	A comma-delimited list of groups authorized for write, by short name only. This must be a subset of `read_groups`. If set to `NULL`, then this setting defaults to `read_groups`.
`def_groups`	Specifies the default groups, by short name only. This must be a subset of `read_groups`. If set to `NULL`, then this setting defaults to `read_groups`.
`row_groups`	Specifies the row groups, by short name only. This must be a subset of `write_groups` and `def_groups`. If set to `NULL`, then this setting defaults to the groups in `def_groups` that are authorized for write access.

Example

The following example defines groups for the `hr_ols_pol` policy.

```
BEGIN
 SA_USER_ADMIN.SET_GROUPS (
  policy_name    => 'hr_ols_pol',
  user_name      => 'jjones',
  read_groups    => 'ER_FIN',
  write_groups   => 'ER_FIN',
  def_groups     => 'ER_FIN',
  row_groups     => 'ER_FIN');
END;
/
```

D.7.14 SA_USER_ADMIN.SET_LEVELS

The `SA_USER_ADMIN.SET_LEVELS` procedure assigns a user minimum and maximum levels and identifies default values for the user's session label and row label.

Syntax

```
SA_USER_ADMIN.SET_LEVELS (policy_name IN VARCHAR2,
   user_name         IN VARCHAR2,
   max_level         IN VARCHAR2,
   min_level         IN VARCHAR2 DEFAULT NULL,
   def_level         IN VARCHAR2 DEFAULT NULL,
   row_level         IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table D-63 SA_USER_ADMIN.SET_LEVELS Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`user_name`	Specifies the user name. This user does not need to have any Oracle Label Security authorizations before you run this procedure.
`max_level`	The highest level for read and write access, by short name only. To find existing levels, query the `SHORT_NAME` column of the `ALL_SA_LEVELS` view.
`min_level`	The lowest level for write access, by short name only. If set to `NULL`, then the default is the lowest level for the policy.
`def_level`	Specifies the default level (equal to or greater than the minimum level, and equal to or less than the maximum level). Use the short name only. If set to `NULL`, then the default is the `max_level`.
`row_level`	Specifies the row level (equal to or greater than the minimum level, and equal to or less than the default level). Use the short name only. If set to `NULL`, then it is set to the `def_level`.

Example

The following example sets levels for the `hr_ols_pol` policy.

```
BEGIN
  SA_USER_ADMIN.SET_LEVELS (
   policy_name     => 'hr_ols_pol',
   user_name       => 'jjones',
   max_level       => 'PUB',
   min_level       => 'HS');
END;
/
```

D.7.15 SA_USER_ADMIN.SET_PROG_PRIVS

The SA_USER_ADMIN.SET_PROG_PRIVS procedure sets policy-specific privileges for program units.

If the privileges parameter is NULL, then the program unit's privileges for the policy are removed.

To grant privileges to a stored program unit, you must have the *policy*_DBA role, and the EXECUTE permission on the SA_USER_ADMIN.SA_USER_ADMIN package. You can use either the SA_USER_ADMIN package or Oracle Enterprise Manager to manage Oracle Label Security privileges.

Syntax

```
SA_USER_ADMIN.SET_PROG_PRIVS (
  policy_name            IN VARCHAR2,
  schema_name            IN VARCHAR2,
  program_unit_name      IN VARCHAR2,
  privileges             IN VARCHAR2);
```

Parameters

Table D-64 SA_USER_ADMIN.SET_PROG_PRIVS Parameters

Parameter	Description
policy_name	Specifies the policy. To find existing policies, query the POLICY_NAME column of the ALL_SA_POLICIES data dictionary view.
schema_name	The name of the schema that contains the program unit
program_unit_name	Specifies the program unit to be granted privileges
privileges	A comma-delimited character string of policy-specific privileges. If you set privileges to NULL, then the program unit's privileges for the policy are removed. See About Granting Privileges to Users and Trusted Program Units for the Policy for list of available privileges to grant.

Example

The following example gives the READ privilege to the SUM_PURCHASES function (described in Example: Trusted Stored Program Unit):

```
BEGIN
 SA_USER_ADMIN.SET_PROG_PRIVS (
  policy_name          => 'hr_ols_pol',
  schema_name          => 'HR',
  program_unit_name    => 'sum_purchases',
  privileges           => 'READ');
END;
/
```

When the check_emp_hours procedure is then called, it runs with the READ privilege as well as the current user's Oracle Label Security privileges. Using this technique, the user can be allowed to find the value of the total employee hours that were logged, without learning what hours any individual employee logged.

D.7.16 SA_USER_ADMIN.SET_ROW_LABEL

The `SA_USER_ADMIN.SET_ROW_LABEL` procedure sets a user's initial row label to the one specified.

The user can set the row label independently, but only to:

- A level that is less than or equal to the level of the session label, and greater than or equal to the user's minimum level
- Include a subset of the compartments and groups from the session label, for which the user is authorized to have write access

If you try to set the row label to an invalid value, then the operation is disallowed, and the row label value is unchanged.

Syntax

```
SA_USER_ADMIN.SET_ROW_LABEL (
  policy_name    IN VARCHAR2,
  user_name      IN VARCHAR2,
  row_label      IN VARCHAR2);
```

Parameters

Table D-65 SA_USER_ADMIN.SET_ROW_LABEL Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`user_name`	Specifies the user name. This user must have the sufficient compartment, group, and level authorizations. To find this user, query the `USER_NAME` column of the `DBA_SA_USER_COMPARTMENTS`, `DBA_SA_USER_GROUPS`, and `DBA_SA_USER_LEVELS` views.
`row_label`	Specifies the label string to be used to initialize the user's row label. The label must contain only those compartments and groups from the default label that are authorized for write access. To find existing compartments and groups, query the `ALL_SA_COMPARTMENTS` and `ALL_SA_GROUPS` views.

Example

The following example sets the row label for the `hr_ols_pol` policy for user `jjones`.

```
BEGIN
 SA_USER_ADMIN.SET_ROW_LABEL (
  policy_name        => 'hr_ols_pol',
  user_name          => 'jjones',
  row_label          => 'HS');
END;
/
```

Related Topics

- SA_SESSION.SET_ROW_LABEL
  The `SA_SESSION.SET_ROW_LABEL` procedure sets the default row label value for the current database session.

D.7.17 SA_USER_ADMIN.SET_USER_LABELS

The `SA_USER_ADMIN.SET_USER_LABELS` procedure sets the user's levels, compartments, and groups using a set of labels, instead of the individual components.

Syntax

```
SA_USER_ADMIN.SET_USER_LABELS (
  policy_name      IN VARCHAR2,
  user_name        IN VARCHAR2,
  max_read_label   IN VARCHAR2,
  max_write_label  IN VARCHAR2 DEFAULT NULL,
  min_write_label  IN VARCHAR2 DEFAULT NULL,
  def_label        IN VARCHAR2 DEFAULT NULL,
  row_label        IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table D-66 SA_USER_ADMIN.SET_USER_LABELS Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`user_name`	Specifies the user name. The user can be an existing database user or an Oracle Real Application Security user. This user does not need any Oracle Label Security authorizations before you run this procedure.
`max_read_label`	Specifies the label string to be used to initialize the user's maximum authorized read label. Composed of the user's maximum level, compartments authorized for read access, and groups authorized for read access. To find information for these settings, query the `DBA_SA_USERS` data dictionary view.
`max_write_label`	Specifies the label string to be used to initialize the user's maximum authorized write label. Composed of the user's maximum level, compartments authorized for write access, and groups authorized for write access. If `max_write_label` is not specified, then it is set to `max_read_label`.
`min_write_label`	Specifies the label string to be used to initialize the user's minimum authorized write label. Contains only the level, with no compartments or groups. If `min_write_label` is not specified, then it is set to the lowest defined level for the policy, with no compartments or groups.
`def_label`	Specifies the label string to be used to initialize the user's session label, including level, compartments, and groups (a subset of `max_read_label`). If `default_label` is not specified, then it is set to `max_read_label`.

Table D-66 (Cont.) SA_USER_ADMIN.SET_USER_LABELS Parameters

Parameter	Description
`row_label`	Specifies the label string to be used to initialize the program's row label. Includes level, components, and groups: subsets of `max_write_label` and `def_label`. If `row_label` is not specified, then it is set to `def_label`, with only the compartments and groups authorized for write access.

Examples

The following example sets user labels for the `hr_ols_pol` policy for user `jjones`.

```
BEGIN
 SA_USER_ADMIN.SET_USER_LABELS (
  policy_name       => 'hr_ols_pol',
  user_name         => 'jjones',
  max_read_label    => 'HS:FIN',
  max_write_label   => 'HS',
  def_label         => 'HS',
  row_label         => 'HS');
END;
/
```

The following example sets user labels for the `XSOLSPOL1` policy for the Oracle Database Real Application Security user `XSUSER1`. To run the following example, you must either an Oracle Label Security administrator, be granted the `LBAC_DBA` database role and granted the `EXECUTE` privilege, or be granted the `XSOLSPOL1_DBA` role and granted the `EXECUTE` privilege on the `SA_USER_ADMIN` package.

```
EXEC SA_USER_ADMIN.SET_USER_LABELS('XSOLSPOL1', 'XSUSER1','MID','MID');
```

In this specification:

- `XSOLSPOL1` is the name of an existing OLS policy.
- `XSUSER1` is the name of an existing Oracle Database Real Application Security user.
- `MID` is the value of the `max_read_label`.
- `MID` is the value of the `max_write_label`.

Related Topics

- SA_USER_ADMIN.SET_PROG_PRIVS
  The `SA_USER_ADMIN.SET_PROG_PRIVS` procedure sets policy-specific privileges for program units.

D.7.18 SA_USER_ADMIN.SET_USER_PRIVS

The `SA_USER_ADMIN.SET_USER_PRIVS` procedure sets policy-specific privileges for users.

These privileges do not become effective until the next time the user logs into the database. The new set of privileges replaces any existing privileges. A `NULL` value for the privileges parameter removes the user's privileges for the policy.

