Oracle® Label Security

Administrator’s Guide

Release 2 (9.2)

March 2002

Part No. A96578-01

Oracle Label Security Administrator’s Guide, Release 2 (9.2)

Part No. A96578-01

Author: Jeff Levinger

Contributing Author: Rita Moran

Contributors: Paul Needham, Rae Burns, Gary Murphy, Patrick Sack, Vikram Pesati, Shiu Wong, Ramprasad Sripada, Krishnamurthy Raghuraman, Douglas Kemp, Srvidya Tata

Graphic Designer: Valarie Moore

Contents

Send Us Your Comments

Oracle Label Security Administrator's Guide, Release 2 (9.2)

Part No. A96578-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this document. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most?

If you find any errors or have any other suggestions for improvement, please indicate the document title and part number, and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: infodev_us@oracle.com
- FAX: (650) 506-7227   Attn: Server Technologies Documentation Manager
- Postal service:
  Oracle Corporation
  Server Technologies Documentation
  500 Oracle Parkway, Mailstop 4op11
  Redwood Shores, CA 94065
  USA

If you would like a reply, please give your name, address, telephone number, and (optionally) electronic mail address.

If you have problems with the software, please contact your local Oracle Support Services.

Preface

The *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.

This preface contains these topics:

- Audience
- Organization
- Related Documentation
- Conventions
- Documentation Accessibility

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" on page -xxii. To use SQL*Loader, you must know how to use the file management facilities of your operating system.

Organization

This document contains:

Part I: Concepts

This part introduces basic conceptual information about Oracle Label Security.

Chapter 1, "Introduction to Oracle Label Security"

This chapter introduces Oracle Label Security in the larger context of data security. It gives an overview of computer security issues and data access controls, and outlines the architecture and major features of Oracle Label Security.

Chapter 2, "Understanding Data Labels and User Labels"

This chapter discusses the fundamental concepts of data labels and user authorizations, and introduces the terminology that will help you understand Oracle Label Security. It covers label components, label syntax and type, and explains how data labels and user authorizations work together.

Chapter 3, "Understanding Access Controls and Privileges"

This chapter presents the access controls and privileges which determine the *type* of access users can have to the rows affected. It introduces the concepts of session label and row label, and explains how rows are evaluated for access mediation.

Part II: Using Oracle Label Security Functionality

This part provides the information needed by users of Oracle Label Security policies.

Chapter 4, "Working with Labeled Data"

This chapter explains how to use Oracle Label Security features to manage labeled data. It then shows how to view and change the value of security attributes for a session.

Part III: Administering an Oracle Label Security Application

This part explains how to create and manage an Oracle Label Security application.

Chapter 5, "Creating an Oracle Label Security Policy"

This chapter explains how to create an Oracle Label Security policy, and its underlying label components and labels.

Chapter 6, "Administering User Labels and Privileges"

This chapter explains how you can set authorizations for users, and grant privileges to users or stored program units by means of the available Oracle Label Security packages, or Oracle Policy Manager.

Chapter 7, "Implementing Policy Options and Labeling Functions"

This chapter explains how to customize the enforcement of Oracle Label Security policies, and how to implement labeling functions and SQL predicates.

Chapter 8, "Applying Policies to Tables and Schemas"

This chapter describes the SA_POLICY_ADMIN package, which enables you to administer policies on tables and schemas.

Chapter 9, "Administering and Using Trusted Stored Program Units"

This chapter explains how to use trusted stored program units to enhance system security.

Chapter 10, "Auditing Under Oracle Label Security"

This chapter explains how Oracle Label Security supplements the Oracle9*i* audit facility by tracking use of its own administrative operations and policy privileges. It describes the SA_AUDIT_ADMIN package, which enables you to set and change the policy auditing options.

Chapter 11, "Using Oracle Label Security with a Distributed Database"

This chapter describes special considerations for using Oracle Label Security in a distributed configuration.

Chapter 12, "Performing DBA Functions Under Oracle Label Security"

The standard Oracle9*i* utilities can be used under Oracle Label Security, but certain restrictions apply, and extra steps may be required to get the expected results. This chapter describes these special considerations.

Chapter 13, "Releasability Using Inverse Groups"

This chapter discusses the Oracle Label Security implementation of releasability using inverse groups.

Part IV: Appendix

Appendix A, "Advanced Topics in Oracle Label Security"

This appendix describes dominance relationships, and other ways in which the relationships between labels can be analyzed. It also describes the OCI interface for setting session labels.

Appendix B, "Reference"

This appendix documents the MAX_LABEL_POLICIES initialization parameter, the Oracle Label Security data dictionary tables, and Oracle Label Security restrictions.

Related Documentation

For more information, see these Oracle resources:

- *Oracle9i Security Overview*
- *Oracle9i Database Concepts*
- *Oracle9i Application Developer's Guide - Fundamentals*
- *Oracle9i Database Administrator's Guide*
- *Oracle9i SQL Reference*
- *Oracle9i Database Reference*
- *Oracle9i Replication*
- *Oracle9i Database Utilities*

- *Oracle9i Database Performance Tuning Guide and Reference*

Many of the examples in the documentation set use the sample schemas of the seed database, which is installed by default when you install Oracle. Refer to *Oracle9i Sample Schemas* for information on how these schemas were created and how you can use them yourself.

In North America, printed documentation is available for sale in the Oracle Store at

`http://oraclestore.oracle.com/`

Customers in Europe, the Middle East, and Africa (EMEA) can purchase documentation from

`http://www.oraclebookshop.com/`

Other customers can contact their Oracle representative to purchase printed documentation.

To download free release notes, installation documentation, white papers, or other collateral, please visit the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and can be done at

`http://otn.oracle.com/admin/account/membership.html`

If you already have a username and password for OTN, then you can go directly to the documentation section of the OTN Web site at

`http://otn.oracle.com/docs/index.htm`

To access the database documentation search engine directly, please visit

`http://tahiti.oracle.com`

Conventions

This section describes the conventions used in the text and code examples of this documentation set. It describes:

- Conventions in Text
- Conventions in Code Examples

Conventions in Text

We use various conventions in text to help you more quickly identify special terms. The following table describes those conventions and provides examples of their use.

Convention	Meaning	Example
Bold	Bold typeface indicates terms that are defined in the text or terms that appear in a glossary, or both.	When you specify this clause, you create an **index-organized table**.
Italics	Italic typeface indicates book titles or emphasis.	*Oracle9i Database Concepts* Ensure that the recovery catalog and target database do *not* reside on the same disk.
`UPPERCASE monospace (fixed-width) font`	Uppercase monospace typeface indicates elements supplied by the system. Such elements include parameters, privileges, datatypes, RMAN keywords, SQL keywords, SQL*Plus or utility commands, packages and methods, as well as system-supplied column names, database objects and structures, usernames, and roles.	You can specify this clause only for a `NUMBER` column. You can back up the database by using the `BACKUP` command. Query the `TABLE_NAME` column in the `USER_TABLES` data dictionary view. Use the `DBMS_STATS.GENERATE_STATS` procedure.
`lowercase monospace (fixed-width) font`	Lowercase monospace typeface indicates executables, filenames, directory names, and sample user-supplied elements. Such elements include computer and database names, net service names, and connect identifiers, as well as user-supplied database objects and structures, column names, packages and classes, usernames and roles, program units, and parameter values. **Note:** Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.	Enter `sqlplus` to open SQL*Plus. The password is specified in the `orapwd` file. Back up the datafiles and control files in the `/disk1/oracle/dbs` directory. The `department_id`, `department_name`, and `location_id` columns are in the `hr.departments` table. Set the `QUERY_REWRITE_ENABLED` initialization parameter to `true.` Connect as `oe` user. The `JRepUtil` class implements these methods.
`lowercase italic monospace (fixed-width) font`	Lowercase italic monospace font represents placeholders or variables.	You can specify the *`parallel_clause`*. Run `U`*`old_release`*`.SQL` where *`old_release`* refers to the release you installed prior to upgrading.

Conventions in Code Examples

Code examples illustrate SQL, PL/SQL, SQL*Plus, or other command-line statements. They are displayed in a monospace (fixed-width) font and separated from normal text as shown in this example:

```
SELECT username FROM dba_users WHERE username = 'MIGRATE';
```

The following table describes typographic conventions used in code examples and provides examples of their use.

Convention	Meaning	Example
[ ]	Brackets enclose one or more optional items. Do not enter the brackets.	`DECIMAL (`*digits* `[ ,` *precision* `])`
{ }	Braces enclose two or more items, one of which is required. Do not enter the braces.	`{ENABLE | DISABLE}`

|	A vertical bar represents a choice of two or more options within brackets or braces. Enter one of the options. Do not enter the vertical bar.	`{ENABLE | DISABLE}`
		 `[COMPRESS | NOCOMPRESS]`

. . .	Horizontal ellipsis points indicate either:	
	■ That we have omitted parts of the code that are not directly related to the example	`CREATE TABLE ... AS` *subquery*`;`
	■ That you can repeat a portion of the code	`SELECT` *col1*`,` *col2*`, ... ,` *coln* `FROM employees;`
. . .	Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example.	

Other notation	You must enter symbols other than brackets, braces, vertical bars, and ellipsis points as shown.	`acctbal NUMBER(11,2);`
		 `acct    CONSTANT NUMBER(4) := 3;`

Italics	Italicized text indicates placeholders or variables for which you must supply particular values.	`CONNECT SYSTEM/`*system_password* `DB_NAME =` *database_name*

Convention	Meaning	Example

`UPPERCASE`	Uppercase typeface indicates elements supplied by the system. We show these terms in uppercase in order to distinguish them from terms you define. Unless terms appear in brackets, enter them in the order and with the spelling shown. However, because these terms are not case sensitive, you can enter them in lowercase.	`SELECT last_name, employee_id FROM employees;`
		`SELECT * FROM USER_TABLES;`
		`DROP TABLE hr.employees;`

`lowercase`	Lowercase typeface indicates programmatic elements that you supply. For example, lowercase indicates names of tables, columns, or files.
	Note: Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.
		`SELECT last_name, employee_id FROM employees;`
		`sqlplus hr/hr`
		`CREATE USER mjones IDENTIFIED BY ty3MU9;`

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Standards will continue to evolve over time, and Oracle Corporation is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For additional information, visit the Oracle Accessibility Program Web site at

`http://www.oracle.com/accessibility/`

Accessibility of Code Examples in Documentation JAWS, a Windows screen reader, may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, JAWS may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation This documentation may contain links to Web sites of other companies or organizations that Oracle Corporation does not own or control. Oracle Corporation neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Part I

Concepts

1

Introduction to Oracle Label Security

Oracle Label Security enables application developers to add label-based access control to their Oracle9*i* applications. It mediates access to rows in database tables based on a label contained in the row, and the label and privileges associated with each user session. Oracle Label Security is built on the virtual private database technology of Oracle9*i* Enterprise Edition. It includes the Oracle Policy Manager, a graphical user interface for ease of administration.

This chapter introduces Oracle Label Security in the larger context of data security. It contains the following sections:

- Computer Security and Data Access Controls
- Oracle Label Security Architecture
- Features of Oracle Label Security

Note: This book assumes that you understand the basic concepts and terminology of Oracle9*i* database administration and application development. It supplements core Oracle9*i* documentation by focusing on the extra considerations involved in using, administering, and developing applications for Oracle Label Security.

See Also: For a complete introduction to Oracle9*i* features and terminology, see *Oracle9i Database Concepts*

Computer Security and Data Access Controls

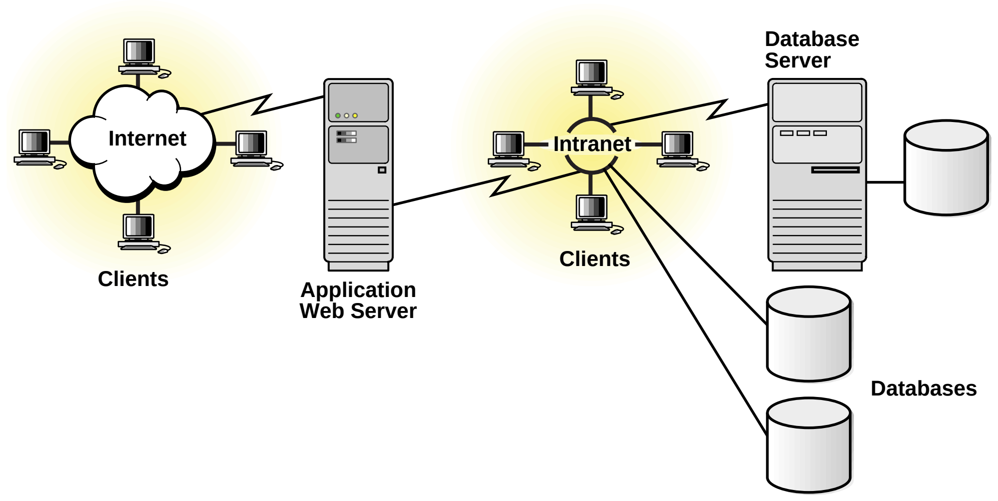
This section introduces basic concepts of computer security. It contains the following topics:

- Introduction to Computer Security
- Oracle Label Security and Security Standards
- Security Policies
- Access Control

Introduction to Computer Security

Computer security involves the protection of computerized data and processes from unauthorized modification, destruction, disclosure, or delay. In the Internet age, the risks to valuable and sensitive data are greater than ever before. Figure 1–1 shows the complex computing environment that your data security plan must encompass.

Figure 1–1 Scope of Data Security Needs



You must protect databases and the servers on which they reside; you must administer and protect the rights of internal database users; and you must guarantee the confidentiality of electronic commerce customers as they access your database. Oracle Corporation provides products to address the full spectrum of computer security issues.

Oracle Label Security and Security Standards

Oracle Corporation strives to create products which meet stringent international security standards. Security evaluation is a formal assessment process performed by independent bodies against national and international criteria. It provides external and objective assurance that a system meets the security criteria for which it was designed. Upon successful completion of an evaluation, a security rating is assigned to the system or product. This certification provides confidence in the security of information technology products and systems to commercial, government and military users.

Oracle9*i* is designed to meet the Database Management System Protection Profile (DBMS PP). Oracle Label Security is designed for evaluation under the ISO/IEC 15408 Common Criteria.

Security Policies

A database security policy is an implementation of an overall system security policy, which in turn is often derived from a broad, organizational security policy. The overall security policy can include:

Data Integrity Policy	Defines rules to ensure that information in the system is consistent
Availability Policy	Defines rules to ensure that information is available
Access Control Policy	Defines rules to prevent the unauthorized disclosure of information. Oracle Label Security provides one of many possible information access control policies. You can use it to define one or more customized policies for use at a given site.

Access Control

Access control is the process of defining a user's ability to read or write information. Application developers must decide which approach to access control best meets their needs.

- Discretionary Access Control
- Label-Based Access Control
- How Label-Based Access Control Works with Discretionary Access Control

Discretionary Access Control

Oracle9*i* provides *discretionary access control* (DAC). DAC is a means of controlling access to information through privileges, which are permissions to perform an operation within the system. On a table-by-table basis, DAC provides the SELECT, INSERT, UPDATE, and DELETE privileges. These privileges authorize the corresponding SQL operation upon the table.

With discretionary access control, access to data is controlled in a single dimension. The administrator grants users privileges which determine the operations (such as read, write) they can perform upon data. A *subject*, defined as a user and the processes or tasks running on behalf of that user, must have the appropriate privilege, such as the SELECT privilege, to access an *object*, such as a table or view. A user must first have the necessary DAC privileges to access data in an object.

In Oracle9*i*, row-level access control is available with the *virtual private database* (VPD) technology which is a standard feature of the Enterprise Edition. Virtual private database provides fine-grained access control which is data-driven, context-dependent, and row-based. You can implement VPD by writing a stored procedure to append a SQL predicate to each SQL statement to control row level access for that statement. For example, if John Doe (who belongs to Department 10) inputs the statement SELECT * FROM emp, you can use VPD to tack on the clause WHERE DEPT = 10. In this way query modification is used to restrict data access to certain rows.

See Also: *Oracle9i Application Developer's Guide - Fundamentals*

Label-Based Access Control

Label-based access is a way of controlling data on a row level. Each data row is given a label used to store information about data sensitivity. A label provides additional sophisticated access control rules in addition to those provided by discretionary access control. It further mediates access to a data row based on the identity and label of the user and the label of the row. This provides an additional level of access control to a system.