To assign policy privileges to users, you must have the `EXECUTE` privilege for the `SA_USER_ADMIN` package, and must have been granted the *`policy`*`_DBA` role.

Syntax

```
SA_USER_ADMIN.SET_USER_PRIVS (
  policy_name      IN VARCHAR2,
  user_name        IN VARCHAR2,
  privileges       IN VARCHAR2);
```

Parameters

Table D-67 SA_USER_ADMIN.SET_USER_PRIVS Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`user_name`	The name of the user to be granted privileges. The user can be an existing database user or an Oracle Real Application Security user. This user should already have been authorized for policy levels, compartments, and groups. To find this user, query the `USER_NAME` column of the `DBA_SA_USER_COMPARTMENTS`, `DBA_SA_USER_GROUPS`, and `DBA_SA_USER_LABELS` views.
`privileges`	A character string of policy-specific privileges separated by commas. See About Granting Privileges to Users and Trusted Program Units for the Policy for list of available privileges to grant.

Examples

The following example grants user `jgodfrey` full privileges for the `hr_ols_pol` policy settings.

```
BEGIN
 SA_USER_ADMIN.SET_USER_PRIVS (
  policy_name       => 'hr_ols_pol',
  user_name         => 'jgodfrey',
  privileges        => 'FULL');
END;
/
```

The following example grants Oracle Database Real Application Security user `XSUSER1` the `READ` privilege for the Oracle Label Security policy `XSOLSPOL1`. To run the following example, you must either be an Oracle Label Security administrator, be granted the `LBAC_DBA` database role and granted the `EXECUTE` privilege, or be granted the `XSOLSPOL1_DBA` role and granted the `EXECUTE` privilege on the `SA_USER_ADMIN` package.

```
EXEC SA_USER_ADMIN.SET_USER_PRIVS('XSOLSPOL1', 'XSUSER1','READ');
```

In this specification:

- `XSOLSPOL1` is the name of an existing OLS policy.
- `XSUSER1` is the name of an existing Oracle Database Real Application Security user.

- READ is the privilege to be granted to XSUSER1 in OLS policy XSOLSPOL1.

Related Topics

- About Granting Privileges to Users and Trusted Program Units for the Policy
  After you have authorized users for policy levels, compartments, and groups, you are ready to grant the user privileges.

D.8 SA_UTL PL/SQL Utility Functions and Procedures

The SA_UTL PL/SQL package contains utility functions and procedures that are used in PL/SQL programs.

- About the SA_UTL PL/SQL Package
  The SA_UTL PL/SQL package utility functions include returning the values such as user privileges or label information.
- SA_UTL.CHECK_LABEL_CHANGE
  The SA_UTL.CHECK_LABEL_CHANGE function checks if the user can change the data label for a policy protected table row.
- SA_UTL.CHECK_READ
  The SA_UTL.CHECK_READ function checks if a user can read a policy-protected table row.
- SA_UTL.CHECK_WRITE
  The SA_UTL.CHECK_WRITE function to checks if the user can insert, update, or delete data in a policy protected table row.
- SA_UTL.DATA_LABEL
  The SA_UTL.DATA_LABEL function returns TRUE if the label is a data label.
- SA_UTL.GREATEST_LBOUND
  The SA_UTL.GREATEST_LBOUND function returns a label that is the greatest lower bound of the two label arguments.
- SA_UTL.NUMERIC_LABEL
  The SA_UTL.NUMERIC_LABEL function returns the current session label.
- SA_UTL.NUMERIC_ROW_LABEL
  The SA_UTL.NUMERIC_ROW_LABEL function returns the current row label. .
- SA_UTL.SET_LABEL
  The SA_UTL.SET_LABEL procedure sets the label of the current database session.
- SA_UTL.SET_ROW_LABEL
  The SA_UTL.SET_ROW_LABEL procedure sets the row label of the current database session.

D.8.1 About the SA_UTL PL/SQL Package

The SA_UTL PL/SQL package utility functions include returning the values such as user privileges or label information.

These programs return information about the current values of the session security attributes, as numeric label values. They are primarily for use in trusted stored program units. You do not need special privileges to use this package.

Related Topics

- How Setting and Returning Label Information Works
  The SA_UTL package has functions to return information about current values of session security attributes using numeric label values.

D.8.2 SA_UTL.CHECK_LABEL_CHANGE

The SA_UTL.CHECK_LABEL_CHANGE function checks if the user can change the data label for a policy protected table row.

This function returns 1 if the user can change the data label. It returns 0 if the user cannot change the data label. The input values are the policy name, the current data label, and the new data label.

Syntax

```
SA_UTL.CHECK_LABEL_CHANGE (
  policy_name      IN VARCHAR2,
  current_label    IN NUMBER,
  new_label        IN NUMBER)
RETURN NUMBER;
```

Note:

You must have update privileges on the table to write any data into the table.

Parameters

Table D-68 SA_UTL.CHECK_LABEL_CHANGE Parameters

Parameter	Description
policy_name	Specifies the policy. To find existing policies, query the POLICY_NAME column of the ALL_SA_POLICIES data dictionary view.
current_label	The current value of the label. To find existing label values, query the LABEL column of the ALL_SA_LABELS view.
new_label	The new value for the label

Example

The following example indicates if users can change data labels in policy-protected rows.

```
SET SERVEROUTPUT ON
BEGIN
  IF SA_UTL.CHECK_LABEL_CHANGE('hr_ols_pol',2000, 2200) = 1
   THEN DBMS_OUTPUT.PUT_LINE('Users can chagne data labels in policy-protected
rows.');
  ELSE
   DBMS_OUTPUT.PUT_LINE('Users cannot change data labels in policy-protected
rows.');
  END IF;
```

```
END;
/
```

D.8.3 SA_UTL.CHECK_READ

The `SA_UTL.CHECK_READ` function checks if a user can read a policy-protected table row.

This function returns `1` if the user can read the table row. It returns `0` if the user cannot read the table row.

Note:

The user must have the `SELECT` privilege on the table to read any data from the table.

Syntax

```
SA_UTL.CHECK_READ (
  policy_name      IN VARCHAR2,
  label            IN NUMBER)
RETURN NUMBER;
```

Parameters

Table D-69 SA_UTL.CHECK_READ Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`label`	The label to be checked. To find existing label values, query the `LABEL` column of the `ALL_SA_LABELS` view.

Example

The following example indicates if users can read a policy-protected row.

```
SET SERVEROUTPUT ON
BEGIN
  IF SA_UTL.CHECK_READ('hr_ols_pol',2000) = 1
   THEN DBMS_OUTPUT.PUT_LINE('Users can read policy-protected rows.');
  ELSE
   DBMS_OUTPUT.PUT_LINE('Users cannot read policy-protected rows.');
  END IF;
END;
/
```

D.8.4 SA_UTL.CHECK_WRITE

The `SA_UTL.CHECK_WRITE` function to checks if the user can insert, update, or delete data in a policy protected table row.

The user should already have the UPDATE privilege on the table to write any data into the table. This function returns 1 if the user can write to the table row. It returns 0 if the user cannot write to the table row. The input values are the policy name and the row data label.

Syntax

```
SA_UTL.CHECK_WRITE (
  policy_name     IN VARCHAR2,
  label           IN NUMBER)
RETURN NUMBER;
```

Parameters

Table D-70 SA_UTL.CHECK_WRITE Parameters

Parameter	Description
policy_name	Specifies the policy. To find existing policies, query the POLICY_NAME column of the ALL_SA_POLICIES data dictionary view.
label	The label to be checked. To find existing label values, query the LABEL and TAG columns of the ALL_SA_LABELS view.

Example

The following example indicates if users can write to policy-protected rows.

```
SET SERVEROUTPUT ON
BEGIN
  IF SA_UTL.CHECK_WRITE('hr_ols_pol',2000) = 1
   THEN DBMS_OUTPUT.PUT_LINE('Users can write to policy-protected rows.');
  ELSE
   DBMS_OUTPUT.PUT_LINE('Users cannot write to policy-protected rows.');
  END IF;
END;
/
```

D.8.5 SA_UTL.DATA_LABEL

The SA_UTL.DATA_LABEL function returns TRUE if the label is a data label.

Syntax

```
SA_UTL.DATA_LABEL(
 label IN NUMBER)
RETURN BOOLEAN;
```

Parameters

Table D-71 SA_UTL.DATA_LABEL Parameter

Parameter	Description
label	The label to be checked. To find existing label values, query the LABEL and TAG columns of the ALL_SA_LABELS view.

Example

The following example indicates if the label `2000` is a data label.

```
SET SERVEROUTPUT ON
BEGIN
 IF SA_UTL.DATA_LABEL(2000)
  THEN DBMS_OUTPUT.PUT_LINE('Label 2000 is a data label.');
 ELSE
  DBMS_OUTPUT.PUT_LINE('Label 2000 is not a data label.');
 END IF;
END;
/
```

D.8.6 SA_UTL.GREATEST_LBOUND

The `SA_UTL.GREATEST_LBOUND` function returns a label that is the greatest lower bound of the two label arguments.

Syntax

```
SA_UTL.GREATEST_LBOUND (
 label1 IN NUMBER,
 label2 IN NUMBER)
RETURN NUMBER;
```

Parameters

Table D-72 SA_UTL.GREATEST_LBOUND Parameters

Parameter	Description
`label1`	The first label to check. To find existing label values, query the `LABEL` and `TAG` columns of the `ALL_SA_LABELS` view.
`label2`	The second label to check

Examples

The following example compares existing label tags `3110` and `3111`.

```
SELECT SA_UTL.GREATEST_LBOUND(3110,3111) FROM DUAL;

SA_UTL.GREATEST_LBOUND(3110,3111)
---------------------------------
                             3111
```

D.8.7 SA_UTL.NUMERIC_LABEL

The `SA_UTL.NUMERIC_LABEL` function returns the current session label.

This function takes a policy name as the input parameter and returns a `NUMBER` value.

Syntax

```
SA_UTL.NUMERIC_LABEL (
  policy_name)
RETURN NUMBER;
```

Parameters

Table D-73 SA_UTL.NUMERIC_LABEL Parameter

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.

Example

The following example returns a the session numeric label for the user who is currently connected to the database instance.

```
SET SERVEROUTPUT ON
DECLARE
 num_label number;
BEGIN
 num_label := SA_UTL.NUMERIC_LABEL('hr_ols_pol');
 DBMS_OUTPUT.PUT_LINE('Numeric label: '||num_label);
END;
/
```

D.8.8 SA_UTL.NUMERIC_ROW_LABEL

The `SA_UTL.NUMERIC_ROW_LABEL` function returns the current row label. .

This function takes a policy name as the input parameter and returns a `NUMBER` value

Syntax

```
SA_UTL.NUMERIC_ROW_LABEL (
  policy_name)
RETURN NUMBER;
```

Parameters

Table D-74 SA_UTL.NUMERIC_ROW_LABEL Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.

Examples

The following example returns the session numeric row label for the user who is currently connected to the database instance.

```
SET SERVEROUTPUT ON
DECLARE
 num_row number;
BEGIN
 num_row := SA_UTL.NUMERIC_ROW_LABEL('hr_ols_pol');
 DBMS_OUTPUT.PUT_LINE('Numeric row label: '||num_row);
```

```
END;
/
```

D.8.9 SA_UTL.SET_LABEL

The `SA_UTL.SET_LABEL` procedure sets the label of the current database session.

The session's write label and row label are set to the subset of the label's compartments and groups that are authorized for write access.

Syntax

```
SA_UTL.SET_LABEL (
 policy_name IN VARCHAR2,
 label       IN LBAC_LABEL);
```

Parameters

Table D-75 SA_UTL.SET_LABEL Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`label`	The label to set as the session label. To find existing label values, query the `LABEL` column of the `ALL_SA_LABELS` view. You must pass this parameter through as an output of the `TO_LBAC_DATA_LABEL` function, which converts a label in character form to an `LBAC_LABEL` type. (The example in the next section shows how to do this.)

Example

The following example sets the label for the `hr_ols_pol` policy.

```
BEGIN
  SA_UTL.SET_LABEL (
    policy_name => 'hr_ols_pol',
    label       => to_lbac_data_label('hr_ols_pol','hs:pii'));
END;
/
```

Related Topics

- How Labeling Functions in Oracle Label Security Policies Works
  Labeling functions enable you to consider, in your rules for assigning labels, information drawn from the application context.

D.8.10 SA_UTL.SET_ROW_LABEL

The `SA_UTL.SET_ROW_LABEL` procedure sets the row label of the current database session.

The compartments and groups in the label must be a subset of compartments and groups in the session label that are authorized for write access.

Syntax

```
SA_UTL.SET_ROW_LABEL (
 policy_name IN VARCHAR2,
 label       IN BINARY_INTEGER);
```

Parameters

Table D-76 SA_UTL.SET_ROW_LABEL Parameters

Parameter	Description
`policy_name`	Specifies the policy. To find existing policies, query the `POLICY_NAME` column of the `ALL_SA_POLICIES` data dictionary view.
`label`	The label to set as the session default row label. To find existing label values, query the `LABEL` column of the `ALL_SA_LABELS` view.

Example

The following example sets the row label for the `hr_ols_pol` policy to `1111`.

```
BEGIN
 SA_UTL.SET_ROW_LABEL (
  policy_name         => 'hr_ols_pol',
  label               => 1111);
END;
/
```

Related Topics

- SA_SESSION Session Management PL/SQL Package
  The `SA_SESSION` PL/SQL package manages session behavior for user authorizations.

E

Oracle Label Security Tables and Views

Oracle Label Security provides data dictionary tables, data dictionary views, and an user-created auditing view.

- Oracle Database Data Dictionary Tables
  Oracle Label Security does not label the Oracle data dictionary tables; access is controlled by standard Oracle Database system and object privileges.
- Oracle Label Security Data Dictionary Views
  Oracle Label Security maintains an independent set of data dictionary views, which are exempt from any policy enforcement.
- Oracle Label Security User-Created Auditing View
  The `SA_AUDIT_ADMIN.CREATE_VIEW` procedure can be used to create an audit trail view for a specific policy.

E.1 Oracle Database Data Dictionary Tables

Oracle Label Security does not label the Oracle data dictionary tables; access is controlled by standard Oracle Database system and object privileges.

See Also:

Oracle Database Reference for detailed information about all data dictionary tables and views

E.2 Oracle Label Security Data Dictionary Views

Oracle Label Security maintains an independent set of data dictionary views, which are exempt from any policy enforcement.

Access to the data dictionary views is granted by default to the `SELECT_CATALOG_ROLE`, a standard Oracle Database role that lets you examine the Oracle Database data dictionary.

- About Oracle Label Security Data Dictionary Views
  The Oracle Label Security data dictionary views are used only for traditional auditing configurations.
- ALL_SA_AUDIT_OPTIONS View
  The `ALL_SA_AUDIT_OPTIONS` data dictionary view shows for the current user Oracle Label Security auditing options, based on the `SA_AUDIT_ADMIN.AUDIT` procedure settings.
- ALL_SA_COMPARTMENTS
  The `ALL_SA_COMPARTMENTS` data dictionary view shows information for the current user about Oracle Label Security policy compartments, based on the `SA_COMPONENTS.CREATE_COMPARTMENT` procedure settings.

- ALL_SA_DATA_LABELS
  The `ALL_SA_DATA_LABELS` data dictionary view shows for the current user Oracle Label Security policy labels and tags, based on the `SA_LABEL_ADMIN.CREATE_LABEL` procedure settings.
- ALL_SA_GROUPS
  The `ALL_SA_GROUPS` data dictionary shows information about the current user's Oracle Label Security policy groups, based on the `SA_COMPONENTS.CREATE_GROUP` and `SA_COMPONENTS.ALTER_GROUP_PARENT` procedures.
- ALL_SA_LABELS
  The `ALL_SA_LABELS` data dictionary view shows for the current user information about the tags and types of labels, based on `SA_LABEL_ADMIN.CREATE_LABEL` and `SA_LABEL_ADMIN.ALTER_LABEL`.
- ALL_SA_LEVELS
  The `ALL_SA_LEVELS` data dictionary view shows for the current user information about levels, based on the `SA_COMPONENTS.CREATE_LEVEL` procedure.
- ALL_SA_POLICIES
  The `ALL_SA_POLICIES` data dictionary view shows for the current user information about Oracle Label Security policies, based on the `SA_SYSDBA.CREATE_POLICY` procedure.
- ALL_SA_PROG_PRIVS
  The `ALL_SA_PROG_PRIVS` data dictionary view shows for the current user information about the policy-specific privileges for program units, based on `SA_USER_ADMIN.SET_PROG_PRIVS`.
- ALL_SA_SCHEMA_POLICIES
  The `ALL_SA_SCHEMA_POLICIES` data dictionary view shows for the current user information about policies applied to all tables in the schema, based on `SA_POLICY_ADMIN.APPLY_SCHEMA_POLICY`.
- ALL_SA_TABLE_POLICIES
  The `ALL_SA_TABLE_POLICIES` data dictionary view shows for the current user information about a policy added to a database table, based `SA_POLICY_ADMIN.APPLY_TABLE_POLICY` settings.
- ALL_SA_USERS
  The `ALL_SA_USERS` data dictionary view shows for the current user information about Oracle Label Security user privileges, based on `SA_USER_ADMIN.SET_USER_LABELS` and `SA_USER_ADMIN.SET_USER_PRIVS`.
- ALL_SA_USER_LABELS
  The `ALL_SA_USER_LABELS` data dictionary view shows for the current user label-specific information about users, based on the `SA_USER_ADMIN.SET_USER_` procedure settings.
- ALL_SA_USER_LEVELS
  The `ALL_SA_USER_LEVELS` data dictionary view shows for the current user the minimum and maximum levels assigned to users, based on the `SA_USER_ADMIN.SET_LEVELS` procdure.
- ALL_SA_USER_PRIVS
  The `ALL_SA_USER_PRIVS` data dictionary view shows for the current user policy-specific privileges granted to users, based on the `SA_USER_ADMIN.SET_USER_PRIVS` procedure.

- DBA_SA_AUDIT_OPTIONS
  The `DBA_SA_AUDIT_OPTIONS` data dictionary view data dictionary view shows for the entire database the Oracle Label Security audit options.
- DBA_SA_COMPARTMENTS
  The `ALL_SA_COMPARTMENTS` data dictionary view shows for the entire database information about Oracle Label Security policy compartments.
- DBA_SA_DATA_LABELS
  The `ALL_SA_DATA_LABELS` data dictionary view shows for the entire database the labels and label tags for the specified Oracle Label Security policy.
- DBA_SA_GROUPS
  The `ALL_SA_GROUPS` data dictionary view shows for the entire database information about Oracle Label Security policy groups.
- DBA_SA_GROUP_HIERARCHY
  The `DBA_SA_GROUP_HIERARCHY` data dictionary view shows the hierarchy of groups (that is, parent-child relationships) in a policy.
- DBA_SA_LABELS
  The `DBA_SA_LABELS` data dictionary view shows for the entire database information about the tags and types of labels for a policy.
- DBA_SA_LEVELS
  The `DBA_SA_LEVELS` data dictionary view shows for the entire database information about levels associated with a policy.
- DBA_SA_POLICIES
  The `DBA_SA_POLICIES` data dictionary view shows for the entire database information about Oracle Label Security policies, based on the `SA_SYSDBA.CREATE_POLICY` procedure.
- DBA_SA_PROG_PRIVS
  The `DBA_SA_PROG_PRIVS` data dictionary view shows for the entire database information about the policy-specific privileges for program units.
- DBA_SA_SCHEMA_POLICIES
  The `DBA_SA_SCHEMA_POLICIES` data dictionary view shows for the entire database information about policies that have been applied to all tables in the schema.
- DBA_SA_TABLE_POLICIES
  The `DBA_SA_TABLE_POLICIES` data dictionary view shows for the entire database information about a policy that has been added to a database table.
- DBA_SA_USERS
  The `DBA_SA_USERS` data dictionary view shows for the entire database information about the privileges that Oracle Label Security users have.
- DBA_SA_USER_COMPARTMENTS
  The `DBA_SA_USER_COMPARTMENTS` data dictionary view shows for the entire database the user authorizations, based on the `SA_USER_ADMIN.ADD_COMPARTMENTS` procedure.
- DBA_SA_USER_GROUPS
  The `DBA_SA_USER_GROUPS` data dictionary view shows for the entire database the groups associated with users, based on the `SA_USER_ADMIN.ADD_GROUPS` procedure.
- DBA_SA_USER_LABELS
  The `DBA_SA_USER_LABELS` data dictionary view shows for the entire database label-specific information about users.