Label-based access control depends on the basic DAC policy; together these policies dictate the criteria by which access to an object is either permitted or denied. In most applications, a relatively small number of application tables will require label-based access controls. The protection provided by standard DAC will suffice for a majority of application tables.

How Label-Based Access Control Works with Discretionary Access Control

To be allowed access to a row, a user must satisfy both Oracle Label Security and Oracle9*i* DAC requirements. Oracle9*i* enforces DAC based on the user's system-level privileges and database object privileges. First a user must be authenticated to the Oracle9*i* database. Second, the user must have the DAC object and system privileges required for the operation.

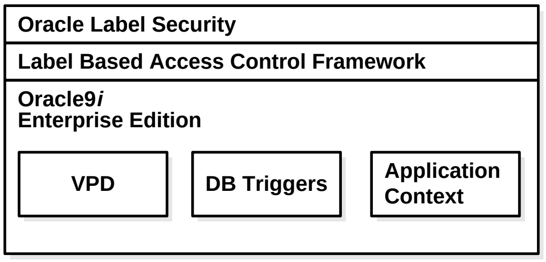
Finally, the user must meet the criteria added by Oracle Label Security. This product adheres to the label definitions, label hierarchies, and other security policy rules defined within the database by the site administrators. On top of this, Oracle Label Security enforces access based on the labels of the user and row, as well as the user's Oracle Label Security policy privileges.

Oracle Label Security is flexible and functional enough to support applications in a variety of production environments. It supports Oracle9*i* data integrity, availability, and recovery capabilities, as well as user accountability and auditing, while enforcing a site's security policy.

Oracle Label Security Architecture

Oracle Label Security is built on the virtual private database (VPD) technology found in the Oracle9*i* Enterprise Edition. It also uses the Application Context functionality of this product.

Figure 1–2 Oracle Label Security and Oracle9i Enterprise Edition

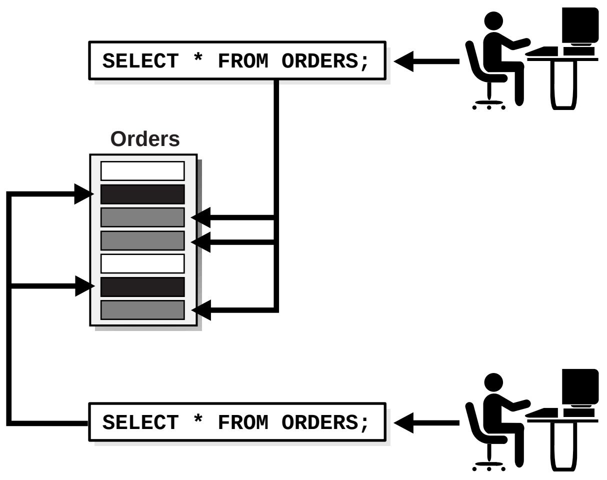

Oracle Label Security
Label Based Access Control Framework
Oracle9*i* Enterprise Edition
VPD
DB Triggers
Application Context

Oracle9*i* Enterprise Edition: Virtual Private Database Technology

VPD supports fine-grained access control to data rows. It provides an application programmatic interface (API) which allows security policies to be assigned to database tables and views. Using PL/SQL, developers and security administrators can create security policies with stored procedures, and bind the procedures to a table or view by means of a call to an RDBMS package. Such policies are based on the content of application data stored within the Oracle9*i* database, or based on context variables provided by Oracle9*i*. In this way, VPD permits access security mechanisms to be removed from applications, and centralized within Oracle9*i*.

As illustrated in Figure 1–3, VPD lets you associate security conditions with tables or views. In this example, when each user selects from the ORDERS table, the appropriate security condition is automatically enforced. The server automatically enforces security policies, no matter how the data is accessed. In this way, VPD eliminates the need to use many views to implement security.

Figure 1–3 Oracle9i Enterprise Edition Virtual Private Database Technology



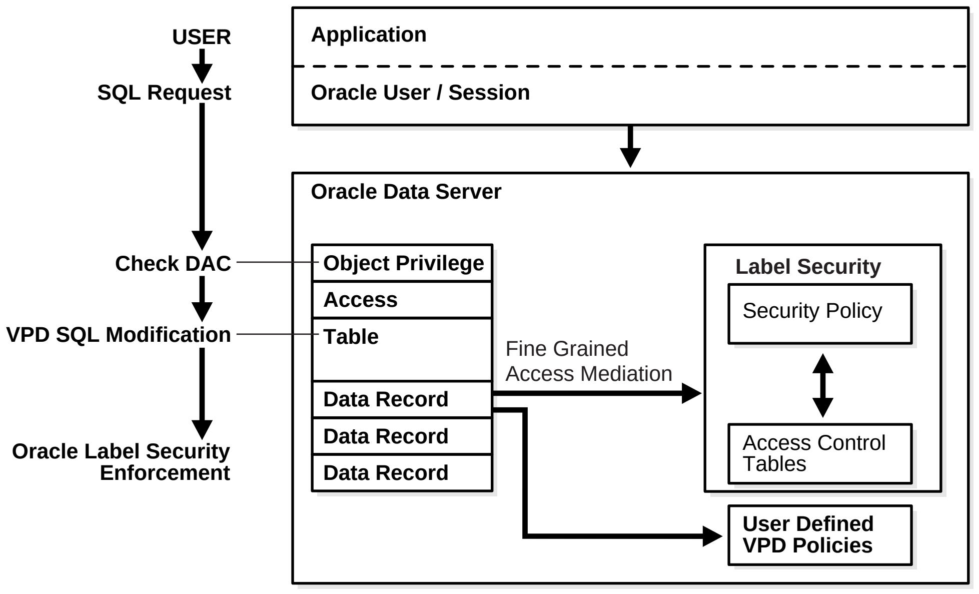
Oracle Label Security: An Out-of-the-Box VPD Policy

Oracle Label Security provides a functional, out-of-the-box VPD policy which enhances your ability to implement row-level security. It supplies an infrastructure—a label-based access control framework—whereby you can specify labels for users and data. It also enables you to create one or more custom security policies to be used for label access decisions. You can implement these policies without any knowledge of a programming language. There is no need to write additional code; in a single step you can apply a security policy to a given table. In this way, Oracle Label Security provides a straightforward, efficient way to implement fine-grained security policies using data labeling technology. Finally, the structure of Oracle Label Security labels provides a degree of granularity and flexibility which cannot easily be derived from the application data alone. Oracle Label Security is thus a generic solution which can be used in many different circumstances.

Figure 1–4 illustrates the process by which data is accessed under Oracle Label Security. Within an application and Oracle9*i* session, a user issues a SQL request. Oracle9*i* checks the DAC privileges, making sure the user has SELECT privileges on the table. Then it checks to see if a VPD policy has been attached to the table. It finds that the table is protected by Oracle Label Security. The SQL statement is modified on the fly.

Each data record has a label; Oracle Label Security is invoked for each row, to determine whether, based on the label, the user can or cannot access the row.

Figure 1–4 Oracle Label Security Architecture



Features of Oracle Label Security

Oracle Label Security provides additional row level security access controls to the underlying Oracle9*i* database. This section contains these topics:

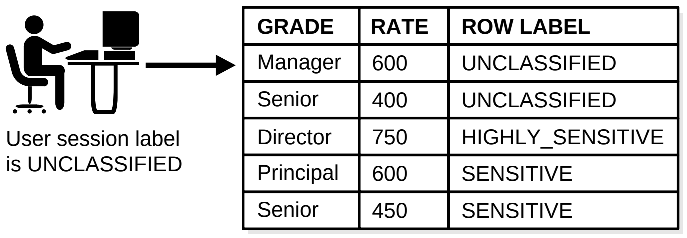
- Overview of Oracle Label Security Policy Functionality
- Label Policy Framework Features
- Auditing Features
- Oracle Label Security Distributed Capabilities

Overview of Oracle Label Security Policy Functionality

To create a customized Oracle Label Security policy, you define a set of labels and a set of rules that govern data access, based on these labels.

For example, assume that a user has SELECT privilege on an application table. As illustrated in Figure 1–5, when the user executes a SELECT statement, Oracle Label Security evaluates each row selected and determines whether the user can access it based on the privileges and access labels assigned to the user by the security administrator. Oracle Label Security can also be configured to perform security checks on UPDATE, DELETE, and INSERT statements.

Figure 1–5 Oracle Label Security Label-Based Security



GRADE	RATE	ROW LABEL
Manager	600	UNCLASSIFIED
Senior	400	UNCLASSIFIED
Director	750	HIGHLY_SENSITIVE
Principal	600	SENSITIVE
Senior	450	SENSITIVE

- Oracle Label Security enables a comprehensive set of access authorization privileges, to ensure that the data label can be protected—independent of the data itself.
- Oracle Label Security provides for flexible policy enforcement to handle special processing requirements.
- Individual application tables can be protected. All the tables in the application need not be protected by the policy. Lookup tables such as zip codes, for example, do not need to be protected.
- Oracle Label Security allows the security administrator to add special labeling functions and SQL predicates to the policy.
- You can create multiple Oracle Label Security policies. For example, a human resources policy could co-exist with a defense policy in the same database. Each of the policies can be independently configured, and have its own unique label definitions.

Label Policy Framework Features

Oracle Label Security adds label-based access controls to the Oracle9*i* object-relational database management system. Access to data is mediated based on these factors:

- the label associated with a row of data
- the label associated with a user session
- the policy privileges associated with a user session
- the policy enforcement options associated with a table

Consider, for example, a standard Data Manipulation Language operation (such as SELECT) performed upon a row of data. When evaluating this access request by a user with the CONFIDENTIAL label, to a data row labeled CONFIDENTIAL, Oracle Label Security determines that this access can, in fact, be achieved.

In this way, data of different sensitivities—or belonging to different companies—can be stored and managed on a single system, while preserving data security through standard Oracle access controls. Likewise, applications from a broad range of industries can use row labels to provide additional access control functionality where necessary.

Data Labels

In Oracle Label Security, each row of a table can be labeled as to its level of confidentiality. The label contains three components:

- a single level or sensitivity ranking
- one or more horizontal compartments or categories
- one or more hierarchical groups

The level specifies the sensitivity of the data. A typical government organization might define levels CONFIDENTIAL, SENSITIVE, and HIGHLY_SENSITIVE. A commercial organization might define a single level for COMPANY_CONFIDENTIAL data. The compartment component is non-hierarchical; compartments are typically defined to segregate data—such as data related to an ongoing strategic initiative. Finally, groups are used to record ownership and can be used hierarchically. For example, FINANCE and ENGINEERING groups can be defined as children of the CEO group, creating an ownership relation.

Labels can contain a single level component, a level combined with a set of either compartments or groups, or a level with both compartments and groups.

Label Authorizations

Users can be granted label authorizations which determine what kind of access (read or write) they have to the rows that are labeled.

Policy Privileges

Policy privileges enable a user or stored program unit to bypass aspects of the label-based access control policy. In addition, the administrator can authorize the user or program unit to perform specific actions, such as the ability of one user to assume the authorizations of a different user.

Privileges can be granted to program units, authorizing the procedure, rather than the user, to perform privileged operations. When only stored program units—and not individual users—have Oracle Label Security privileges, your system is most secure. Further, such program units encapsulate the policy, which minimizes the amount of application code that needs to be reviewed for security.

Policy Enforcement Options

In Oracle Label Security you can apply different enforcement options for maximum flexibility in controlling the different Data Manipulation Language operations that users can perform. For each operation—SELECT, INSERT, UPDATE, and DELETE—you can specify a particular type of enforcement of the security policy, for each table. In this way you can customize the label-based access controls on each table.

Summary: Four Aspects of Label-Based Row Access

When label-based access is enforced, a user's label must meet certain criteria, determined by policy definitions, to access a row within a protected table. These access controls act as a secondary access mediation check, on top of the discretionary access controls which have been implemented by the application developers.

In summary, Oracle Label Security provides four aspects of label-based access control:

- A user's label indicates the information that a user is permitted to access. The user's label also indicates the type of access (read or write) the user is allowed to perform.
- A row's label indicates the sensitivity of the information that the row contains.
- The policy privileges granted to a user enable him or her to bypass aspects of the label-based access control policy.
- The policy enforcement options on the table determine how access controls are enforced.

Auditing Features

Oracle Label Security supplements the Oracle9*i* audit facility by tracking the use of its own Oracle Label Security administrative operations and policy privileges. Under Oracle Label Security, audit trail records contain a label associated with the session that generated the audit, so that you can see the relationship between operations, data labels, and the label of the user performing the operation.

Oracle Label Security Distributed Capabilities

Oracle Label Security supports distributed operation when labels in the local and remote databases are compatible.

See Also: Chapter 11, "Using Oracle Label Security with a Distributed Database"

2 Understanding Data Labels and User Labels

This chapter discusses the fundamental concepts of data labels and user labels, and introduces the terminology that will help you understand Oracle Label Security.

The chapter includes:

Introduction to 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 or label of the data. This provides an additional level of security to a system.

With an Oracle Label Security policy, access to data is controlled in three dimensions:

Data Labels	Rows of data are labeled to indicate the level and nature of their sensitivity. A label on a row of data specifies the sensitivity of the information in the row and explicitly defines the criteria that must be met for a user to access that row.
User Labels	Users are assigned a range of levels, compartments, and groups which indicate their label authorizations. A label assigned to a user determines the user's access to labeled data.
Policy Privileges	Certain users may be given rights to perform special operations, and to access data beyond their label authorizations.

Note that the discussion here concerns *access* to data. The particular *type* of access (that is, the ability to read or to write the data in question) is covered in Chapter 3, "Understanding Access Controls and Privileges."

When a database table is protected by an Oracle Label Security policy, a column is added to the table. This policy label column contains the label information for each data row. The administrator can choose to display or hide this column.

- A label on a row of data specifies the sensitivity of the information in the row and explicitly defines the criteria that must be met for a user to access that row.
- Label authorizations assigned to a user determine the user's access to labeled data.

Label Components

This section describes the elements which make up a sensitivity label.

- Label Component Definitions and Valid Characters
- Levels
- Compartments
- Groups
- Industry Examples of Levels, Compartments, and Groups

Label Component Definitions and Valid Characters

A sensitivity label is a single attribute, with multiple components. All data labels must contain a level component; compartment and group components are optional. The administrator must define the label components before he or she can create labels.

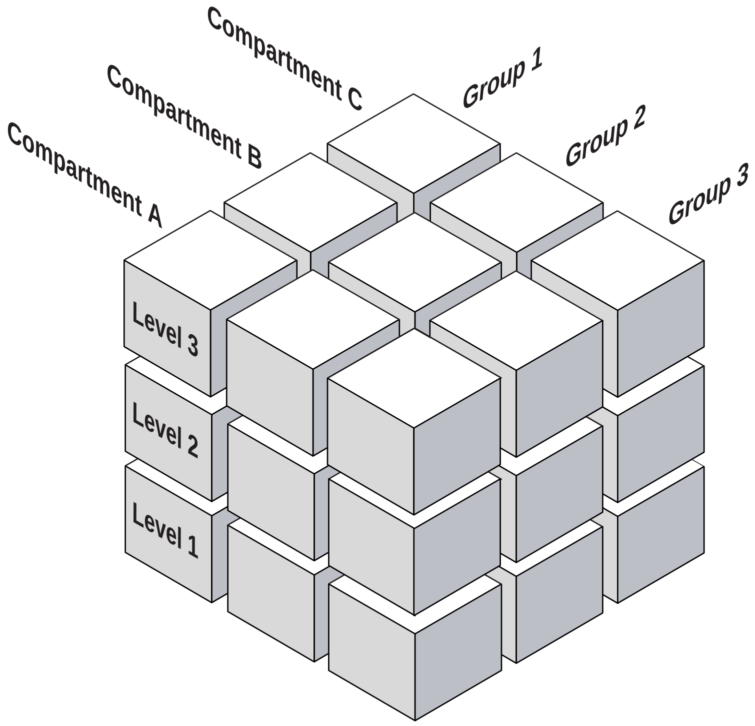
Table 2–1 Sensitivity Label Components

Component	Description	Examples
Level	A single specification of the labeled data's ordered sensitivity ranking	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 of organizations owning or accessing the data	EASTERN_REGION, WESTERN_REGION

Valid characters for all label component specifications include alphanumeric characters and underscores. Additionally, spaces can be used within the string. (Leading or 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, Groups



Levels

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.