- DBA_SA_USER_LEVELS
  The `DBA_SA_USER_LEVELS` data dictionary view shows for the entire database the minimum and maximum levels that have been assigned to users.
- DBA_SA_USER_PRIVS
  The `DBA_SA_USER_PRIVS` data dictionary view shows for the current user the policy-specific privileges that have been granted to users.
- DBA_OLS_STATUS
  The `DBA_OLS_STATUS` data dictionary view shows the configuration status of Oracle Label Security in the database.
- USER_SA_SESSION
  The `USER_SA_SESSION` data dictionary view shows the security attribute values for the current database session.

E.2.1 About Oracle Label Security Data Dictionary Views

The Oracle Label Security data dictionary views are used only for traditional auditing configurations.

Even though traditional auditing is desupported (starting with Oracle Database 23ai), the Oracle Label Security data dictionary views will continue to capture audit records for any current existing traditional Label Security audit settings. Audit records for any Label Security unified audit policies are written to the unified audit trail.

E.2.2 ALL_SA_AUDIT_OPTIONS View

The `ALL_SA_AUDIT_OPTIONS` data dictionary view shows for the current user Oracle Label Security auditing options, based on the `SA_AUDIT_ADMIN.AUDIT` procedure settings.

Note:

This view captures audit records from existing traditional audit configurations. You cannot configure new audit configurations using the Oracle Label Security PL/SQL packages. You must create new audit configurations only with unified auditing. Unified audit records are written to the unified audit trail views, such as `UNIFIED_AUDIT_TRAIL`.

This view displays whether auditing is configured to generate audit records per session (`BY SESSION`) or per access (`BY ACCESS`) and for successful or unsuccessful operations. Possible values are as follows:

- A dash (`-`) indicates that the audit option is not set.
- The `S` character indicates that the audit option is set `BY SESSION`.
- The `A` character indicates that the audit option is set `BY ACCESS`.
- Each audit option has two possible settings, `WHENEVER SUCCESSFUL` and `WHENEVER NOT SUCCESSFUL`, separated by a slash (`/`).

For example, in the following output, user `jjones` is audited with the `BY ACCESS` audit type for successful actions involving policy-specific privileges. User `rlayton` is audited with the BY SESSION audit type: audit records are written for failed attempts to remove policies and for successful attempts at setting user authorizations.

```
SELECT * FROM DBA_SA_AUDIT_OPTIONS;

POLICY_NAME      USER_NAME     APY  REM   SET_  PRV
-----------      ------------  ---  ----  ----  ---
HR_OLS_POL       JJONES        -/-   -/-  -/-   A/-
HR_OLS_POL       RLAYTON       -/-  -/S   S/-   -/-
```

Column	Datatype	Null	Description
POLICY_NAME	VARCHAR2(30)	NOT NULL	Name of the Oracle Label Security policy
USER_NAME	VARCHAR2(128)	NOT NULL	Name of the user associated with the policy
APY	VARCHAR2(3)	NULL	Audit option; refers to the application of specified Oracle Label Security policies to tables and schemas
REM	VARCHAR2(3)	NULL	Audit option; refers to the removal of specified Oracle Label Security policies from tables and schemas
SET_	VARCHAR2(3)	NULL	Audit option; refers to the setting of user authorizations, and user and program privileges
PRV	VARCHAR2(3)	NULL	Audit option; refers to the use of all policy-specific privileges

E.2.3 ALL_SA_COMPARTMENTS

The ALL_SA_COMPARTMENTS data dictionary view shows information for the current user about Oracle Label Security policy compartments, based on the SA_COMPONENTS.CREATE_COMPARTMENT procedure settings.

Column	Datatype	Null	Description
POLICY_NAME	VARCHAR2(30)	NOT NULL	Name of the Oracle Label Security policy
COMP_NUM	NUMBER(4)	NOT NULL	Compartment number in the range of (0-9999)
SHORT_NAME	VARCHAR2(30)	NOT NULL	Short name for the compartment
LONG_NAME	VARCHAR2(80)	NOT NULL	Long name for the compartment

Related Topics

- SA_COMPONENTS.CREATE_COMPARTMENT
  The SA_COMPONENTS.CREATE_COMPARTMENT procedure creates a compartment and specify its short name and long name.

E.2.4 ALL_SA_DATA_LABELS

The ALL_SA_DATA_LABELS data dictionary view shows for the current user Oracle Label Security policy labels and tags, based on the SA_LABEL_ADMIN.CREATE_LABEL procedure settings.

Column	Datatype	Null	Description
POLICY_NAME	VARCHAR2(30)	NOT NULL	Name of the Oracle Label Security policy
LABEL	VARCHAR2(4000)	NULL	Short name of the level, compartment, or group that was specified as the label value
LABEL_TAG	NUMBER	NULL	Integer that represents the sort order of the label, relative to other policy labels (0-99999999)

Related Topics

- SA_LABEL_ADMIN.CREATE_LABEL
  The SA_LABEL_ADMIN.CREATE_LABEL procedure creates data labels.

E.2.5 ALL_SA_GROUPS

The ALL_SA_GROUPS data dictionary shows information about the current user's Oracle Label Security policy groups, based on the SA_COMPONENTS.CREATE_GROUP and SA_COMPONENTS.ALTER_GROUP_PARENT procedures.

Column	Datatype	Null	Description
POLICY_NAME	VARCHAR2(30)	NOT NULL	Name of the Oracle Label Security policy
GROUP_NUM	NUMBER(4)	NOT NULL	Group number (0-9999)
SHORT_NAME	VARCHAR2(30)	NOT NULL	Short name of the group
LONG_NAME	VARCHAR2(80)	NOT NULL	Long name of the group
PARENT_NUM	NUMBER(4)	NULL	Numerical ID for the associated parent group
PARENT_NAME	VARCHAR2(30)	NULL	Name of the group assigned as the parent for the group

Related Topics

- SA_COMPONENTS.CREATE_GROUP
  The SA_COMPONENTS.CREATE_GROUP procedure creates a group and specify its short name and long name, and optionally a parent group.
- SA_COMPONENTS.ALTER_GROUP_PARENT
  The SA_COMPONENTS.ALTER_GROUP_PARENT procedure changes the parent group associated with a particular group.

E.2.6 ALL_SA_LABELS

The `ALL_SA_LABELS` data dictionary view shows for the current user information about the tags and types of labels, based on `SA_LABEL_ADMIN.CREATE_LABEL` and `SA_LABEL_ADMIN.ALTER_LABEL`.

Access to `ALL_SA_LABELS` is `PUBLIC`. However, only the labels authorized for read access by the session are visible.

Column	Datatype	Null	Description
POLICY_NAME	VARCHAR2(30)	NOT NULL	Name of the Oracle Label Security policy
LABEL	VARCHAR2(4000)	NOT NULL	Short name of the level associated with this label
LABEL_TAG	NUMBER(30)	NOT NULL	Integer tag assigned to the label
LABEL_TYPE	VARCHAR2(15)	NULL	Type of label

Related Topics

- SA_LABEL_ADMIN.CREATE_LABEL
  The `SA_LABEL_ADMIN.CREATE_LABEL` procedure creates data labels.
- SA_LABEL_ADMIN.ALTER_LABEL
  The `SA_LABEL_ADMIN.ALTER_LABEL` procedure changes the character string label definition associated with a label tag.

E.2.7 ALL_SA_LEVELS

The `ALL_SA_LEVELS` data dictionary view shows for the current user information about levels, based on the `SA_COMPONENTS.CREATE_LEVEL` procedure.

Column	Datatype	Null	Description
POLICY_NAME	VARCHAR2(30)	NOT NULL	Name of the Oracle Label Security policy
LEVEL_NUM	NUMBER(4)	NOT NULL	Level number (0-9999)
SHORT_NAME	VARCHAR2(30)	NOT NULL	Short name for the level
LONG_NAME	VARCHAR2(80)	NOT NULL	Long name for the level

Related Topics

- SA_COMPONENTS.CREATE_LEVEL
  The `SA_COMPONENTS.CREATE_LEVEL` procedure creates a level and specify its short name and long name.

E.2.8 ALL_SA_POLICIES

The `ALL_SA_POLICIES` data dictionary view shows for the current user information about Oracle Label Security policies, based on the `SA_SYSDBA.CREATE_POLICY` procedure.

Column	Datatype	Null	Description
POLICY_NAME	VARCHAR2(30)	NOT NULL	Name of the Oracle Label Security policy
COLUMN_NAME	VARCHAR2(128)	NOT NULL	Name of the column that was added to tables protected by the policy
STATUS	VARCHAR2(8)	NULL	Whether the policy has been enabled or disabled
POLICY_OPTIONS	VARCHAR2(4000)	NULL	Options that were set for this policy See Categories of Policy Enforcement Options for a listing of the possible enforcement options.

Related Topics

- SA_SYSDBA.CREATE_POLICY
  The `SA_SYSDBA.CREATE_POLICY` procedure creates a new Oracle Label Security policy, defines a policy-specific column name, and specifies default policy options.