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

For example, you can define a set of levels like the following:

Table 2–2 Level Example

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

Numeric Form	The numeric form of the level can range from 0 to 9999. Levels of sensitivity are ranked by this numeric value, so you must assign higher numbers to levels which are more sensitive, and lower numbers to levels which are less sensitive. In Table 2–2, 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. This enables you to insert additional levels between two pre-existing 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. When users manipulate the labels, they use only the short form of the component names.

Other sets of labels which users commonly define include TOP_SECRET, SECRET, CONFIDENTIAL, and UNCLASSIFIED; or TRADE_SECRET, PROPRIETARY, COMPANY_CONFIDENTIAL, PUBLIC_DOMAIN.

Note: In this guide, all labels (including "TOP_SECRET," "SECRET," "CONFIDENTIAL," and so on) are used as illustrations only.

Compartments

Compartments identify areas which describe the sensitivity of the labeled data. They provide a finer level of granularity within a level.

Compartments associate the data with one or more security areas. All of the data related to a particular project can be labeled with the same compartment. For example, you can define a set of compartments like the following:

Table 2–3 Compartment Example

Numeric Form	Long Form	Short Form
40	FINANCIAL	FINCL
30	CHEMICAL	CHEM
20	OPERATIONAL	OP

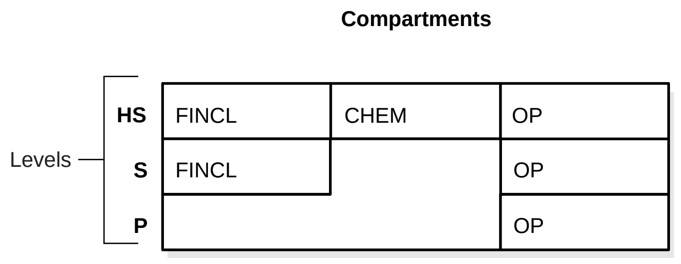
Numeric Form	The numeric form can range from 0 to 9999. The numeric form of the compartment does not indicate greater or less sensitivity. Rather, it controls display order of the short form compartment name in the label character string. For example, assume that a label is created which has all three compartments listed in Table 2–3, and a level of SENSITIVE. If the label containing the level and compartments is displayed in string format, it looks like this: `S:OP,CHEM,FINCL` This is because 20 comes before 30, and 30 before 40. By contrast, if the numeric form for the FINCL compartment were set to 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 can contain 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 up to 10,000 compartments.

All labels need not have all compartments. For example, you can specify HIGHLY_ SENSITIVE and CONFIDENTIAL levels with no compartments, and a SENSITIVE level which does contain compartments.

When you analyze your data's sensitivity, you may find that some compartments are only used at specific levels. Figure 2–2 shows how compartments can be used to categorize data.

Figure 2–2 Label Matrix



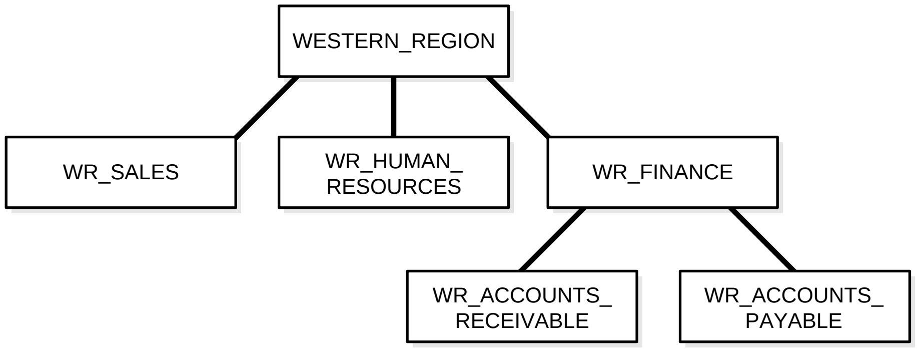
Here, compartments FINCL, CHEM, and OP are used with the level HIGHLY_SENSITIVE (40). The label HIGHLY_SENSITIVE:FINCL, CHEM indicates a level of 40 with the two named compartments. Compartment FINCL is not more sensitive than CHEM, nor is CHEM more sensitive than FINCL. Note also that some data in the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

Groups

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. For example, 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 further subdivided into WR_ACCOUNTS_RECEIVABLE and WR_ACCOUNTS_PAYABLE.

Table 2–4 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 2–4 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

Numeric Form

The numeric form of the group can range from 0 to 9999, and 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 simply controls display order of the short form group name in the label character string.

For example, assume that a label is created which has the level SENSITIVE, the compartment CHEMICAL, and the groups WESTERN_REGION and WR_HUMAN_RESOURCES as listed in Table 2–4. 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 up to 10,000 groups.

All labels need not have groups. When you analyze your data's sensitivity, 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 which does contain groups.

See Also: Chapter 13, "Releasability Using Inverse Groups"

Industry Examples of Levels, Compartments, and Groups

Table 2–5 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 2–5 Typical Levels, Compartments, and Groups, by Industry

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

Label Syntax and Type

After defining the label components, the administrator creates data labels by combining particular sets of level, compartments, and groups. Out of all the possible permutations of label components, the administrator specifies those combinations which will actually be used as valid data labels in the database.

This can be done using the Oracle Policy Manager graphical user interface, or using 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. It is not necessary to enter trailing delimiters in this syntax.

For example, the administrator might 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 which can be assigned to data. Oracle Label Security can also create valid data labels dynamically at runtime. Most users, however, prefer to create the labels manually in order to control 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—which is stored in the policy label column of the protected table.

See Also: Chapter 5, "Creating an Oracle Label Security Policy" for instructions on creating label components and labels

"Label Tags" on page 4-4

How Data Labels and User Labels Work Together

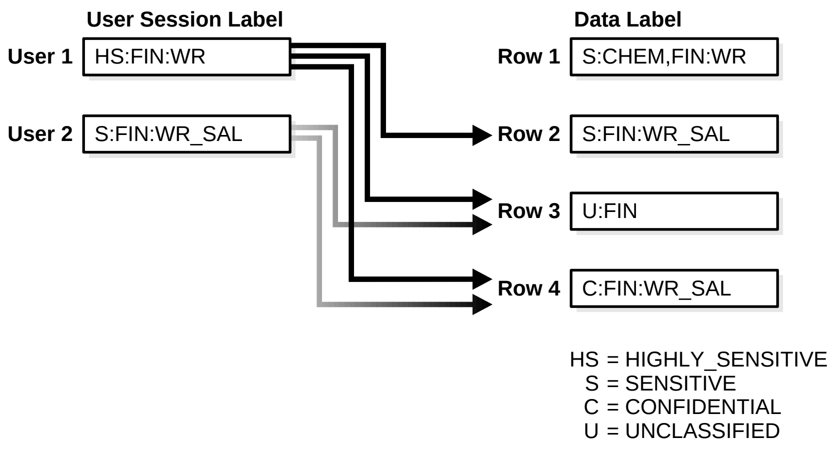
A user can only access data within the range of his or her own label authorizations. A user has:

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

For example, if a user is assigned a maximum level of SENSITIVE, then the user potentially has access to SENSITIVE, CONFIDENTIAL, and UNCLASSIFIED data. The user has no access to HIGHLY_SENSITIVE data.

Figure 2–4 shows how data labels and user labels work together, to provide access control in Oracle Label Security. Whereas 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 his or her range.

Figure 2–4 Example: Data Labels and User Labels

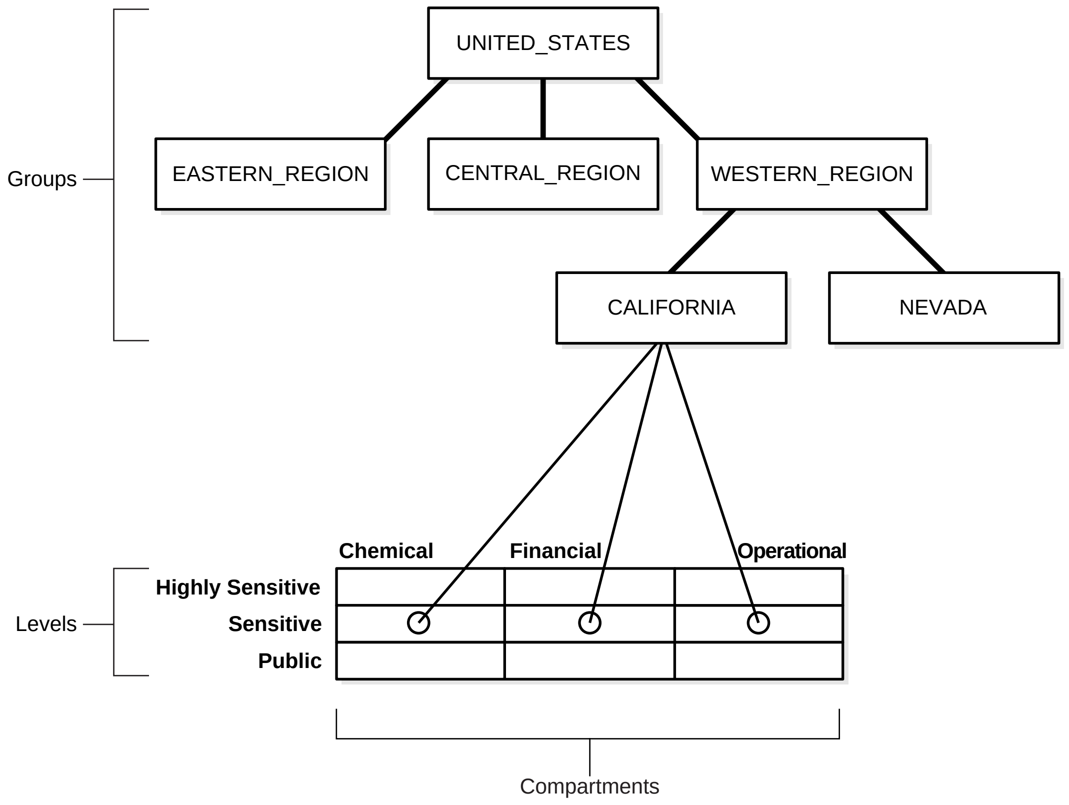


As shown in the figure, User 1 can access rows 2, 3, and 4 because her maximum level is HS; she has access to the FIN compartment; and her access to group WR hierarchically includes group WR_SAL. She cannot access row 1 because she does not have the CHEM compartment. (A user must have authorization for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

User 2 can access rows 3 and 4. His maximum level is S, which is less than HS in row 2. Although he has access to the FIN compartment, he only has authorization for group WR_SAL. He cannot, therefore, access row 1.

Figure 2–5 shows how data pertaining to an organizational hierarchy fits in to 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. Thus there may be SENSITIVE data which 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, that group may implicitly include child groups. Thus a user associated with the WESTERN_REGION group has access to all data; but a user associated with CALIFORNIA would only have access to data pertaining to that subgroup.

Administering Labels

Oracle Label Security provides administrative interfaces to define and manage the labels used in a database. You define labels in an Oracle database using Oracle Label Security packages, or using the Oracle Policy Manager. Initially, an administrator must define the levels, compartments, and groups that compose the labels, and then she or he can define the set of valid data labels for the contents of the database.

The 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) appropriate for the person's job function.

See Also: Chapter 8, "Applying Policies to Tables and Schemas" for information about the Oracle Label Security interfaces used to manage label components

3 Understanding Access Controls and Privileges

Chapter 2 introduced the concept of labels (with their levels, compartments, and groups) and the basic notion of access control based on the row's data label and the user's label. The present chapter examines the access controls and privileges which determine the *type* of access users can have to the rows affected.

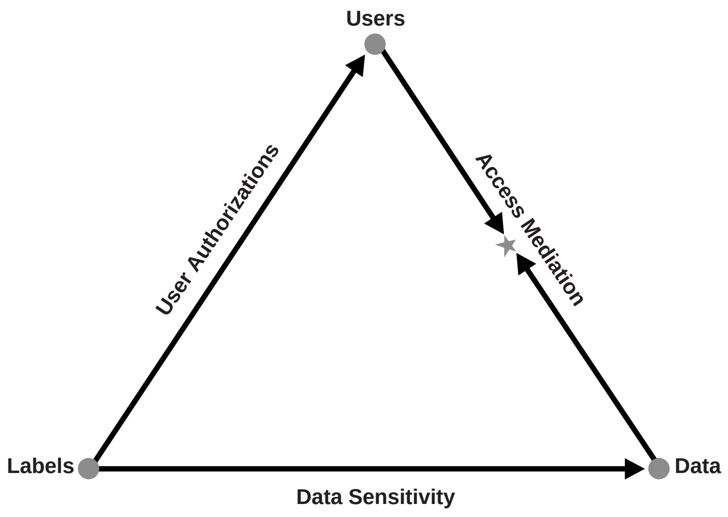
This chapter contains these sections:

Introduction to 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. Figure 3–1 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 upon their labels.

Figure 3–1 Relationships Between Users, Data, and Labels



Note: Oracle Label Security provides a number of enforcement options which affect the way in which access controls are applied to tables and schemas. This chapter assumes that all policy enforcement options are in effect.

For more information, see "Choosing Policy Options" on page 7-2.

Understanding Session Label and Row Label

This section introduces the basic user labels.

- The Session Label
- The Row Label
- Session Label Example

The Session Label

Each Oracle Label Security user has a set of authorizations which include:

- 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

When the administrator sets up these authorizations for the user, he or she also specifies the user's initial session label.

The *session label* is the particular combination of level, compartments, and groups at which a user works at any given time. The user can change the session label to any combination of his or her authorized components.

The Row Label

The *row label* is the particular label assigned by default to data which a user enters during a session. It can be set to any level, from the one specified in the user's current session label, down to the user's minimum level. It can include only compartments and groups contained in the current session label, and for which the user has write access. The user can change the row label to any label for which he or she is authorized.

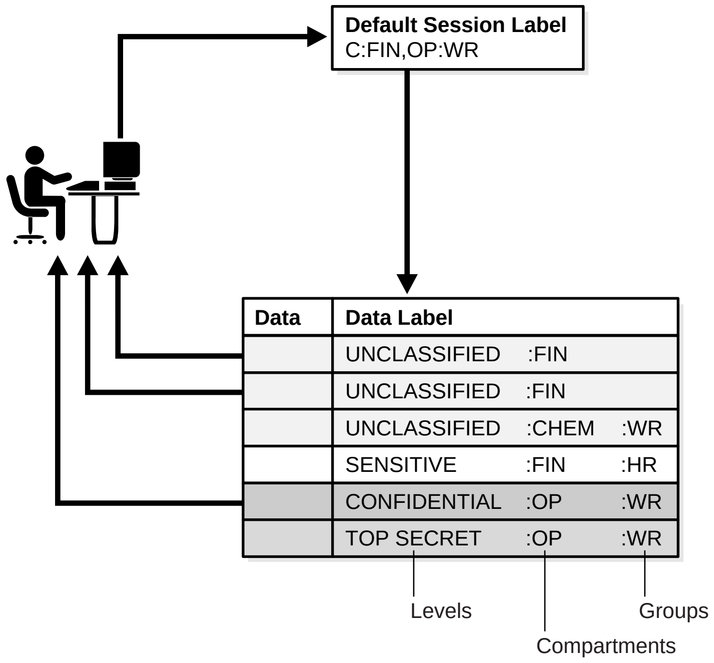
When the administrator sets up the user authorizations, he or she also specifies an initial default row label.