E.2.9 ALL_SA_PROG_PRIVS

The `ALL_SA_PROG_PRIVS` data dictionary view shows for the current user information about the policy-specific privileges for program units, based on `SA_USER_ADMIN.SET_PROG_PRIVS`.

Column	Datatype	Null	Description
SCHEMA_NAME	VARCHAR2(128)	NOT NULL	Name of the schema that contains the program unit
PROGRAM_NAME	VARCHAR(128)	NOT NULL	Program unit that was granted privileges
POLICY_NAME	VARCHAR2(30)	NOT NULL	Name of the Oracle Label Security policy
PROGRAM_PRIVILEGES	VARCHAR2(4000)	NULL	Policy-specific privileges. See About Granting Privileges to Users and Trusted Program Units for the Policy for list of possible privileges.

Related Topics

- SA_USER_ADMIN.SET_PROG_PRIVS
  The `SA_USER_ADMIN.SET_PROG_PRIVS` procedure sets policy-specific privileges for program units.

E.2.10 ALL_SA_SCHEMA_POLICIES

The `ALL_SA_SCHEMA_POLICIES` data dictionary view shows for the current user information about policies applied to all tables in the schema, based on `SA_POLICY_ADMIN.APPLY_SCHEMA_POLICY`.

Column	Datatype	Null	Description
POLICY_NAME	VARCHAR2(30)	NOT NULL	Name of the Oracle Label Security policy
SCHEMA_NAME	VARCHAR2(128)	NOT NULL	Name of the schema associated with this policy
STATUS	VARCHAR2(8)	NULL	Whether the policy has been enabled or disabled for the schema (by the SA_POLICY_ADMIN.APPLY_SCHEMA_POLICY or SA_POLICY_ADMIN.DISABLE_SCHEMA_POLICY for procedure)
SCHEMA_OPTIONS	VARCHAR2(4000)	NULL	Options that have been applied.

Related Topics

- SA_POLICY_ADMIN.APPLY_SCHEMA_POLICY
  The SA_POLICY_ADMIN.APPLY_SCHEMA_POLICY procedure applies a policy to all of the tables in a schema and enables the policy for these tables.
- Categories of Policy Enforcement Options
  Oracle Label Security enforces policies using three categories: label management options, access control options, and overriding options.

E.2.11 ALL_SA_TABLE_POLICIES

The ALL_SA_TABLE_POLICIES data dictionary view shows for the current user information about a policy added to a database table, based SA_POLICY_ADMIN.APPLY_TABLE_POLICY settings.

Column	Datatype	Null	Description
POLICY_NAME	VARCHAR2(30)	NOT NULL	Name of the Oracle Label Security policy
SCHEMA_NAME	VARCHAR2(128)	NOT NULL	Schema that contains the table that the policy protects
TABLE_NAME	VARCHAR2(128)	NOT NULL	Table to be protected by the policy
STATUS	VARCHAR2(8)	NULL	Whether the policy has been enabled or disabled for the table (by the SA_POLICY_ADMIN.APPLY_TABLE_POLICY or SA_POLICY_ADMIN.DISABLE_TABLE_POLICY for procedure)
TABLE_OPTIONS	VARCHAR2(4000)	NULL	Policy enforcement options to be used for the table
FUNCTION	VARCHAR2(1024)	NULL	Name of the function to return a label value to use as the default
PREDICATE	VARCHAR2(256)	NULL	Predicate to combine (using AND or OR) with the label-based predicate for READ_CONTROL

Related Topics

- SA_POLICY_ADMIN.APPLY_TABLE_POLICY
  The `SA_POLICY_ADMIN.APPLY_TABLE_POLICY` procedure adds the specified policy to a table.
- Categories of Policy Enforcement Options
  Oracle Label Security enforces policies using three categories: label management options, access control options, and overriding options.

E.2.12 ALL_SA_USERS

The `ALL_SA_USERS` data dictionary view shows for the current user information about Oracle Label Security user privileges, based on `SA_USER_ADMIN.SET_USER_LABELS` and `SA_USER_ADMIN.SET_USER_PRIVS`.

Column	Type	Null	Description
`USER_NAME`	`VARCHAR2(1024)`	`NOT NULL`	Name of the user
`POLICY_NAME`	`VARCHAR2(30)`	`NOT NULL`	Name of the Oracle Label Security policy
`USER_PRIVILEGES`	`VARCHAR2(4000)`	`NULL`	Policy-specific privileges granted to the user.
`MAX_READ_LABEL`	`VARCHAR2(4000)`	`NULL`	Label string to initialize the user's maximum authorized read label
`MAX_WRITE_LABEL`	`VARCHAR2(4000)`	`NULL`	Label string to initialize the user's maximum authorized write label
`MIN_WRITE_LABEL`	`VARCHAR2(4000)`	`NULL`	Label string to initialize the user's minimum authorized write label
`DEFAULT_READ_LABEL`	`VARCHAR2(4000)`	`NULL`	Label string to initialize the user's session label, including level, compartments, and groups, for read access
`DEFAULT_WRITE_LABEL`	`VARCHAR2(4000)`	`NULL`	Label string to initialize the user's session label, including level, compartments, and groups, for write access
`DEFAULT_ROW_LABEL`	`VARCHAR2(4000)`	`NULL`	Label string to initialize the program's row label; includes level, components, and groups
`USER_LABELS`	`VARCHAR2(4000)`	`NULL`	Retained solely for backward compatibility and will be removed in the next release. The `USER_LABELS` column is deprecated starting with Oracle Database 18c because it is redundant. The information in this column is displayed in other `ALL_SA_USERS` and `DBA_SA_USERS` columns.

Related Topics

- SA_USER_ADMIN.SET_USER_LABELS
  The SA_USER_ADMIN.SET_USER_LABELS procedure sets the user's levels, compartments, and groups using a set of labels, instead of the individual components.
- SA_USER_ADMIN.SET_USER_PRIVS
  The SA_USER_ADMIN.SET_USER_PRIVS procedure sets policy-specific privileges for users.
- About Granting Privileges to Users and Trusted Program Units for the Policy
  After you have authorized users for policy levels, compartments, and groups, you are ready to grant the user privileges.

E.2.13 ALL_SA_USER_LABELS

The ALL_SA_USER_LABELS data dictionary view shows for the current user label-specific information about users, based on the SA_USER_ADMIN.SET_USER_ procedure settings.

Column	Datatype	Null	Description
USER_NAME	VARCHAR2(1024)	NOT NULL	Name of the user
POLICY_NAME	VARCHAR2(30)	NOT NULL	Name of the Oracle Label Security policy
MAX_READ_LABEL	VARCHAR2(4000)	NOT NULL	Label string to initialize the user's maximum authorized read label
MAX_WRITE_LABEL	VARCHAR2(4000)	NULL	Label string to initialize the user's maximum authorized write label
MIN_WRITE_LABEL	VARCHAR2(4000)	NULL	Label string to initialize the user's minimum authorized write label
DEFAULT_READ_LABEL	VARCHAR2(4000)	NULL	Label string to initialize the user's session label, including level, compartments, and groups, for read access
DEFAULT_WRITE_LABEL	VARCHAR2(4000)	NULL	Label string to initialize the user's session label, including level, compartments, and groups, for write access
DEFAULT_ROW_LABEL	VARCHAR2(4000)	NULL	Label string to initialize the program's row label; includes level, components, and groups

Related Topics

- SA_USER_ADMIN.SET_USER_LABELS
  The SA_USER_ADMIN.SET_USER_LABELS procedure sets the user's levels, compartments, and groups using a set of labels, instead of the individual components.

E.2.14 ALL_SA_USER_LEVELS

The ALL_SA_USER_LEVELS data dictionary view shows for the current user the minimum and maximum levels assigned to users, based on the SA_USER_ADMIN.SET_LEVELS procdure.

It also lists the user's session label and row label default values.

Column	Datatype	Null	Description
POLICY_NAME	VARCHAR2(30)	NOT NULL	Name of the Oracle Label Security policy
USER_NAME	VARCHAR2(1024)	NOT NULL	Name of the user
MAX_LEVEL	VARCHAR2(30)	NOT NULL	Short name of the highest level for read and write access
MIN_LEVEL	VARCHAR2(30)	NOT NULL	Short name of the lowest level for read and write access
DEF_LEVEL	VARCHAR2(30)	NOT NULL	Short name of the default level
ROW_LEVEL	VARCHAR2(30)	NOT NULL	Short name of the row level

Related Topics

- SA_USER_ADMIN.SET_LEVELS
  The `SA_USER_ADMIN.SET_LEVELS` procedure assigns a user minimum and maximum levels and identifies default values for the user's session label and row label.

E.2.15 ALL_SA_USER_PRIVS

The `ALL_SA_USER_PRIVS` data dictionary view shows for the current user policy-specific privileges granted to users, based on the `SA_USER_ADMIN.SET_USER_PRIVS` procedure.

Column	Datatype	Null	Description
USER_NAME	VARCHAR2(1024)	NOT NULL	Name of the user
POLICY_NAME	VARCHAR2(30)	NOT NULL	Name of the Oracle Label Security policy
USER_PRIVILEGES	VARCHAR2(4000)	NULL	Policy-specific privileges granted to the user

Related Topics

- SA_USER_ADMIN.SET_USER_PRIVS
  The `SA_USER_ADMIN.SET_USER_PRIVS` procedure sets policy-specific privileges for users.
- About Granting Privileges to Users and Trusted Program Units for the Policy
  After you have authorized users for policy levels, compartments, and groups, you are ready to grant the user privileges.

E.2.16 DBA_SA_AUDIT_OPTIONS

The `DBA_SA_AUDIT_OPTIONS` data dictionary view data dictionary view shows for the entire database the Oracle Label Security audit options.