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. In Figure 3–2, the user's maximum level is SENSITIVE, and his minimum level is UNCLASSIFIED. However, his 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, even though 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



Data	Data Label		
	UNCLASSIFIED	:FIN	
	UNCLASSIFIED	:FIN	
	UNCLASSIFIED	:CHEM	:WR
	SENSITIVE	:FIN	:HR
	CONFIDENTIAL	:OP	:WR
	TOP SECRET	:OP	:WR

Understanding User Authorizations

There are two types of user authorizations:

- Authorizations Set by the Administrator
- Computed Session Labels

Authorizations Set by the Administrator

The administrator explicitly sets a number of user authorizations:

- Authorized Levels
- Authorized Compartments
- Authorized Groups

Authorized Levels

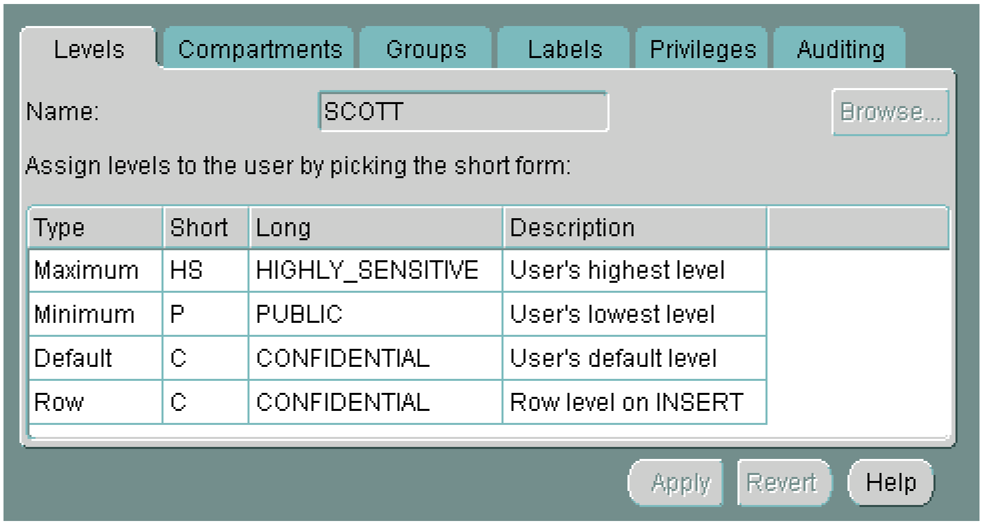
The administrator explicitly sets the following level authorizations:

Table 3–1 Authorized Levels Set by the Administrator

Authorization	Meaning
User Max Level	The maximum ranking of sensitivity which a user can access during read and write operations
User Min Level	The minimum ranking of sensitivity which 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 which is assumed by default when connecting to Oracle9*i*
User Default Row Level	The level which is used by default when inserting data into Oracle9*i*

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

Figure 3–3 Setting Up Authorized Levels



Name: SCOTT

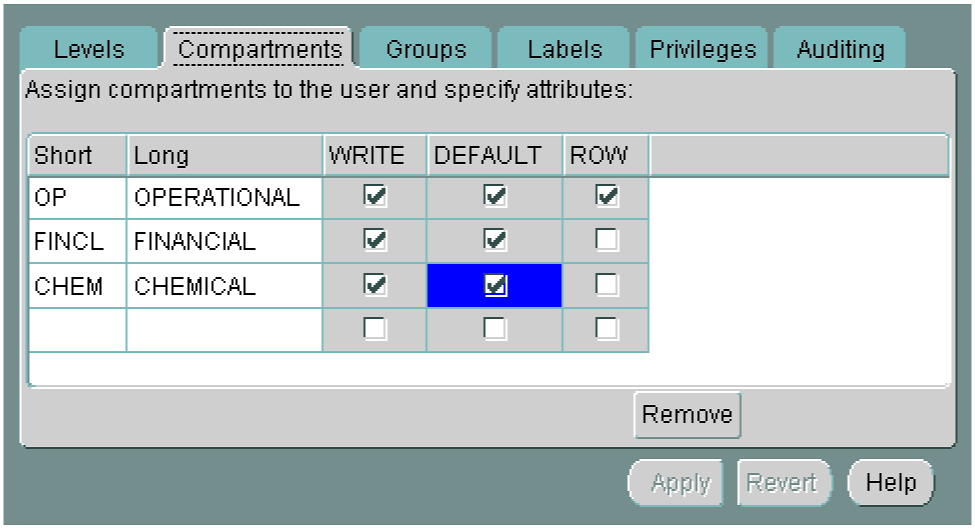
Assign levels to the user by picking the short form:

Type	Short	Long	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

Authorized Compartments

The administrator specifies the list of compartments which a user can place in her 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 which she does not have authorization to write. For example, in Oracle Policy Manager, the administrator might set the following authorizations:

Figure 3–4 Setting Up Authorized Compartments



Assign compartments to the user and specify attributes:

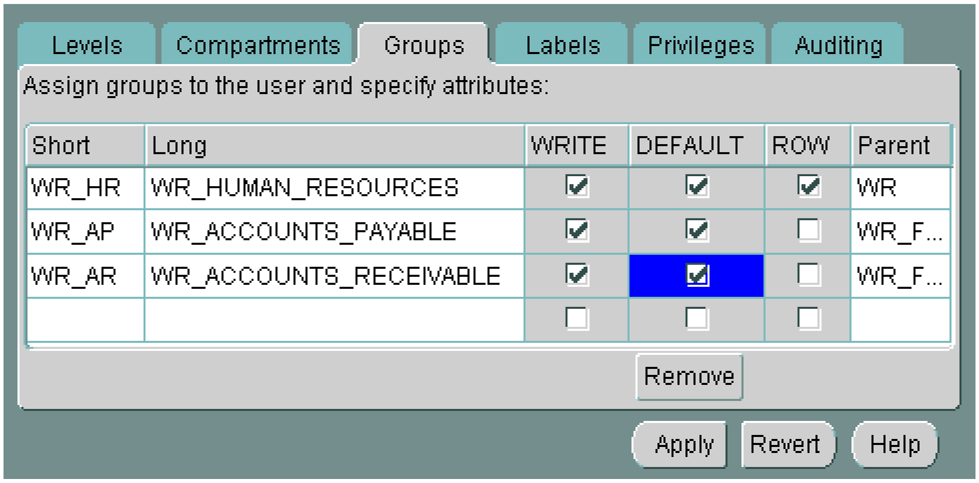
Short	Long	WRITE	DEFAULT	ROW
OP	OPERATIONAL	✓	✓	✓
FINCL	FINANCIAL	✓	✓	☐
CHEM	CHEMICAL	✓	✓	☐
		☐	☐	☐

In Figure 3–4, 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 LABEL_DEFAULT policy option must be in effect for this setting to be valid.

Authorized Groups

The administrator specifies the list of groups which a user can place in her session label. Write access must be explicitly given for each group listed. For example, in Oracle Policy Manager, the administrator might set the following authorizations:

Figure 3–5 Setting Up Authorized Groups



Assign groups to the user and specify attributes:

Short	Long	WRITE	DEFAULT	ROW	Parent
WR_HR	WR_HUMAN_RESOURCES	✓	✓	✓	WR
WR_AP	WR_ACCOUNTS_PAYABLE	✓	✓	☐	WR_F...
WR_AR	WR_ACCOUNTS_RECEIVABLE	✓	✓	☐	WR_F...
		☐	☐	☐	

In Figure 3–5, 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.

> **See Also:** Chapter 6, "Administering User Labels and Privileges" for instructions on setting the authorizations
>
> "LABEL_DEFAULT: Using the Session’s Default Row Label" on page 7-5

Computed Session Labels

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

Table 3–2 Computed Session Labels

Computed Label	Definition
Maximum Read Label	The user’s maximum level combined with his or her authorized compartments and groups.
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 which 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.

See Also: "Computed Labels with Inverse Groups" on page 13-5

How Labels Are Evaluated for Access Mediation

When a table is protected by an Oracle Label Security policy, the user's label components are compared to the row's label components to determine whether the user can access the data. In this way, Oracle Label Security evaluates whether the user is authorized to perform the requested operation on the data in the row. This section explains the rules and options by which user access is mediated. It contains these topics:

- Introduction to Read / Write Access
- The Oracle Label Security Algorithm for Read Access
- The Oracle Label Security Algorithm for Write Access

Introduction to Read/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 DML operations exist, within protected tables:

- read access
- write access

The user has a maximum authorization for the data he or she 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 to any level lower than his or her minimum level.

In addition, there is a list of compartments 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 individual compartments.

There is also a list of 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 individual groups.

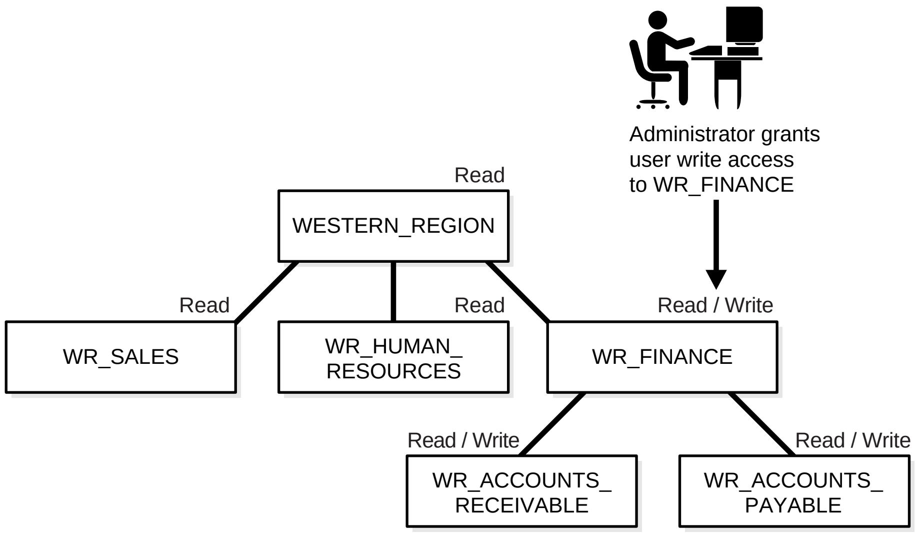
Propagation of Read/Write Authorizations on Groups

When groups are organized hierarchically, a user's assigned groups include all subgroups that are subordinate to the group to which she 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 3–6. If the user has read access to WESTERN_REGION, she also has read access to the three subgroups. The administrator can give the user write access to subgroup WR_FINANCE, without granting her 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 she also has read / write access 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, the administrator can grant her write access to WR_ACCOUNTS_RECEIVABLE, without affecting her read-only access to the higher-level groups.

Figure 3–6 Subgroup Inheritance of Read/Write Access



See Also: "Introduction to User Label and Privilege Management" on page 6-2

"How Inverse Groups Work" on page 13-4

The Oracle Label Security Algorithm for Read Access

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:

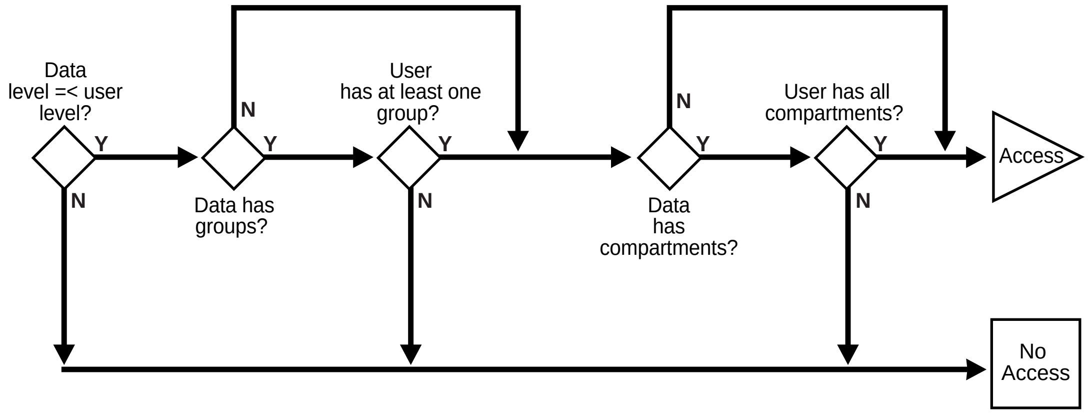
1. The user's level must be *greater than or equal to* the level of the data.
2. The user's label must include *at least one of the groups* which belong to the data (or the parent group of one such subgroup).
3. The user's label must include *all the compartments* which belong to the data.

If the user's label passes these tests, 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 her current session level. No privileges (other than FULL) allow the user to write below her minimum authorized level.

The label evaluation process proceeds from levels to groups to compartments, as illustrated in Figure 3–7. Note that if the data label is null or invalid, then the user is denied access.

Figure 3–7 Label Evaluation Process for Read Access


Data level =< user level?
Y
N
Data has groups?
N
Y
User has at least one group?
Y
N
Data has compartments?
N
Y
User has all compartments?
Y
N
Access
No Access

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

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 she or he possesses special Oracle Label Security privileges.

See Also: "Privileges Defined by Oracle Label Security Policies" on page 3-18

"Analyzing the Relationships Between Labels" on page A-2

"Algorithm for Read Access with Inverse Groups" on page 13-9

The Oracle Label Security Algorithm for Write Access

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 which he 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* which 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.

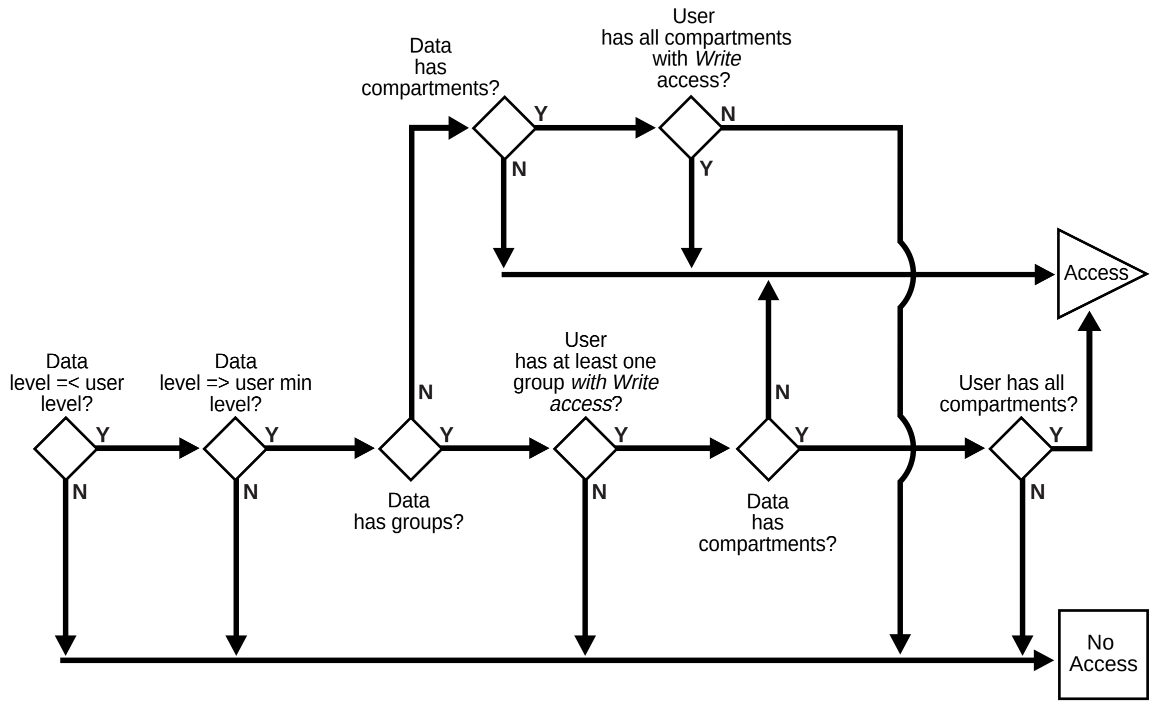
To state tests 2 and 3 another way:

- 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, she only needs 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 her authorized minimum level, nor above her current session level. The user can always read below her minimum level.

The following figure 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 3–8 Label Evaluation Process for Write Access


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

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

1. Is the data’s level equal to, or less than, the level of the user?
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, 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 which 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 only have write access to 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 she cannot write. She cannot update or delete any rows labeled with levels below her minimum, nor can she insert a row with a row label containing a level less than her minimum.

See Also: "Algorithm for Write Access with Inverse Groups" on page 13-11

Using Oracle Label Security Privileges

This section introduces the Oracle Label Security database and row label privileges:

- Privileges Defined by Oracle Label Security Policies
- Special Access Privileges
- Special Row Label Privileges
- System Privileges, Object Privileges, and Policy Privileges

Privileges Defined by Oracle Label Security Policies

Oracle Label Security supports special privileges which allow authorized users to *bypass* certain parts of the policy. Table 3–3 lists the full set of privileges which can be granted to users or trusted stored program units.

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
WRITEUP	Allows users to set or raise only the level, within a row label, up to the maximum level authorized for the user. (With LABEL_UPDATE enforcement.)
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. (With LABEL_UPDATE enforcement.)
WRITEACROSS	Allows a user to set or change groups and compartments of a row label, but does not allow changes to the level. (With LABEL_UPDATE enforcement.)

Special Access Privileges

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

- READ
- FULL
- COMPACCESS
- PROFILE_ACCESS

READ

A user with READ privilege can read all data protected by the policy, regardless of his 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 he or she has write access, based on any label authorizations.

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 SELECTs, since the system behaves as though the Oracle Label Security policy were not even present.

FULL

The FULL privilege has the same effect and benefits as the READ privilege, with one difference: a user with FULL privilege can also *write* to all the data. For a user with the FULL privilege, the READ and WRITE algorithms are not enforced.

COMPACCESS

The COMPACCESS privilege allows a user to access data based on the row's compartments, independent of the row's groups. If a row 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 privileged users to access all of the data for a particular compartment, independent of what groups may own or otherwise be allowed access to the data.

Figure 3–9 shows the label evaluation process for read 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 Read Access with COMPACCESS Privilege

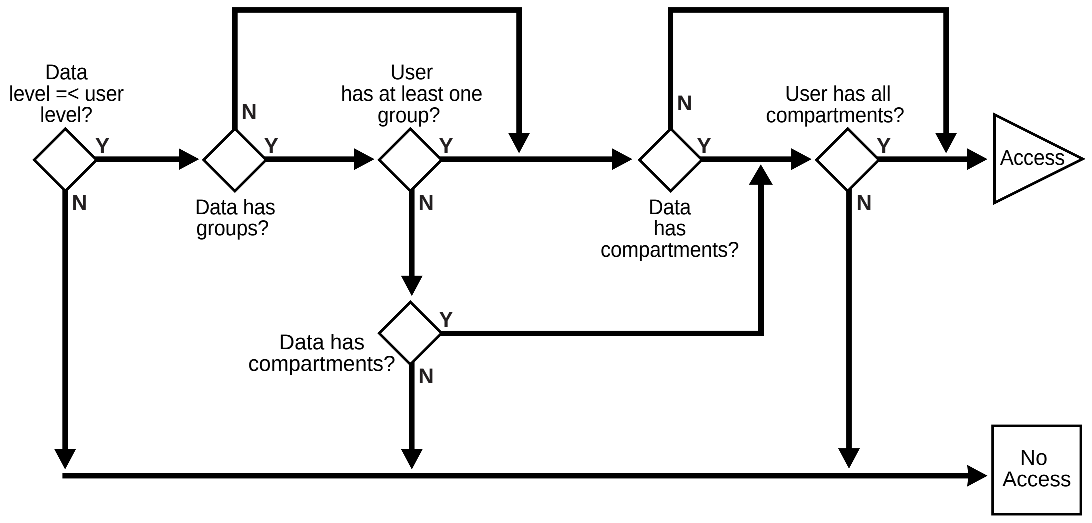
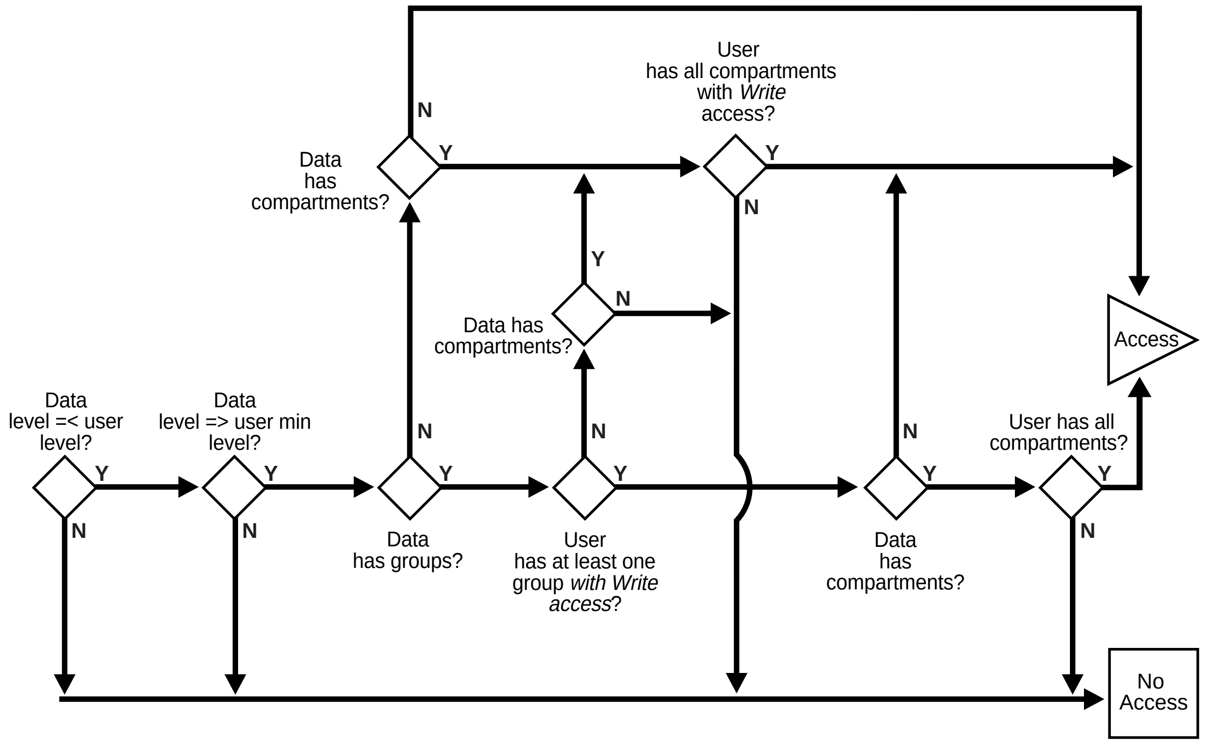


Figure 3–10 shows the label evaluation process for write access with COMPACCESS privilege. Note that if the data label is null or invalid, then the user is denied access.

Figure 3–10 Label Evaluation Process for Write Access with COMPACCESS Privilege


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

PROFILE_ACCESS

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, since the user can potentially become a user with FULL privileges. This privilege cannot be granted to a trusted stored program unit.

Special Row Label Privileges

Once the label on a row has been set, Oracle Label Security privileges are required to modify the label. These privileges include WRITEUP, WRITEDOWN, and WRITEACROSS.

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.

WRITEUP

The WRITEUP privilege enables the user to raise the level of data within a row, without compromising the compartments or groups. The user can raise the level up to his or her maximum authorized level.

For example, an authorized user can raise the level of a data row which has a level lower than his own minimum level. If a row is UNCLASSIFIED and the user's maximum level is SENSITIVE, he can raise the row's level to SENSITIVE. He can raise the level above his current session level, but cannot change the compartments.

WRITEDOWN

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 his or her minimum authorized level.

WRITEACROSS

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.

System Privileges, Object Privileges, and Policy Privileges

Remember that Oracle Label Security privileges are different from the standard Oracle9*i* system and object privileges.

Table 3–4 Types of Privilege

Source	Privileges	Definition
Oracle9*i*	System Privileges	The right to execute a particular type of SQL statement
	Object Privileges	The right to access another user’s object
Oracle Label Security	Policy Privileges	The ability to *bypass certain parts of the label security policy*

Oracle9*i* enforces the discretionary access control privileges which 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 Oracle9*i*, 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.

See Also: For more information about which Oracle9*i* privileges are required to perform a certain operation, and how to grant and revoke these discretionary access control privileges, see *Oracle9i Database Administrator’s Guide*

Access Mediation and Views

Prior to accessing data through a view, end users must have the appropriate system and object privileges on the view. If the underlying table (upon which the view is based) is protected by Oracle Label Security, then the end user of the view must have authorization from Oracle Label Security to access specific rows of labeled data.

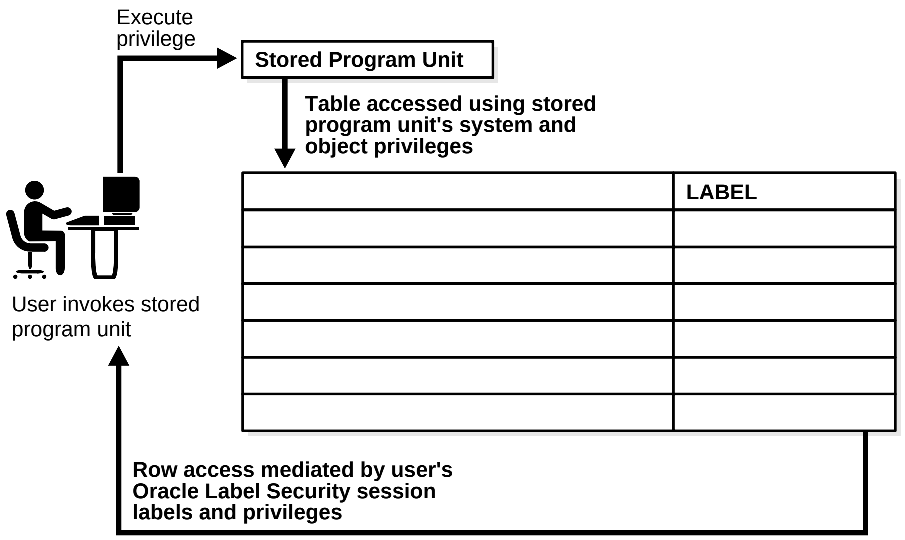
Access Mediation and Program Unit Execution

In Oracle9*i*, if User1 executes a procedure which belongs to User2, the procedure runs with User2's system and object privileges. However, any procedure executed by User1 runs with User1's own Oracle Label Security labels and privileges. This is true even when User1 executes stored program units owned by other users.

Figure 3–11 illustrates this process:

- Stored program units execute 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 3–11 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 invokes it. For example, if you possess no Oracle Label Security privileges in your own right, but execute a stored program unit which has the WRITEDOWN privilege, 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/or functions that have been 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 of the data in a table, not just the data visible to you, you might make use of a trusted program unit set up by an administrator. Program units can thus perform operations on behalf of users, without the need to grant privileges directly to users.

> **See Also:** Chapter 9, "Administering and Using Trusted Stored Program Units"

Access Mediation and Policy Enforcement Options

An administrator can choose among a set of *policy enforcement options* when applying an Oracle Label Security policy to individual tables. These provide mechanisms to tailor the enforcement 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, to further define the rows in the table accessible to the user. In cases where the label associated with a new or updated row should be automatically computed, an administrator can specify a *labeling function* that will always be invoked to provide the data label.

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

> **See Also:** Chapter 7, "Implementing Policy Options and Labeling Functions"

Multiple Oracle Label Security Policies

This section describes aspects of using multiple policies.

Multiple Oracle Label Security Policies in a Single Database

There may be several Oracle Label Security policies protecting data in a single database. Each defined policy is associated with a set of labels that are used only by that policy. Data labels are constrained by the set of defined labels for each policy.

The tables protected may be disjoint, or in some cases a single table may be protected by more than one Oracle Label Security policy. You must have label authorizations for all policies protecting the data you need to access. To access any particular row, you must be authorized by *all* policies protecting the table. If you require privileges, then you may need privileges for all of the policies affecting your work.

Multiple Oracle Label Security Policies in a Distributed Environment

If you work in a distributed environment, where multiple databases may be protected by the same or different Oracle Label Security policies, your remote connections will also be controlled by Oracle Label Security.

See Also: Chapter 11, "Using Oracle Label Security with a Distributed Database"

Part II

Using Oracle Label Security Functionality

4 Working with Labeled Data

This chapter explains how to use Oracle Label Security features to manage labeled data. It then shows how to view and change the value of security attributes for a session. The chapter contains these sections:

- The Policy Label Column and Label Tags
- Presenting the Label
- Filtering Data Using Labels
- Inserting Labeled Data
- Changing Your Session and Row Labels with SA_SESSION

Note: Many of the examples in this book 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.

The Policy Label Column and Label Tags

This section contains these topics:

- The Policy Label Column
- Label Tags

The Policy Label Column

Labels defined in Oracle Label Security have an associated label tag that uniquely identifies the label in the database. The label tag can be manually specified by the administrator at the time the label is created, or it will be automatically generated when the label is created.

The label tag (rather than the character-string label value) is stored in the policy-specific label column that is added when an Oracle Label Security policy is applied to a table. By default, the datatype of the policy label column is NUMBER; it is used to store the numeric label tag.

Hiding the Policy Label Column

The administrator can specify whether or not to display the column. If he or she applies the HIDE option to a table, then the policy label column is not displayed when a user executes SELECT *, or performs a DESCRIBE. If the policy label column is not hidden, then it is displayed as datatype NUMBER.

Example 1: Numeric Column Datatype (NUMBER)

```
SQL> 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)
```

Example 2: Numeric Column Datatype with Hidden Column

Notice that in this example, the HR_LABEL column is *not* displayed.

```
SQL> 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)
```

Label Tags

As noted in Chapter 2, the administrator first defines a set of label components to be used in a policy. When the administrator creates labels, he or she specifies the set of valid combinations of components which can make up a label. For each such valid label, an associated numeric tag uniquely identifies the label within the policy.

The label tag can be manually defined by the administrator, to control the ordering of label values when they are sorted or logically compared. If the tag is not pre-defined before a label is used, then a tag is automatically generated for each label as it is used.

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 the label tag is the unique identifier of a label. The data rows do not store the label's character-string representation; rather, they store the label tag.

This section contains these topics:

- Manually Defining Label Tags to Order Labels
- Manually Defining Label Tags to Manipulate Data
- Automatically Generated Label Tags

Manually Defining Label Tags to Order Labels

By manually defining label tags, the administrator can implement a data manipulation strategy which permits labels to be meaningfully sorted and compared. To do this, the administrator pre-defines 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 8 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 4–1 illustrates a set of label tags which have been assigned by an administrator. Notice that in this example the administrator has based the label tag value on the numeric form of the levels, compartments, and rows which were discussed in Chapter 2 (Table 2–2, Table 2–3, and Table 2–4).

Table 4–1 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.

Manually Defining Label Tags to Manipulate Data

An administratively defined label tag can serve as a convenient way to reference a complete label string (that is, a particular combination of label components). As illustrated in Table 4–1, 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 users from Company Y all 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;
```

Automatically Generated Label Tags

Dynamically generated label tags, illustrated in Table 4–2 , have 10 digits. In this case there is no relationship at all between the label tag and the numbers assigned to the various label components, nor is there any other means of grouping the data by label.

Table 4–2 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

See Also: "Creating a Valid Data Label with SA_LABEL_ADMIN.CREATE_LABEL" on page 5-23

"Planning a Label Tag Strategy to Enhance Performance" on page 12-10

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
- Converting a Label Tag to a Character String, with LABEL_TO_CHAR

Converting a Character String to a Label Tag, with CHAR_TO_LABEL

Use the CHAR_TO_LABEL function to convert a character string to a label tag. This function returns the label tag for the specified character string.

Syntax:

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

Example:

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

Here, "HR" is the label's policy name.

Converting a Label Tag to a Character String, with LABEL_TO_CHAR

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.

Syntax:

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

LABEL_TO_CHAR Examples

Example 1: To retrieve the label of a row from a table or view, specify the policy label column in the SELECT statement as follows:

```
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 2: You can also specify the policy label column in the WHERE clause of a SELECT statement. The following statement displays all rows which 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, you will see the label tag.

Example 3: The following example is with the numeric column datatype (NUMBER) and dynamically generated label tags, but without using the LABEL_TO_CHAR function. If you do not use the LABEL_TO_CHAR function, you will see the label tag.

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

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

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

If the policy label column is hidden, then it is not automatically returned when you select all columns from a table using the SELECT * command. You must explicitly specify that you want to retrieve the label. For example, to retrieve all columns from the DEPT table (including the policy label column in its character representation), enter the following:

```
SQL> column label format a10
SQL> select label_to_char (hr_label) as label, dept.*
  2  from dept;

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.

See Also: "The HIDE Policy Column Option" on page 7-4

Filtering Data Using Labels

During the processing of SQL statements, Oracle Label Security makes calls to the security policies defined in the database. For SELECT statements, the policy filters the data rows which 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.

This section contains these topics:

- Using Numeric Label Tags in WHERE Clauses
- Ordering Labeled Data Rows
- Ordering by Character Representation of Label
- Determining Upper and Lower Bounds of Labels
- Merging Labels with the MERGE_LABEL Function

See Also: "Partitioning Data Based on Numeric Label Tags" on page 12-12

Using Numeric Label Tags in WHERE Clauses

This section describes techniques of using numeric label tags in WHERE clauses of SELECT statements.

When using labels in the NUMBER format, the administrator can set up labels such that a list of their 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 the administrator has 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 by entering:

```
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 only have meaning if the administrator has applied a numeric ordering strategy to the label tags which he or she 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.