> **Note:**
>
> This view captures audit records from existing traditional audit configurations. You cannot configure new audit configurations using the Oracle Label Security PL/SQL packages. You must create new audit configurations only with unified auditing. Unified audit records are written to the unified audit trail views, such as `UNIFIED_AUDIT_TRAIL`.

The columns for this view are the same as `ALL_SA_AUDIT_OPTIONS`.

Related Topics

- ALL_SA_AUDIT_OPTIONS View
  The `ALL_SA_AUDIT_OPTIONS` data dictionary view shows for the current user Oracle Label Security auditing options, based on the `SA_AUDIT_ADMIN.AUDIT` procedure settings.

E.2.17 DBA_SA_COMPARTMENTS

The `ALL_SA_COMPARTMENTS` data dictionary view shows for the entire database information about Oracle Label Security policy compartments.

Its columns are the same as `ALL_SA_COMPARTMENTS`.

Related Topics

- ALL_SA_COMPARTMENTS
  The `ALL_SA_COMPARTMENTS` data dictionary view shows information for the current user about Oracle Label Security policy compartments, based on the `SA_COMPONENTS.CREATE_COMPARTMENT` procedure settings.

E.2.18 DBA_SA_DATA_LABELS

The `ALL_SA_DATA_LABELS` data dictionary view shows for the entire database the labels and label tags for the specified Oracle Label Security policy.

Its columns are the same as `ALL_SA_DATA_LABELS`.

Related Topics

- ALL_SA_DATA_LABELS
  The `ALL_SA_DATA_LABELS` data dictionary view shows for the current user Oracle Label Security policy labels and tags, based on the `SA_LABEL_ADMIN.CREATE_LABEL` procedure settings.

E.2.19 DBA_SA_GROUPS

The ALL_SA_GROUPS data dictionary view shows for the entire database information about Oracle Label Security policy groups.

Its columns are the same as ALL_SA_GROUPS.

Related Topics

- ALL_SA_GROUPS
  The ALL_SA_GROUPS data dictionary shows information about the current user's Oracle Label Security policy groups, based on the SA_COMPONENTS.CREATE_GROUP and SA_COMPONENTS.ALTER_GROUP_PARENT procedures.

E.2.20 DBA_SA_GROUP_HIERARCHY

The DBA_SA_GROUP_HIERARCHY data dictionary view shows the hierarchy of groups (that is, parent-child relationships) in a policy.

Column	Type	Null	Description
POLICY_NAME	VARCHAR2(30)	NOT NULL	Name of the Oracle Label Security policy
HIERARCHY_LEVEL	NUMBER	NULL	Indicates the level of a particular group in a group hierarchy. A group with no parent group will have HIERARCHY_LEVEL 1. Its child group will have HIERARCHY_LEVEL 2 and so on. For example, consider these groups in the following order: 1. G1, G4 2. G2, G5 3. G3 Here, G1 and G4 have HIERARCHY_LEVEL 1; G2 and G5 have HIERARCHY_LEVEL 2, and G3 has HIERARCHY_LEVEL 3. The parent-child relationships are: • G3 is the child group of G2, and G2 is the child group of G1. • G5 is the child group of G4.
GROUP_NAME	VARCHAR2(4000)	NULL	Short name of the group intended to indicate the hierarchy level

E.2.21 DBA_SA_LABELS

The DBA_SA_LABELS data dictionary view shows for the entire database information about the tags and types of labels for a policy.

Its columns are the same as ALL_SA_LABELS.

Related Topics

- ALL_SA_LABELS
  The ALL_SA_LABELS data dictionary view shows for the current user information about the tags and types of labels, based on SA_LABEL_ADMIN.CREATE_LABEL and SA_LABEL_ADMIN.ALTER_LABEL.

E.2.22 DBA_SA_LEVELS

The DBA_SA_LEVELS data dictionary view shows for the entire database information about levels associated with a policy.

Its columns are the same as ALL_SA_LEVELS.

Related Topics

- ALL_SA_LABELS
  The ALL_SA_LABELS data dictionary view shows for the current user information about the tags and types of labels, based on SA_LABEL_ADMIN.CREATE_LABEL and SA_LABEL_ADMIN.ALTER_LABEL.

E.2.23 DBA_SA_POLICIES

The DBA_SA_POLICIES data dictionary view shows for the entire database information about Oracle Label Security policies, based on the SA_SYSDBA.CREATE_POLICY procedure.

This view also shows whether the policy has been enabled or disabled and its subscription status.

Column	Datatype	Null	Description
POLICY_NAME	VARCHAR2(30)	NOT NULL	Name of the Oracle Label Security policy
COLUMN_NAME	VARCHAR2(128)	NOT NULL	Name of the column that was added to tables protected by the policy
STATUS	VARCHAR2(8)	NULL	Whether the policy has been enabled or disabled
POLICY_OPTIONS	VARCHAR2(4000)	NULL	Options that were set for this policy. See Categories of Policy Enforcement Options for a listing of the possible enforcement options.
POLICY_SUBSCRIBED	VARCHAR2(5)	NULL	Indicates the policy's subscription status, based on the SA_POLICY_ADMIN.POLICY_SUBSCRIBE or SA_POLICY_ADMIN.POLICY_UNSUBSCRIBE procedure

E.2.24 DBA_SA_PROG_PRIVS

The DBA_SA_PROG_PRIVS data dictionary view shows for the entire database information about the policy-specific privileges for program units.

Its columns are the same as ALL_SA_PROG_PRIVS.

Related Topics

- ALL_SA_PROG_PRIVS
  The `ALL_SA_PROG_PRIVS` data dictionary view shows for the current user information about the policy-specific privileges for program units, based on `SA_USER_ADMIN.SET_PROG_PRIVS`.

E.2.25 DBA_SA_SCHEMA_POLICIES

The `DBA_SA_SCHEMA_POLICIES` data dictionary view shows for the entire database information about policies that have been applied to all tables in the schema.

Its columns are the same as `ALL_SA_SCHEMA_POLICIES`.

Related Topics

- ALL_SA_SCHEMA_POLICIES
  The `ALL_SA_SCHEMA_POLICIES` data dictionary view shows for the current user information about policies applied to all tables in the schema, based on `SA_POLICY_ADMIN.APPLY_SCHEMA_POLICY`.

E.2.26 DBA_SA_TABLE_POLICIES

The `DBA_SA_TABLE_POLICIES` data dictionary view shows for the entire database information about a policy that has been added to a database table.

Its columns are the same as `ALL_SA_TABLE_POLICIES`.

Related Topics

- ALL_SA_SCHEMA_POLICIES
  The `ALL_SA_SCHEMA_POLICIES` data dictionary view shows for the current user information about policies applied to all tables in the schema, based on `SA_POLICY_ADMIN.APPLY_SCHEMA_POLICY`.

E.2.27 DBA_SA_USERS

The `DBA_SA_USERS` data dictionary view shows for the entire database information about the privileges that Oracle Label Security users have.

Its columns are the same as `ALL_SA_USERS`.

Related Topics

- ALL_SA_USERS
  The `ALL_SA_USERS` data dictionary view shows for the current user information about Oracle Label Security user privileges, based on `SA_USER_ADMIN.SET_USER_LABELS` and `SA_USER_ADMIN.SET_USER_PRIVS`.

E.2.28 DBA_SA_USER_COMPARTMENTS

The `DBA_SA_USER_COMPARTMENTS` data dictionary view shows for the entire database the user authorizations, based on the `SA_USER_ADMIN.ADD_COMPARTMENTS` procedure.

This view also indicates whether the compartments are authorized for write and read privileges

Column	Datatype	Null	Description
`POLICY_NAME`	`VARCHAR2(30)`	`NOT NULL`	Name of the Oracle Label Security policy
`USER_NAME`	`VARCHAR2(1024)`	`NOT NULL`	Name of the user
`COMP`	`VARCHAR2(30)`	`NOT NULL`	Short name of compartments that were added
`RW_ACCESS`	`VARCHAR2(5)`	`NULL`	Access mode. Possible values are: • `SA_UTL.READ_ONLY` indicates no write access • `SA_UTL.READ_WRITE` indicates that write is authorized
`DEF_COMP`	`VARCHAR2(1)`	`NOT NULL`	Whether the compartments are in the default compartments
`ROW_COMP`	`VARCHAR2(1)`	`NOT NULL`	whether the compartments are in the row label

Related Topics

- SA_USER_ADMIN.ADD_COMPARTMENTS
  The `SA_USER_ADMIN.ADD_COMPARTMENTS` procedure adds (assigns) compartments to a user's authorizations, indicating if the compartments are authorized for write and read privileges.

E.2.29 DBA_SA_USER_GROUPS

The `DBA_SA_USER_GROUPS` data dictionary view shows for the entire database the groups associated with users, based on the `SA_USER_ADMIN.ADD_GROUPS` procedure.

Column	Datatype	Null	Description
`POLICY_NAME`	`VARCHAR2(30)`	`NOT NULL`	Name of the Oracle Label Security policy
`USER_NAME`	`VARCHAR2(1024)`	`NOT NULL`	Name of the user
`GRP`	`VARCHAR2(30)`	`NOT NULL`	Short name of groups that were added
`RW_ACCESS`	`VARCHAR2(5)`	`NULL`	Access mode. Possible values are: • `SA_UTL.READ_ONLY` indicates read-only access • `SA_UTL.READ_WRITE` indicates read and write access
`DEF_GROUP`	`VARCHAR2(1)`	`NOT NULL`	Whether the group is in a default group
`ROW_GROUP`	`VARCHAR2(1)`	`NOT NULL`	Whether the group is in a label

Related Topics

- SA_USER_ADMIN.ADD_GROUPS
  The `SA_USER_ADMIN.ADD_GROUPS` procedure adds (assigns) groups to a user, indicating if the groups are authorized for write and read privileges.

E.2.30 DBA_SA_USER_LABELS

The `DBA_SA_USER_LABELS` data dictionary view shows for the entire database label-specific information about users.

Its columns are the same as `ALL_SA_USER_LABELS`.

Related Topics

- ALL_SA_USER_LABELS
  The `ALL_SA_USER_LABELS` data dictionary view shows for the current user label-specific information about users, based on the `SA_USER_ADMIN.SET_USER_` procedure settings.

E.2.31 DBA_SA_USER_LEVELS

The `DBA_SA_USER_LEVELS` data dictionary view shows for the entire database the minimum and maximum levels that have been assigned to users.

This view also shows the default values for the user's session label and row label.

Its columns are the same as `ALL_SA_USER_LEVELS`.

Related Topics

- ALL_SA_USER_LEVELS
  The `ALL_SA_USER_LEVELS` data dictionary view shows for the current user the minimum and maximum levels assigned to users, based on the `SA_USER_ADMIN.SET_LEVELS` procdure.

E.2.32 DBA_SA_USER_PRIVS

The `DBA_SA_USER_PRIVS` data dictionary view shows for the current user the policy-specific privileges that have been granted to users.

Its columns are the same as `ALL_SA_USER_PRIVS`.

Related Topics

- ALL_SA_USER_PRIVS
  The `ALL_SA_USER_PRIVS` data dictionary view shows for the current user policy-specific privileges granted to users, based on the `SA_USER_ADMIN.SET_USER_PRIVS` procedure.

E.2.33 DBA_OLS_STATUS

The `DBA_OLS_STATUS` data dictionary view shows the configuration status of Oracle Label Security in the database.

Column	Datatype	Null	Description
`NAME`	`VARCHAR2(20)`	`NULL`	Name of the status. Values are: • `OLS_CONFIGURE_STATUS` • `OLS_ENABLE_STATUS`

Column	Datatype	Null	Description
STATUS	VARCHAR2(5)	NULL	Indicates the status of the feature mentioned in the corresponding name column. For example, a TRUE value for the OLS_CONFIGURE_STATUS status says that Oracle Label Security has been configured.
DESCRIPTIO N	VARCHAR2(400 0)	NULL	Description of the status: • OLS_CONFIGURE_STATUS:Determines if Oracle Label Security is configured. • OLS_ENABLE_STATUS: Determines if Oracle Label Security is enabled.

E.2.34 USER_SA_SESSION

The USER_SA_SESSION data dictionary view shows the security attribute values for the current database session.

Access to this view is PUBLIC.

Column	Datatype	Null	Description
POLICY_NAME	VARCHAR2(30)	NOT NULL	Name of the Oracle Label Security policy
SA_USER_NAME	VARCHAR2(4000)	NULL	Name of the current session user
PRIVS	VARCHAR2(4000)	NULL	Current session privileges
MAX_READ_LABEL	VARCHAR2(4000)	NULL	Label string that initialized the user's maximum authorized read label
MAX_WRITE_LABEL	VARCHAR2(4000)	NULL	Label string that initialized the user's maximum authorized write label
MIN_LEVEL	VARCHAR2(4000)	NULL	Minimum Oracle Label Security level authorized for the session
LABEL	VARCHAR2(4000)	NULL	Label for the current database session
COMP_WRITE	VARCHAR2(4000)	NULL	Compartments to which the user is authorized to write
GROUP_WRITE	VARCHAR2(4000)	NULL	Groups to which the user is authorized to write
ROW_LABEL	VARCHAR2(4000)	NULL	Row label that is associated with the policy for the current session

E.3 Oracle Label Security User-Created Auditing View

The SA_AUDIT_ADMIN.CREATE_VIEW procedure can be used to create an audit trail view for a specific policy.

By default, this view is named DBA_*policyname*_AUDIT_TRAIL.

Column	Datatype	Null	Description
`USERNAME`	`VARCHAR2(128)`	`NULL`	Name of the user whose actions were audited
`USERHOST`	`VARCHAR2(128)`	`NULL`	Client host machine name
`TERMINAL`	`VARCHAR2(255)`	`NULL`	Identifier of the user's terminal
`TIMESTAMP`	`DATE`	`NULL`	Date and time of the creation of the audit trail entry (date and time of user login for entries created by `AUDIT SESSION`) in the local database session time zone
`OWNER`	`VARCHAR2(128)`	`NULL`	Creator of the object affected by the action
`OBJ_NAME`	`VARCHAR2(128)`	`NULL`	Name of the object affected by the action
`ACTION`	`NUMBER`	`NOT NULL`	Numeric action type code. The corresponding name of the action type is in the `ACTION_NAME` column.
`ACTION_NAME`	`VARCHAR2(47)`	`NULL`	Name of the action type corresponding to the numeric code in the `ACTION` column
`COMMENT_TEXT`	`VARCHAR2(4000)`	`NULL`	Text comment on the audit trail entry, providing more information about the statement audited Also indicates how the user was authenticated. The method can be one of the following: • `DATABASE`: Authentication was done by password • `NETWORK`: Authentication was done by Oracle Net Services or by strong authentication
`SESSIONID`	`NUMBER`	`NOT NULL`	Numeric ID for each Oracle session
`ENTRYID`	`NUMBER`	`NOT NULL`	Numeric ID for each audit trail entry in the session
`STATEMENTID`	`NUMBER`	`NOT NULL`	Numeric ID for each statement run
`RETURNCODE`	`NUMBER`	`NOT NULL`	Oracle error code generated by the action. Some useful values: • `0`: Action succeeded • `2004`: Security violation
`EXTENDED_TIME STAMP`	`TIMESTAMP (6) WITH TIME ZONE`	`NULL`	Timestamp of the creation of the audit trail entry (timestamp of user login for entries created by `AUDIT SESSION`) in UTC (Coordinated Universal Time) time zone
`OLS_COL`	`VARCHAR2(4000)`	`NULL`	Name of the column that was added to the tables that Oracle Label Security protects

Related Topics

- SA_AUDIT_ADMIN.CREATE_VIEW
  The `SA_AUDIT_ADMIN.CREATE_VIEW` procedure creates an audit trail view named `DBA_`*`policyname`*`_AUDIT_TRAIL`.

F

Oracle Label Security Restrictions

Several restrictions exist in this Oracle Label Security release.

These restrictions are as follows:

- `CREATE TABLE AS SELECT` restriction

  If you attempt to perform `CREATE TABLE AS SELECT` in a schema that is protected by an Oracle Label Security policy, then the statement will fail.

- Label tag restriction

  Label tags must be unique across the policies in the database. When you use multiple policies in a database, you cannot use the same numeric label tag in different policies.

- Export restriction

  Before Oracle Database 12c release 1 (12.1), the `LBACSYS` schema could not be exported due to the use of opaque types in Oracle Label Security. An export of the entire database (parameter `FULL=Y`) with Oracle Label Security installed can be done, except that the `LBACSYS` schema would not be exported.

  From Oracle Database release 12c on, this restriction has been removed. See Full Database Export for additional details on the database versions that the export can be supported from.

- Oracle Label Security removal restriction

  Do not perform a `DROP USER CASCADE` on the `LBACSYS` account.

  Connect to the database as user `SYS`, using the `AS SYSDBA` syntax, and run the file `$ORACLE_HOME/rdbms/admin/catnools.sql` to remove Oracle Label Security.

  See Uninstalling Oracle Label Security for information about the different values that you can use to uninstall Oracle Label Security.

- Shared schema support restriction

  The Oracle Label Security function `SET_ACCESS_PROFILE` can be used programmatically to set the label authorization profile to use after a user has been authenticated and mapped to a shared schema. Oracle Label Security does not enforce a mapping between users who are given label authorizations in Oracle Label Security and actual database users.

- Hidden columns restriction

  PL/SQL does not recognize references to hidden columns in tables. A compiler error will be generated.

- Database objects restriction

  Oracle Label Security does not support the following objects:

  - External tables
  - Blockchain tables
  - Immutable tables

- Analytic workspace tables
- Common objects
- Application common objects
- JSON duality views
- Materialized views

G

Frequently Asked Questions about Oracle Label Security

Customers have frequently asked questions about Oracle Label Security.

- Who Uses Oracle Label Security?
  Sensitivity labels can categorize data in virtually every industry.
- How Can Oracle Label Security Address My Security Needs?
  Oracle Label Security can label data and restrict access with a high degree of granularity.
- Should I Use Oracle Label Security to Protect All My Tables?
  No, you should not use Oracle Label Security to protect all of your tables.
- What Is the Difference Between Oracle Virtual Private Database and Oracle Label Security?
  Oracle Virtual Private Database (VPD) is provided at no additional cost with the Enterprise Edition of Oracle Database.
- Can I Combine Oracle Virtual Private Database and Oracle Label Security?
  Yes. You can use a `WHERE` clause or a VPD policy.
- Can I Use Oracle Label Security with Oracle E-Business Suite?
  Oracle Applications use Oracle Virtual Private Database (VPD) to provide new functionality and security protections.
- Can I Use Oracle Label Security with Oracle Database Vault?
  Oracle Database Vault and Oracle Label Security can be used together within the same database.
- Does Oracle Label Security Provide Column-Level Access Control?
  No, Oracle Label Security is not column aware.
- Can I Base Secure Application Roles on Oracle Label Security?
  Yes, you can base secure application roles on Oracle Label Security.
- What Are Trusted Stored Program Units?
  Trusted stored program units are stored procedures, functions, and packages that run with the system and object privileges (DAC) of the definer.
- Does VPD or OLS Add an Additional Column to the Protected Table?
  When you apply an Oracle Label Security (OLS) policy to a table, the policy adds an additional column to the table.
- Why Should the Additional OLS Row Label Column Be Hidden?
  Most applications are designed with access control mechanisms in mind, so Oracle Label Security must do this transparently.

G.1 Who Uses Oracle Label Security?

Sensitivity labels can categorize data in virtually every industry.

These industries include health care, law enforcement, energy, retail, national security, and defense industries.

The following list gives some examples of sensitivity labels:

- Internal
- ConfidentialPhysician OnlyHighly SensitiveWidget CorporationConfidential: Chicago OperationSensitive: Finance : EuropeTop SecretUnclassified

G.2 How Can Oracle Label Security Address My Security Needs?

Oracle Label Security can label data and restrict access with a high degree of granularity.

This is especially useful when multiple organizations or companies share a single application. Sensitivity labels can be used to restrict application users to an organization or to a subset of data within an organization.

Data privacy is important to consumers and regulatory measures continue to be announced. Oracle Label Security can be used to implement privacy policies on data, restricting access to only those who have a need-to-know.

G.3 Should I Use Oracle Label Security to Protect All My Tables?

No, you should not use Oracle Label Security to protect all of your tables.

The traditional Oracle discretionary access control (DAC) object privileges such as `SELECT`, `INSERT`, `UPDATE`, and `DELETE` combined with database roles and stored procedures are sufficient in most cases. You can find a user's privileges by querying the `DBA_SYS_PRIVS` data dictionary view.

In addition, there are many other ways that you can protect access to your database tables, such using Oracle Virtual Private Database (VPD), Oracle Database Vault, Oracle Data Redaction, Transparent Data Encryption (TDE), or Transparent Sensitive Data Protection (TSDP).

G.4 What Is the Difference Between Oracle Virtual Private Database and Oracle Label Security?

Oracle Virtual Private Database (VPD) is provided at no additional cost with the Enterprise Edition of Oracle Database.

Oracle Label Security is an add-on security option for the Oracle Database Enterprise Edition.

Oracle VPD is a term used for several powerful security features like, fine grained access control (FGAC), application context and global application context. VPD policies are written using `PL/SQL`, and can be assigned to an individual table or view. An information request, that accesses a table or view protected by VPD, is modified according to the policy assigned to the table or view.

VPD policies can be as simple as enforcing access during business hours. VPD policies can restrict access by comparing the value of an attribute in an individual row with an application context value. Global application context allows an application context to be accessed across multiple database sessions, reducing or eliminating the need to create a separate application context for each user session.

Oracle Label Security is an out-of-the-box solution for row level security. No coding or software development is required, allowing the administrator to focus completely on the policy. Oracle Label Security provides an interface for creating policies, specifying enforcement options, defining data sensitivity labels, establishing user label authorizations, and protecting individual tables or schemes.

Data sensitivity labels provide a powerful and flexible method of restricting access to data. For example, data belonging to different organizations or companies can be separated using data sensitivity labels and selectively shared between companies by changing the data sensitivity label.

Depending on the complexity of the security policy, Oracle Virtual Private Database may be the preferred method for implementing your security policy. Oracle Label Security is best suited for situations where access control decisions need to be based on the sensitivity of the information.

G.5 Can I Combine Oracle Virtual Private Database and Oracle Label Security?

Yes. You can use a `WHERE` clause or a VPD policy.

- A `WHERE` clause can be appended to an OLS policy, which provides one more level of granularity. An example would be that users, regardless of their label authorizations, are only allowed to connect from a specific IP address or subnet, and during business hours only.
- A VPD policy, whether column sensitive or not, can evaluate user labels and determine access to columns and rows without the need to apply data labels.

G.6 Can I Use Oracle Label Security with Oracle E-Business Suite?

Oracle Applications use Oracle Virtual Private Database (VPD) to provide new functionality and security protections.

In addition, you can use other Oracle security products with Oracle E-Business Suite, such as Oracle Database Vault. Contact Oracle Support for more information.

G.7 Can I Use Oracle Label Security with Oracle Database Vault?

Oracle Database Vault and Oracle Label Security can be used together within the same database.

An Oracle Database Vault realm can protect a table that is also protected by an Oracle Label Security policy. The realm can protect the entire table and the Oracle Label Security can provide row level security for users that need to access the table data.

In addition, Oracle Label Security can be used together with Database Vault features. You can assign Oracle Label Security labels to Database Vault Factors. These labels are then merged with the user clearance labels, following the algorithms documented in Merging Labels with the MERGE_LABEL Function, before access control decisions are being made by comparing the merged user labels with the row labels.

The following example on the Oracle Technology Network Web site discusses using Oracle Label security along with Oracle Database Vault features:

`http://www.oracle.com/technetwork/database/security/label-security-factors-093209.html`

G.8 Does Oracle Label Security Provide Column-Level Access Control?

No, Oracle Label Security is not column aware.

This behavior is available with Virtual Private Database (VPD). A VPD policy can be written so that it only becomes active when a certain column is part of a SQL statement against a protected table. If the *column sensitivity* switch is on, then VPD either returns only those rows for which the sensitive column values are accessible to the user, or it returns all rows with all cells in the sensitive column being empty, except those values that the user is allowed to see.

The following link on the Oracle Technology Network Web site contains an example:

`http://www.oracle.com/technetwork/database/security/index-088277.html`

A column-sensitive VPD policy can determine access to a specific column by evaluating OLS user labels, which this example demonstrates:

`http://www.oracle.com/technetwork/database/security/ols-cs1-099558.html`

G.9 Can I Base Secure Application Roles on Oracle Label Security?

Yes, you can base secure application roles on Oracle Label Security.

The procedure that determines if the `SET ROLE` command is run can evaluate OLS user labels. In this case, the OLS policy does not need to be applied to a table, since row labels are not part of this solution.

G.10 What Are Trusted Stored Program Units?

Trusted stored program units are stored procedures, functions, and packages that run with the system and object privileges (DAC) of the definer.

If the invoker is a user with Oracle Label Security user clearances (labels), the procedure runs with a combination of the definer's DAC privileges and the invoker's security clearances.

Trusted stored procedures are procedures that are either granted the Oracle Label Security privilege `FULL` or `READ`. When a trusted stored program unit is run, the policy privileges in force are a combination of the invoking user's privileges and the program unit's privileges.

G.11 Does VPD or OLS Add an Additional Column to the Protected Table?

When you apply an Oracle Label Security (OLS) policy to a table, the policy adds an additional column to the table.

The name of this column needs to be specified when the policy is initially created.

An existing column can be used to store the OLS row labels. This column must have the `NUMBER(10)` data type.

Oracle Virtual Private Database (VPD) does not add an additional column to the protected table.

G.12 Why Should the Additional OLS Row Label Column Be Hidden?

Most applications are designed with access control mechanisms in mind, so Oracle Label Security must do this transparently.

When an application queries a table with a `SELECT FROM` *`tablename`* statement, it returns all columns, including the unhidden label column. Existing applications may not be designed to display an additional column, and malfunction. However, if the label column is hidden, then it is displayed only when its name is included in the SQL statement. A `SELECT FROM` *`tablename`* would return all columns as expected by the application, excluding the hidden OLS column.

H

Troubleshooting Oracle Label Security

Oracle provides guidance for troubleshooting Oracle Label Security tasks.

- Invalid SYS.OLS_ENFORCEMENT and SYS.LBAC_EXP Packages
  If the `SYS.OLS_ENFORCEMENT` and `SYS.LBAC_EXP` PL/SQL packages are invalid, then Oracle Label Security may not have been installed correctly.
- Addressing ORA-6550 and ORA-942 Errors
  The `ORA-6550` and `ORA-942` errors indicate that the Oracle Label Security component of the Oracle Database upgrade is invalid.

H.1 Invalid SYS.OLS_ENFORCEMENT and SYS.LBAC_EXP Packages

If the `SYS.OLS_ENFORCEMENT` and `SYS.LBAC_EXP` PL/SQL packages are invalid, then Oracle Label Security may not have been installed correctly.

To remedy this problem, you must validate the `SYS.OLS_ENFORCEMENT` and `SYS.LBAC_EXP` package objects.

1. Log into the PDB as a user who has the `SYSDBA` administrative privilege.

   To find the available PDBs, query the `DBA_PDBS` data dictionary view. To check the current PDB, run the `show con_name` command.
2. Run the following command to validate the two packages.

   ```
   @?/rdbms/admin/catols.sql
   ```
3. Run the following commands:

   ```
   EXEC LBACSYS.CONFIGURE_OLS
   EXEC LBACSYS.OLS_ENFORCEMENT.ENABLE_OLS
   ```
4. Restart the database instance.
5. Check the `INVALID` objects to see if the above two items are still listed.
6. If Oracle Label Security must be disabled after you validate these objects, then do the following:
   a. Disable Oracle Label Security.

      ```
      EXEC LBACSYS.OLS_ENFORCEMENT.DISABLE_OLS;
      ```
   b. Restart the database.

H.2 Addressing ORA-6550 and ORA-942 Errors

The `ORA-6550` and `ORA-942` errors indicate that the Oracle Label Security component of the Oracle Database upgrade is invalid.

The errors occur during the execution of the `catuppst.sql` script. The following errors are encountered:

```
catuppst.sql unable to run in Database: XXXX   Id: 0
ERRORS FOUND: during upgrade CATCTL ERROR COUNT=5
------------------------------------------------------
Identifier OLS 23-01-10 01:44:51 Script =
/release/orabase/oracle/rdbms/admin/ols
ERROR = [ORA-06550: line 119, column 41: PL/SQL: ORA-00942: table or
view
does not exist
ORA-06550: line 118, column 5:
PL/SQL: SQL Statement ignored
```

1. Log into the PDB as a user who has the `SYSDBA` administrative privilege.

   To find the available PDBs, query the `DBA_PDBS` data dictionary view. To check the current PDB, run the `show con_name` command.
2. Disable and then reenable Oracle Label Security.
3. Recompile the objects.

   ```
   @$ORACLE_HOME/rdbms/admin/utlrp.sql
   ```
4. Run the following procedure to validate Oracle Label Security:

   ```
   EXECUTE SYS.VALIDATE_OLS;
   ```
5. If `utlrp.sql` compiles all the Oracle Label Security (`LDAPSYS`) objects successfully, and there are no errors during compilation, then run the following procedure:

   ```
   EXEC DBMS_REGISTRY.VALID('OLS');
   ```

Related Topics

- Disabling and Enabling Oracle Label Security
  You can disable and enable Oracle Label Security as necessary.

Index