See Also: "Using Dominance Functions" on page A-3

Ordering Labeled Data Rows

You can perform an ORDER BY referencing the policy label column to order rows by the numeric label tag value which the administrator has set. For example:

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

Notice that no functions were necessary in this statement. The statement simply made use of label tags set up by the administrator.

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

Ordering by Character Representation of Label

Using the LABEL_TO_CHAR function, you can order data rows by the character representation of the label. For example, the following statement returns all rows sorted by the text order of the label:

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

Determining Upper and Lower Bounds of Labels

This section describes the Oracle Label Security functions which 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.

- Finding Least Upper Bound with LEAST_UBOUND
- Finding Greatest Lower Bound with GREATEST_LBOUND

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

Finding Least Upper Bound with LEAST_UBOUND

The LEAST_UBOUND (LUBD) function returns a character string label that is the least upper bound of *label1* and *label2*: that is, the one label which 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.

Syntax:

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

The 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 his or her department, and returns the higher label—whether it be in the EMP table or the DEPT table.

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

This query returns the following:

```
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
```

Finding Greatest Lower Bound with GREATEST_LBOUND

The GREATEST_LBOUND (GLBD) function can be used to determine the lowest label of the data that can be involved in an operation, given two different labels. It returns a character string label that is the greatest lower bound of *label1* and *label2*. The greatest lower bound is the lowest level, and 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.

Syntax:

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

Merging Labels with the MERGE_LABEL Function

The MERGE_LABEL function is a utility for merging two labels together. It accepts the character string form of two labels, and the three-character specification of a merge format. Its syntax is as follows:

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.

The following table defines the MERGE_LABEL format constants.

Table 4–3 MERGE_LABEL Format Constants

Format Specification	Datatype	Constant	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)
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 comps column, returns no comps. 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 LEAST_UBOUND function does not provide the intended result. The 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 which is provided by the 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 LEAST_UBOUND function is used to compute the least upper bound of these two labels, 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, the resulting label would contain the highest level, the union of the compartments, and no groups—because UNITED_STATES and UNITED_KINGDOM do not intersect.

Inserting Labeled Data

When you insert data into a table protected by Oracle Label Security, you must specify a value for the label in any INSERT statement (unless the LABEL_DEFAULT policy option is set, or a label function exists to compute the label). To do this, you must explicitly specify the label tag value for the desired label, or explicitly convert the character string representation of the label into the appropriate label tag. Note that this does not mean generating label tags, but simply referencing the appropriate one.

This section explains the different ways to insert labeled data:

- Inserting Labels Using CHAR_TO_LABEL
- Inserting Labels Using Numeric Label Tag Values
- Inserting Data Without Specifying a Label
- Inserting Data When the Policy Label Column Is Hidden
- Inserting Labels Using TO_DATA_LABEL

> **See Also:** Chapter 8, "Applying Policies to Tables and Schemas" for information about inserting data with a labeling function, and updating and deleting labeled data

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 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'));
```

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. For example, if the numeric label tag for SENSITIVE is 3000, it would look like this:

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

Inserting Data Without Specifying a Label

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

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

The resulting row label is set according to the default value, or labeling function.

Inserting Data When the Policy Label Column Is Hidden

If the label column is hidden, the existence of the column is transparent to the insertion of data. INSERT statements can be written which do not explicitly list the table columns, and do not include a value for the label column. The session's row label or a label function (if provided) is used to label the data.

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 the table described in "Example 2: Numeric Column Datatype with Hidden Column" on page 4-3:

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

Note that if the policy label column is *not* hidden, you must explicitly include a label value (possibly a null value) in the INSERT statement.

Inserting Labels Using TO_DATA_LABEL

If you are generating new labels dynamically as you insert data, you can use the TO_DATA_LABEL function to guarantee that this produces valid data labels. To do this you must have EXECUTE authority on the TO_DATA_LABEL function.

Whereas the CHAR_TO_LABEL function requires that the label already be an existing *data* label for the transaction to succeed, the TO_DATA_LABEL does not have this requirement. It will automatically create a valid data label.

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.

See Also: Chapter 8, "Applying Policies to Tables and Schemas" for more information about inserting, updating, and deleting labeled data

Changing Your Session and Row Labels with SA_SESSION

During a given session, a user can change his or her labels, within the authorizations set by the administrator.

This section contains these topics:

SA_SESSION Functions to Change Session and Row Labels

The following functions enable the user to change the session and row labels:

Table 4–4 Functions to Change Session Labels

Function	Purpose
SA_SESSION.SET_LABEL	Lets the user set a new level and new compartments and groups to which he or she has read access
SA_SESSION.SET_ROW_LABEL	Lets the user set the default row label that will be applied to new rows
SA_SESSION.RESTORE_DEFAULT_LABELS	Lets the user reset the current session label and row label to the stored default settings
SA_SESSION.SAVE_DEFAULT_LABELS	Lets the user store the current session label and row label as the default for future sessions

Changing the Session Label with SA_SESSION.SET_LABEL

Use the SET_LABEL procedure to set the label of the current database session.

Syntax:

```
PROCEDURE SET_LABEL (policy_name IN VARCHAR2,
                     label IN VARCHAR2);
```

policy_name	The name of an existing policy
label	The value to set as the label

A user can set the session label to:

- Any level equal to or less than his maximum, and equal to or greater than his minimum level
- Include any compartments in his authorized compartment list
- Include any groups in his 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 command 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.

See Also: "SA_USER_ADMIN.SET_DEFAULT_LABEL" on page 6-13

Changing the Row Label with SA_SESSION.SET_ROW_LABEL

Use the SET_ROW_LABEL procedure to set the default 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. When the LABEL_DEFAULT option is set, this row label value is used on insert if the user does not explicitly specify the label.

Syntax:

```
PROCEDURE SET_ROW_LABEL (policy_name IN VARCHAR2,
                         row_label IN VARCHAR2);
```

policy_name	The name of an existing policy
row_label	The value to set as the default row 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 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, 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 which 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, the operation is not permitted, and the row label value is unchanged.

See Also: "SA_USER_ADMIN.SET_ROW_LABEL" on page 6-14

Restoring Label Defaults with SA_SESSION.RESTORE_DEFAULT_LABELS

The 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 executed.

Syntax:

```
PROCEDURE RESTORE_DEFAULT_LABELS (policy_name in VARCHAR2);
```

policy_name	The name of an existing policy

Saving Label Defaults with SA_SESSION.SAVE_DEFAULT_LABELS

The SAVE_DEFAULT_LABELS procedure stores the current session label and row label as your initial session label and default row label. It 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.

Syntax:

```
PROCEDURE SAVE_DEFAULT_LABELS (policy_name in VARCHAR2);
```

policy_name	The name of an existing policy

When you log into 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, he or she 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 invoke Oracle Reports, you ensure that Oracle Reports will initialize at the same labels as are being used by Oracle Forms.

Note: The SA_SESSION.SAVE_DEFAULT_LABELS procedure overrides the settings established by the administrator.

Viewing Session Attributes with SA_SESSION Functions

You can use SA_SESSION functions to view the policy attributes for a session.

- USER_SA_SESSION View to Return All Security Attributes
- Functions to Return Individual Security Attributes

USER_SA_SESSION View to Return All Security Attributes

You can display security attribute values by using the USER_SA_SESSION view. Access to this view is PUBLIC. It lets you see the security attributes for your current session. For example:

```
Name                                      Null?    Type
----------------------------------------- -------- -------------
POLICY_NAME                               NOT NULL VARCHAR2(30)
SA_USER_NAME                                       VARCHAR2(4000)
PRIVS                                              VARCHAR2(4000)
MAX_READ_LABEL                                     VARCHAR2(4000)
MAX_WRITE_LABEL                                    VARCHAR2(4000)
MIN_LEVEL                                          VARCHAR2(4000)
LABEL                                              VARCHAR2(4000)
COMP_WRITE                                         VARCHAR2(4000)
GROUP_WRITE                                        VARCHAR2(4000)
ROW_LABEL                                          VARCHAR2(4000)
```

Functions to Return Individual Security Attributes

The SA_SESSION functions take a *policy_name* as the only input parameter. They return VARCHAR2 character string values for use in SQL statements.

Table 4–5 SA_SESSION Functions to View Security Attributes

Function	Purpose
SA_SESSION.PRIVS	Returns the set of current session privileges, in a comma-separated list
SA_SESSION.MIN_LEVEL	Returns the minimum level authorized for the session
SA_SESSION.MAX_LEVEL	Returns the maximum level authorized for the session
SA_SESSION.COMP_READ	Returns a comma-separated list of compartments which the user is authorized to read
SA_SESSION.COMP_WRITE	Returns a comma-separated list of compartments which the user is authorized to write. This is a subset of SA_SESSION.COMP_READ.
SA_SESSION.GROUP_READ	Returns a comma-separated list of groups which the user is authorized to read
SA_SESSION.GROUP_WRITE	Returns a comma-separated list of groups which the user is authorized to write. This is a subset of SA_SESSION.GROUP_READ.
SA_SESSION.LABEL	Returns the session label (the level, compartments, and groups) with which the user is currently working. The user can change this value.
SA_SESSION.ROW_LABEL	Returns the session’s default row label value. The user can change this value.
SA_SESSION.SA_USER_NAME	Returns the username associated with the current Oracle Label Security session

For example, the following statement shows the current session label for the Human Resources policy:

```
SQL> select sa_session.label ('human_resources')
  2  from dual;

SA_SESSIONs.LABEL('HUMAN_RESOURCES')
---------------------------------------------
L3:M,E
```

See Also: "Using SA_UTL Functions to Set and Return Label Information" on page 9-7 for additional functions that return numeric label tags and BOOLEAN values

Part III

Administering an Oracle Label Security Application

5 Creating an Oracle Label Security Policy

This chapter explains how to create an Oracle Label Security policy. It contains these sections:

Oracle Label Security Administrative Task Overview

To create and implement an Oracle Label Security policy, you perform the following tasks, which are described in the next few chapters:

- Step 1: Create the Policy
- Step 2: Define the Components of the Labels
- Step 3: Identify the Set of Valid Data Labels
- Step 4: Apply the Policy to Tables and Schemas
- Step 5: Authorize Users
- Step 6: Create and Authorize Trusted Program Units (Optional)
- Step 7: Configure Auditing (Optional)

Step 1: Create the Policy

Create a policy by defining:

- The policy name
- The column name for policy labels
- The default options for the policy

To do this in Oracle Policy Manager, you can use the Create Policy icon or the Policy property sheet.

Alternatively, you can use the SA_SYSDBA.CREATE_POLICY command line procedure.

See Also: "Creating a Policy with SA_SYSDBA.CREATE_POLICY" on page 5-9

Step 2: Define the Components of the Labels

Define the levels, compartments, and groups which form the components of the new policy's labels.

To do this in Oracle Policy Manager, go to **Oracle Label Security Policies—>** *policyname*—>**Labels** and use the Labels property sheet.

Alternatively, you can use the SA_COMPONENTS package on the command line.

See Also: "Using the SA_COMPONENTS Package to Define Label Components" on page 5-12

Step 3: Identify the Set of Valid Data Labels

Specify the set of valid labels to support the policy. From all the possible combinations of levels, compartments, and groups, you must define labels which can be assigned to data.

Alternatively, applications that need to create data labels dynamically at runtime can use the TO_DATA_LABEL function.

To do this in Oracle Policy Manager, go to **Oracle Label Security Policies—>** *policyname***—>Labels** and use the Labels property sheet.

See Also: "Using the SA_LABEL_ADMIN Package to Specify Valid Labels" on page 5-22

"Inserting Labels Using TO_DATA_LABEL" on page 4-18

Step 4: Apply the Policy to Tables and Schemas

Protect individual database tables and schemas by applying the policy to them. In the process, you can customize the level of enforcement of the policy for each table and schema, to reflect your application security requirements.

To do this with Oracle Policy Manager, go to **Oracle Label Security Policies—>** *policyname***—>Protected Objects.** Select either Schemas or Tables, and use the corresponding property sheet.

Alternatively, you can use the SA_POLICY_ADMIN package.

See Also: Chapter 8, "Applying Policies to Tables and Schemas"

Step 5: Authorize Users

For individual users, define the authorizations which each person will use for session access. If users do not have appropriate authorizations, they cannot access protected data.

You can optionally assign special privileges which particular users need to do their job. Note that Oracle Label Security privileges may only be necessary to perform special job functions.

To do this with Oracle Policy Manager, go to **Oracle Label Security Policies—>** *policyname***—>Authorizations—>Users** and use the User property sheet.

Alternatively, you can use the SA_POLICY_ADMIN package.

See Also: Chapter 6, "Administering User Labels and Privileges"

Step 6: Create and Authorize Trusted Program Units (Optional)

Create any necessary stored trusted program units, and set their labels and privileges.

To do this with Oracle Policy Manager, go to **Oracle Label Security Policies—>** *policyname***—>Authorizations—>Program Units** and use the User property sheet.

Alternatively, you can use the SA_USER_ADMIN package.

> **See Also:** Chapter 9, "Administering and Using Trusted Stored Program Units"

Step 7: Configure Auditing (Optional)

Configure monitoring of the administrative tasks and use of privileges, if desired.

- Configure policy-wide auditing.

  To do this with Oracle Policy Manager, go to **Oracle Label Security Policies—>** *policyname***—>Auditing** and use the Auditing tab page of the Policy property sheet.

- Configure auditing on a user-by-user basis.

  To do this with Oracle Policy Manager, go to **Oracle Label Security Policies—>Authorizations—>Users—>** *username*. Use the Auditing tab page of the User property sheet.

Alternatively, you can use the SA_AUDIT_ADMIN package to set auditing options for policies, users, and program units.

> **See Also:** Chapter 10, "Auditing Under Oracle Label Security"

Organizing the Duties of Oracle Label Security Administrators

You can manage the administration of an Oracle Label Security policy in various ways. The *policy*_DBA role is created when you create a new policy, and every individual who needs to perform administrative functions must be granted this role. However, you can grant EXECUTE privileges on the administrative packages to different users, so that each administrator can be restricted to a subset of the administrative functions.

For example, you could grant EXECUTE privilege on SA_COMPONENTS and SA_LABEL_ADMIN to one user or role to manage the label definitions, and grant EXECUTE on SA_USER_ADMIN to a different user or role to manage user labels and privileges. Alternatively, you could grant EXECUTE on all of the administrative packages to the *policy*_DBA role, so that anyone with the *policy*_DBA role could perform all of the administrative tasks.

Choosing an Oracle Label Security Administrative Interface

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

- Oracle Label Security Packages
- Oracle Policy Manager

Oracle Label Security Packages

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

Table 5–1 Oracle Label Security Administrative Packages

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

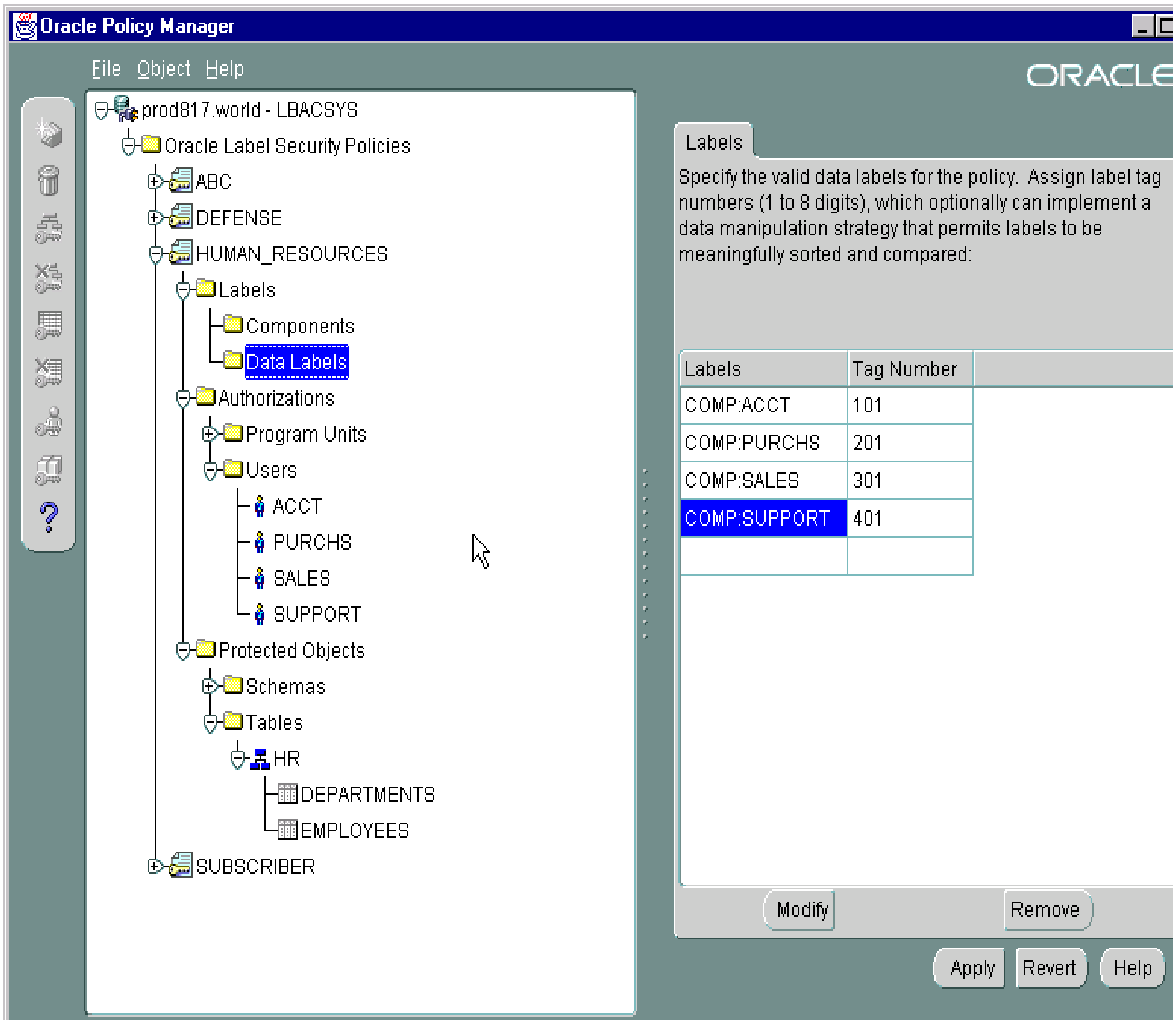
Oracle Label Security Demonstration File

For a demonstration showing how to create and develop an Oracle Label Security policy using the supplied packages, refer to the `demobld.sql` file in your `ORACLE_HOME/lbac/demo` directory.

Oracle Policy Manager

You can use Oracle Policy Manager, an extension to Oracle Enterprise Manager, to administer Oracle Label Security. Figure 5–1 is a representative screenshot which illustrates the Oracle Policy Manager interface. Please see the online help for instructions on how to use this graphical user interface.

Figure 5–1 Oracle Policy Manager Interface



- prod817.world - LBACSYS
  - Oracle Label Security Policies
    - ABC
    - DEFENSE
    - HUMAN_RESOURCES
      - Labels
        - Components
        - Data Labels
      - Authorizations
        - Program Units
        - Users
          - ACCT
          - PURCHS
          - SALES
          - SUPPORT
      - Protected Objects
        - Schemas
        - Tables
          - HR
            - DEPARTMENTS
            - EMPLOYEES
    - SUBSCRIBER

Specify the valid data labels for the policy. Assign label tag numbers (1 to 8 digits), which optionally can implement a data manipulation strategy that permits labels to be meaningfully sorted and compared:

Labels	Tag Number
COMP:ACCT	101
COMP:PURCHS	201
COMP:SALES	301
COMP:SUPPORT	401

Using the SA_SYSDBA Package to Manage Security Policies

This section explains how to manage a policy using the SA_SYSDBA package. To do this in Oracle Policy Manager, use the Create Policy icon or the Policy property sheet.

Who Can Use the SA_SYSDBA Package

To use the SA_SYSDBA package to create, alter, and drop policies a user must have:

- The LBAC_DBA role
- EXECUTE privilege on the SA_SYSDBA package

Who Can Administer a Policy

When you create a policy, a role named *policy*_DBA is automatically created. You can use this role to control the users who are authorized to execute the policy's administrative procedures.

For example, after you have created a human resources policy named HR, an HR_DBA role is automatically created. To use any administrative packages, a user would need to have the HR_DBA role. If Joan is the administrator of the HR policy, and David is the administrator of the FIN policy, then Joan has the HR_DBA role and David has the FIN_DBA role. Each person can only administer the policy for which he or she has the *policy*_DBA role.

The user who creates the policy is automatically granted the *policy*_DBA role with the ADMIN option, and can grant the role to others.

Valid Characters for Policy Specifications

Valid characters for all policy specifications include alphanumeric characters and underscores, as well as any valid character from your database character set.

Creating a Policy with SA_SYSDBA.CREATE_POLICY

Use the CREATE_POLICY procedure to create a new Oracle Label Security policy, define a policy-specific column name, and specify a set of default policy options.

Syntax:

```
PROCEDURE CREATE_POLICY (
   policy_name       IN VARCHAR2,
   column_name       IN VARCHAR2 DEFAULT NULL,
   default_options   IN VARCHAR2 DEFAULT NULL);
```

policy_name	Specifies the policy name, which must be unique within the database. It can have a maximum of 30 characters.
column_name	Specifies the name of the column to be added to tables protected by the policy. If NULL, the default name "SA_LABEL" 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.

See Also: "Choosing Policy Options" on page 7-2

"SYSDBA.CREATE_POLICY with Inverse Groups" on page 13-21

Modifying Policy Options with SA_SYSDBA.ALTER_POLICY

Use the ALTER_POLICY procedure to set and modify policy default options.

Syntax:

```
PROCEDURE ALTER_POLICY (
   policy_name       IN  VARCHAR2,
   default_options   IN  VARCHAR2 DEFAULT NULL);
```

policy_name	Specifies the policy 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.

Disabling a Policy with SA_SYSDBA.DISABLE_POLICY

Use the DISABLE_POLICY procedure to turn 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.

Syntax:

```
PROCEDURE DISABLE_POLICY (policy_name IN VARCHAR2);
```

policy_name	Specifies the policy to be 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. Your site therefore 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 in order 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 re-enable the policy, all of the selected enforcement options become effective again.

Enabling a Policy with SA_SYSDBA.ENABLE_POLICY

Use the ENABLE_POLICY procedure to enforce access control on the tables and schemas protected by the policy. A policy is automatically enabled when it is created. After creation or enabling, the policy is enforced for all subsequent access to tables protected by the policy

Syntax:

```
PROCEDURE ENABLE_POLICY (policy_name IN VARCHAR2);
```

policy_name	Specifies the policy to be enabled

Removing a Policy with SA_SYSDBA.DROP_POLICY

Use the DROP_POLICY procedure to remove the policy and all of its associated user labels and data labels from the database. It purges the policy from the system entirely. You can optionally drop the label column from all tables controlled by the policy.

Syntax:

```
PROCEDURE DROP_POLICY (policy_name IN VARCHAR2,
   drop_column  BOOLEAN DEFAULT FALSE);
```

policy_name	Specifies the policy to be dropped
drop_column	Indicates that the policy column should be dropped from protected tables (TRUE)

Using the SA_COMPONENTS Package to Define Label Components

This package manages the component definitions of an Oracle Label Security label. Each policy defines the components differently. This section contains these topics:

- Creating a Level with SA_COMPONENTS.CREATE_LEVEL
- Modifying a Level with SA_COMPONENTS.ALTER_LEVEL
- Removing a Level with SA_COMPONENTS.DROP_LEVEL
- Creating a Compartment with SA_COMPONENTS.CREATE_COMPARTMENT
- Modifying a Compartment with SA_COMPONENTS.ALTER_
  COMPARTMENT
- Removing a Compartment with SA_COMPONENTS.DROP_COMPARTMENT
- Creating a Group with SA_COMPONENTS.CREATE_GROUP
- Modifying a Group with SA_COMPONENTS.ALTER_GROUP
- Modifying a Group Parent with SA_COMPONENTS.ALTER_GROUP_PARENT
- Removing a Group with SA_COMPONENTS.DROP_GROUP

See Also: Chapter 2, "Understanding Data Labels and User Labels" for information about the components

"Using Oracle Label Security Views" on page 6-17 for information about displaying the label definitions you have set

Using Overloaded Procedures

Oracle Label Security makes use of overloaded subprogram names. That is, the same name is used for several different procedures whose formal parameters differ in number, order, or datatype family.

For example, you can call the SA_COMPONENTS.ALTER_LEVEL procedure this way:

```
PROCEDURE ALTER_LEVEL (policy_name IN VARCHAR2,
   level_num       IN INTEGER,
   new_short_name  IN VARCHAR2 DEFAULT NULL,
   new_long_name   IN VARCHAR2 DEFAULT NULL);
```

or this way:

```
PROCEDURE ALTER_LEVEL (policy_name IN VARCHAR2,
   short_name      IN VARCHAR2,
   new_long_name   IN VARCHAR2);
```

Because the processing in these two procedures is the same, it is logical to give them the same name. PL/SQL determines which of the two procedures is being called by checking their formal parameters. In the preceding example, the version of `initialize` used by PL/SQL depends on whether you call the procedure with a `level_num` or `short_name` parameter.

Creating a Level with SA_COMPONENTS.CREATE_LEVEL

Use the CREATE_LEVEL procedure to create a level and specify its short name and long name. The numeric values assigned to the *level_num* determine the sensitivity ranking (that is, a lower number indicates less sensitive data).

Syntax:

```
PROCEDURE CREATE_LEVEL (policy_name IN VARCHAR2,
   level_num        IN INTEGER,
   short_name       IN VARCHAR2,
   long_name        IN VARCHAR2);
```

policy_name	Specifies the policy
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)

Modifying a Level with SA_COMPONENTS.ALTER_LEVEL

Use the ALTER_LEVEL procedure to change the short name and / or 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:

```
PROCEDURE ALTER_LEVEL (policy_name IN VARCHAR2,
   level_num       IN INTEGER,
   new_short_name  IN VARCHAR2 DEFAULT NULL,
   new_long_name   IN VARCHAR2 DEFAULT NULL);

PROCEDURE ALTER_LEVEL (policy_name IN VARCHAR2,
   short_name      IN VARCHAR2,
   new_long_name   IN VARCHAR2);
```

policy_name	Specifies the policy
level_num	Specifies the number of the level to be altered
short_name	Specifies the short name for the level (up to 30 characters)
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)

Removing a Level with SA_COMPONENTS.DROP_LEVEL

Use the DROP_LEVEL procedure to remove a level. If the level is used in any existing label, it cannot be dropped.

Syntax:

```
PROCEDURE DROP_LEVEL (policy_name IN VARCHAR2,
   level_num   IN INTEGER);

PROCEDURE DROP_LEVEL (policy_name IN VARCHAR2,
   short_name  IN VARCHAR2);
```

policy_name	Specifies the policy
level_num	Specifies the number of an existing level for the policy
short_name	Specifies the short name for the level (up to 30 characters)

Creating a Compartment with SA_COMPONENTS.CREATE_COMPARTMENT

Use the CREATE_COMPARTMENT procedure to create a compartment and specify its short name and long name. The *comp_num* determines the order in which compartments are listed in the character string representation of labels.

Syntax:

```
PROCEDURE CREATE_COMPARTMENT (policy_name IN VARCHAR2,
   comp_num    IN INTEGER,
   short_name  IN VARCHAR2,
   long_name   IN VARCHAR2);
```

policy_name	Specifies the policy
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)

Modifying a Compartment with SA_COMPONENTS.ALTER_COMPARTMENT

Use the ALTER_COMPARTMENT procedure to change the short name and / or long name associated with a compartment.

Once set, the *comp_num* cannot be changed. If the *comp_num* is used in any existing label, then its short name *cannot* be changed, but its long name *can* be changed.

Syntax:

```
PROCEDURE ALTER_COMPARTMENT (policy_name IN VARCHAR2,
   comp_num          IN INTEGER,
   new_short_name    IN VARCHAR2 DEFAULT NULL,
   new_long_name     IN VARCHAR2 DEFAULT NULL);

PROCEDURE ALTER_COMPARTMENT (policy_name IN VARCHAR2,
   short_name        IN VARCHAR2,
   new_long_name     IN VARCHAR2);
```

policy_name	Specifies the policy
comp_num	Specifies the number of the compartment to be altered
short_name	Specifies the short name of the compartment to be altered (up to 30 characters)
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).

Removing a Compartment with SA_COMPONENTS.DROP_COMPARTMENT

Use the DROP_COMPARTMENT procedure to remove a compartment. If the compartment is used in any existing label, it cannot be dropped.

Syntax:

```
PROCEDURE DROP_COMPARTMENT (policy_name IN VARCHAR2,
   comp_num    IN INTEGER);

PROCEDURE DROP_COMPARTMENT (policy_name IN VARCHAR2,
   short_name  IN VARCHAR2);
```

policy_name	Specifies the policy
comp_num	Specifies the number of an existing compartment for the policy
short_name	Specifies the short name of an existing compartment for the policy

Creating a Group with SA_COMPONENTS.CREATE_GROUP

Use the CREATE_GROUP procedure to create a group and specify its short name and long name, and optionally a parent group.

Syntax:

```
PROCEDURE CREATE_GROUP (policy_name IN VARCHAR2,
   group_num   IN INTEGER,
   short_name  IN VARCHAR2,
   long_name   IN VARCHAR2,
   parent_name IN VARCHAR2 DEFAULT NULL);
```

policy_name	Specifies the policy
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, 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.

See Also: "Groups" on page 2-8

Modifying a Group with SA_COMPONENTS.ALTER_GROUP

Use the ALTER_GROUP procedure to change the short name and / or long name associated with a group.

Once set, the *group_num* 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:

```
PROCEDURE ALTER_GROUP (policy_name IN VARCHAR2,
   group_num      IN INTEGER,
   new_short_name IN VARCHAR2 DEFAULT NULL,
   new_long_name  IN VARCHAR2 DEFAULT NULL);

PROCEDURE ALTER_GROUP (policy_name IN VARCHAR2,
   short_name     IN VARCHAR2,
   new_long_name  IN VARCHAR2);
```

policy_name	Specifies the policy
group_num	Specifies the existing group number to be altered
short_name	Specifies the existing group short name to be altered
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)

Modifying a Group Parent with SA_COMPONENTS.ALTER_GROUP_PARENT

The ALTER_GROUP_PARENT procedure changes the parent group associated with a particular group.

Syntax:

```
PROCEDURE ALTER_GROUP_PARENT (policy_name IN VARCHAR2,
   group_num   IN INTEGER,
   parent_name IN VARCHAR2);

PROCEDURE ALTER_GROUP_PARENT (policy_name IN VARCHAR2,
   group_num   IN INTEGER,
   parent_num  IN INTEGER);

PROCEDURE ALTER_GROUP_PARENT (policy_name IN VARCHAR2,
   short_name  IN VARCHAR2,
   parent_name IN VARCHAR2);
```

policy_name	Specifies the policy
group_num	Specifies the existing group number to be altered
short_name	Specifies the existing group short name to be altered
parent_num	Specifies the number of an existing group as the parent group
parent_name	Specifies the short name of an existing group as the parent group

Removing a Group with SA_COMPONENTS.DROP_GROUP

Use the DROP_GROUP procedure to remove a group. If the group is used in existing labels, it cannot be dropped.

Syntax:

```
PROCEDURE DROP_GROUP (policy_name IN VARCHAR2,
   group_num   IN INTEGER);

PROCEDURE DROP_GROUP (policy_name IN VARCHAR2,
   short_name  IN VARCHAR2);
```

policy_name	Specifies the policy
group_num	Specifies the number of an existing group for the policy
short_name	Specifies the short name of an existing group

Using the SA_LABEL_ADMIN Package to Specify Valid Labels

The SA_LABEL_ADMIN package provides an administrative interface to manage the labels used by a policy. To do this, a user must have EXECUTE privilege for the SA_LABEL_ADMIN package and have been granted the *policy*_DBA role.

This section includes:

- Creating a Valid Data Label with SA_LABEL_ADMIN.CREATE_LABEL
- Modifying a Label with SA_LABEL_ADMIN.ALTER_LABEL
- Deleting a Label with SA_LABEL_ADMIN.DROP_LABEL

Creating a Valid Data Label with SA_LABEL_ADMIN.CREATE_LABEL

Use the SA_LABEL_ADMIN.CREATE_LABEL procedure to create a valid data label. You must manually specify a label tag value from 1 to 8 digits long.

Syntax:

```
PROCEDURE CREATE_LABEL (
   policy_name IN VARCHAR2,
   label_tag   IN INTEGER,
   label_value IN VARCHAR2,
   data_label  IN BOOLEAN DEFAULT TRUE);
```

policy_name	Specifies the name of an existing policy
label_tag	Specifies an unique integer value representing the sort order of the label, relative to other policy labels (0-99999999)
label_value	Specifies the character string representation of the label to be created
data_label	TRUE if the label can be used to label row data. Use this to define the label as valid for data.

When specifying labels, use the short name of the level, compartment and group.

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.

Note: If you create a new label by using the TO_DATA_LABEL procedure, a system-generated label tag of 10 digits will be generated automatically.

See Also: "The Policy Label Column and Label Tags" on page 4-2

Modifying a Label with SA_LABEL_ADMIN.ALTER_LABEL

Use the ALTER_LABEL procedure to change the character string label definition associated with a label tag. Note that the label tag itself cannot be changed.

If you change the character string associated with a label tag, 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 though 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.

Note 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:

```
PROCEDURE ALTER_LABEL (
   policy_name      IN VARCHAR2,
   label_tag        IN INTEGER,
   new_label_value  IN VARCHAR2 DEFAULT NULL,
   new_data_label   IN BOOLEAN  DEFAULT NULL);

PROCEDURE ALTER_LABEL (
   policy_name      IN VARCHAR2,
   label_value      IN VARCHAR2,
   new_label_value  IN VARCHAR2 DEFAULT NULL,
   new_data_label   IN BOOLEAN  DEFAULT NULL);
```

policy_name	Specifies the name of an existing policy
label_tag	Identifies the integer tag assigned to the label to be altered
label_value	Identifies the existing character-string representation of the label to be altered
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.

Deleting a Label with SA_LABEL_ADMIN.DROP_LABEL

Use the SA_LABEL_ADMIN.DROP_LABEL procedure to delete 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.

Syntax:

```
PROCEDURE DROP_LABEL (
   policy_name       IN VARCHAR2,
   label_tag         IN INTEGER);

PROCEDURE DROP_LABEL (
   policy_name       IN VARCHAR2,
   label_value       IN VARCHAR2);
```

policy_name	Specifies the name of an existing policy
label_tag	Specifies the integer tag assigned to the label to be dropped
label_value	Specifies the string value of the label to be dropped

Caution: Do not drop a label which is in use anywhere in the database.

Use this procedure only while setting up labels, prior to data population. If you should inadvertently drop a label which is being used, you can recover by disabling the policy, fixing the problem, and then re-enabling the policy.

6
Administering User Labels and Privileges

In Oracle Label Security, you can set authorizations for users, and grant privileges to users or stored program units by means of the available Oracle Label Security packages, or Oracle Policy Manager.

Introduction to User Label and Privilege Management

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

To perform these functions with Oracle Policy Manager, go to **Oracle Label Security Policies—>** *policyname***—>Authorizations—>Users** and use the User property sheet.

The SA_USER_ADMIN package provides the 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, his 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.

Managing User Labels by Component, with SA_USER_ADMIN

The following SA_USER_ADMIN procedures enable you to manage user labels by label component:

- SA_USER_ADMIN.SET_LEVELS
- SA_USER_ADMIN.SET_COMPARTMENTS
- SA_USER_ADMIN.SET_GROUPS
- SA_USER_ADMIN.ADD_COMPARTMENTS
- SA_USER_ADMIN.ALTER_COMPARTMENTS
- SA_USER_ADMIN.DROP_COMPARTMENTS
- SA_USER_ADMIN.DROP_ALL_COMPARTMENTS
- SA_USER_ADMIN.ADD_GROUPS
- SA_USER_ADMIN.ALTER_GROUPS
- SA_USER_ADMIN.DROP_GROUPS
- SA_USER_ADMIN.DROP_ALL_GROUPS

SA_USER_ADMIN.SET_LEVELS

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

- If the *min_level* is NULL, it is set to the lowest defined level for the policy.
- If the *def_level* is not specified, it is set to the *max_level*.
- If the *row_level* is not specified, it is set to the *def_level*.

Syntax:

```
PROCEDURE 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);
```

policy_name	Specifies the policy
user_name	Specifies the user name
max_level	The highest level for read and write access
min_level	The lowest level for write access
def_level	Specifies the default level (equal to or greater than the minimum level, and equal to or less than the maximum level)
row_level	Specifies the row level (equal to or greater than the minimum level, and equal to or less than the default level)

SA_USER_ADMIN.SET_COMPARTMENTS

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

- If *write_comps* are NULL, they are set to the *read_comps*.
- If the *def_comps* are NULL, they are set to the *read_comps*.
- If the *row_comps* are NULL, they are set to the components in *def_comps* which are authorized for write access.

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:

```
PROCEDURE 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);
```

policy_name	Specifies the policy
user_name	Specifies the user name
read_comps	A comma-separated list of compartments authorized for read access
write_comps	A comma-separated list of compartments authorized for write access (subset of *read_comps*)
def_comps	Specifies the default compartments. This must be a subset of *read_comps*.
row_comps	Specifies the row compartments. This must be a subset of *write_comps* and the *def_comps*.

SA_USER_ADMIN.SET_GROUPS

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

- If the *write_groups* are NULL, they are set to the *read_groups*.
- If the *def_groups* are NULL, they are set to the *read_groups*.
- If the *row_groups* are NULL, they are set to the groups in *def_groups* which are authorized for write access.

All users must have their levels set before their authorized groups can be established.

Syntax:

```
PROCEDURE 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);
```

policy_name	Specifies the policy
user_name	Specifies the user name
read_groups	A comma-separated list of groups authorized for read
write_groups	A comma-separated list of groups authorized for write. This must be a subset of *read_groups*.
def_groups	Specifies the default groups. This must be a subset of *read_groups*.
row_groups	Specifies the row groups. This must be a subset of *write_groups* and *def_groups*.

SA_USER_ADMIN.ALTER_COMPARTMENTS

The ALTER_COMPARTMENTS procedure changes the write access, the default label indicator, and/or the row label indicator for each of the compartments in the list.

Syntax:

```
PROCEDURE 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);
```

policy_name	Specifies the policy
user_name	Specifies the user name
comps	A comma-separated list of compartments to modify
access_mode	One of two public variables which contain string values that can specify the type of access authorized. The variable names, values, and meaning are as follows:
	SA_UTL.READ_ONLY READ_ONLY Indicates no write access
	SA_UTL.READ_WRITE READ_WRITE Indicates write is authorized
in_def	Specifies whether these compartments should be in the default compartments (Y/N)
in_row	Specifies whether these compartments should be in the row label (Y/N)

SA_USER_ADMIN.ADD_COMPARTMENTS

This procedure adds compartments to a user's authorizations, indicating whether the compartments are authorized for write as well as read.

Syntax:

```
PROCEDURE ADD_COMPARTMENTS (policy_name IN VARCHAR2,
user_name      IN VARCHAR2,
comps          IN VARCHAR2,
access_model   IN VARCHAR2 DEFAULT NULL,
in_def         IN VARCHAR2 DEFAULT NULL,
in_row         IN VARCHAR2 DEFAULT NULL);
```

policy_name	Specifies the policy
user_name	Specifies the user name
comps	A comma-separated list of read compartments to add
access_mode	One of two public variables which contain string values that can specify the type of access authorized. The variable names, values, and meaning are as follows: SA_UTL.READ_ONLY READ_ONLY Indicates no write access SA_UTL.READ_WRITE READ_WRITE Indicates write is authorized
in_def	Specifies whether these compartments should be in the default compartments (Y/N)
in_row	Specifies whether these compartments should be in the row label (Y/N)

SA_USER_ADMIN.DROP_COMPARTMENTS

The DROP_COMPARTMENTS procedure drops the specified compartments from a user's authorizations.

Syntax:

```
PROCEDURE DROP_COMPARTMENTS (policy_name IN VARCHAR2,
  user_name       IN VARCHAR2,
  comps           IN VARCHAR2);
```

policy_name	Specifies the policy
user_name	Specifies the user name
comps	A comma-separated list of compartments to drop

SA_USER_ADMIN.DROP_ALL_COMPARTMENTS

The DROP_ALL_COMPARTMENTS procedure drops all compartments from a user's authorizations.

Syntax:

```
PROCEDURE DROP_ALL_COMPARTMENTS (policy_name IN VARCHAR2,
     user_name     IN VARCHAR2);
```

policy_name	Specifies the policy
user_name	Specifies the user name

SA_USER_ADMIN.ADD_GROUPS

The ADD_GROUPS procedure adds groups to a user, indicating whether the groups are authorized for write as well as read.

Syntax:

```
PROCEDURE 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);
```

policy_name	Specifies the policy
user_name	Specifies the user name
groups	A comma-separated list of read groups to add
access_mode	One of two public variables which contain string values that can specify the type of access authorized. The variable names, values, and meaning are as follows:

SA_UTL.READ_ONLY	READ_ONLY	Indicates no write access
SA_UTL.READ_WRITE	READ_WRITE	Indicates write is authorized

in_def	Specifies whether these groups should be in the default groups (Y/N)
in_row	Specifies whether these groups should be in the row label (Y/N)

SA_USER_ADMIN.ALTER_GROUPS

The ALTER_GROUPS procedure changes the write access, the default label indicator, and/or the row label indicator for each of the groups in the list.

Syntax:

```
PROCEDURE 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);
```

policy_name	Specifies the policy
user_name	Specifies the user name
groups	A comma-separated list of groups to alter
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	READ_ONLY	Indicates no write access
SA_UTL.READ_WRITE	READ_WRITE	Indicates write is authorized

in_def	Specifies whether these groups should be in the default groups (Y/N)
in_row	Specifies whether these groups should be in the row label (Y/N)

SA_USER_ADMIN.DROP_GROUPS

The DROP_GROUPS procedure drops the specified groups from a user's authorizations.

Syntax:

```
PROCEDURE DROP_GROUPS (policy_name IN VARCHAR2,
  user_name   IN VARCHAR2,
  groups      IN VARCHAR2);
```

policy_name	Specifies the policy
user_name	Specifies the user name
groups	A comma-separated list of groups to drop

SA_USER_ADMIN.DROP_ALL_GROUPS

The DROP_ALL_GROUPS procedure drops all groups from a user's authorizations.

Syntax:

```
PROCEDURE DROP_ALL_GROUPS (policy_name IN VARCHAR2,
  user_name  IN VARCHAR2);
```

policy_name	Specifies the policy
user_name	Specifies the user name

Managing User Labels by Label String, with SA_USER_ADMIN

The following SA_USER_ADMIN procedures enable you to manage user labels by specifying the complete character label string:

- SA_USER_ADMIN.SET_USER_LABELS
- SA_USER_ADMIN.SET_DEFAULT_LABEL
- SA_USER_ADMIN.SET_ROW_LABEL
- SA_USER_ADMIN.SET_DEFAULT_LABEL

SA_USER_ADMIN.SET_USER_LABELS

The SET_USER_LABELS procedure sets the user's levels, compartments, and groups using a set of labels, instead of the individual components.

Syntax:

```
PROCEDURE 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);
```

policy_name	Specifies the policy
user_name	Specifies the user name
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.
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 the *max_write_label* is not specified, it is set to the *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 the *min_write_label* is not specified, 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 the *default_label* is not specified, it is set to the *max_read_label*.
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, it is set to the *def_label*, with only the compartments and groups authorized for write access.

See Also: "Managing Program Unit Privileges with SET_PROG_PRIVS" on page 9-4

SA_USER_ADMIN.SET_DEFAULT_LABEL

The SET_DEFAULT_LABEL procedure sets the user's initial session label to the one specified.

Syntax:

```
PROCEDURE SET_DEFAULT_LABELS (
  policy_name  IN VARCHAR2,
  user_name    IN VARCHAR2,
  def_label    IN VARCHAR2);
```

policy_name	Specifies the policy
user_name	Specifies the user name
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.

As long as the row label will still be dominated by the new write label, the user can set the session label to:

- Any level equal to or less than his maximum, and equal to or greater than his 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 which will result from resetting the session label. If this condition is not true, 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 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 which will be dominated by the resulting session label.

See Also: "Changing the Session Label with SA_SESSION.SET_LABEL" on page 4-20

"Session Labels and Inverse Groups" on page 13-16

SA_USER_ADMIN.SET_ROW_LABEL

Use the SET_ROW_LABEL procedure to set the user's initial row label to the one specified.

Syntax:

```
PROCEDURE SET_ROW_LABEL (
  policy_name   IN VARCHAR2,
  user_name     IN VARCHAR2,
  row_label     IN VARCHAR2);
```

policy_name	Specifies the policy
user_name	Specifies the user name
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.

The user can set the row label independently, but only to:

- A level which 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, the operation is disallowed, and the row label value is unchanged.

See Also: "Changing the Row Label with SA_SESSION.SET_ROW_LABEL" on page 4-21

SA_USER_ADMIN.DROP_USER_ACCESS

Use the DROP_USER_ACCESS procedure to remove all Oracle Label Security authorizations and privileges from the specified user. This procedure must be issued from the command line. It is not available in Oracle Policy Manager.

Syntax:

```
PROCEDURE DROP_USER_ACCESS (
  policy_name      IN VARCHAR2,
  user_name        IN VARCHAR2);
```

policy_name	Specifies the policy
user_name	Specifies the user name

Managing User Privileges with SA_USER_ADMIN.SET_USER_PRIVS

The SET_USER_PRIVS procedure sets policy-specific privileges for users. These privileges do not become effective in the current session; rather, they become effective the next time the user logs in. 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 EXECUTE privilege for the SA_ USER_ADMIN package, and must have been granted the *policy*_DBA role.

To use Oracle Policy Manager to perform these functions, go to the Privileges tab of the User property sheet.

Syntax:

```
PROCEDURE SET_USER_PRIVS (
  policy_name     IN VARCHAR2,
  user_name       IN VARCHAR2,
  privileges      IN VARCHAR2);
```

policy_name	Specifies the policy name of an existing policy
user_name	The name of the user to be granted privileges
privileges	A character string of policy-specific privileges separated by commas

See Also: "Managing Program Unit Privileges with SET_PROG_ PRIVS" on page 9-4

Setting Labels & Privileges with SA_SESSION.SET_ACCESS_PROFILE

The 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 executing the SA_SESSION.SET_ACCESS_PROFILE procedure must have the PROFILE_ACCESS privilege. Note that the logged-in database user (the Oracle userid) 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 SET_ACCESS_PROFILE, the administrator 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 SET_ACCESS_PROFILE privilege, the proxy account can act on behalf of the application users.

Syntax:

```
PROCEDURE SET_ACCESS_PROFILE (policy_name IN VARCHAR2
  user_name   IN VARCHAR2);
```

policy_name	The name of an existing policy
user_name	Name of the user whose authorizations and privileges should be assumed

Returning User Name with SA_SESSION.SA_USER_NAME

The SA_USER_NAME function returns the name of the current Oracle Label Security user, as set by the 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:

```
FUNCTION SA_USER_NAME (policy_name IN VARCHAR2)
RETURN VARCHAR2;
```

policy_name	The name of an existing policy

Using Oracle Label Security Views

This section describes views you can use to see the user authorization and privilege assignments made by the administrator.

Note that the views are designed to display these values from two different perspectives. The DBA_SA_USERS view is optimized for users of the command-line interface. The component views are optimized for users of the Oracle Policy Manager administrative tool.

- View to Display All User Security Attributes: DBA_SA_USERS
- Views to Display User Authorizations by Component

View to Display All User Security Attributes: DBA_SA_USERS

The DBA_SA_USERS view displays the values assigned for privileges, level, compartments, and groups all together—corresponding to the way in which you enter these values through the SA_USER_ADMIN command-line interface. The values include:

USER_PRIVILEGES

MAX_READ_LABEL

MAX_WRITE_LABEL

MIN_WRITE_LABEL

DEFAULT_READ_LABEL

DEFAULT_WRITE_LABEL

DEFAULT_ROW_LABEL

USER_LABELS

> MAX_READ_LABEL
>
> MAX_WRITE_LABEL
>
> MIN_WRITE_LABEL
>
> DEFAULT_READ_LABEL
>
> DEFAULT_WRITE_LABEL
>
> DEFAULT_ROW_LABEL

This information is stored in data dictionary tables, and used to establish session and row labels when a user logs in.

Note: The field USER_LABELS in DBA_SA_USERS is retained solely for backward compatibility and will be removed in the next release.

Views to Display User Authorizations by Component

The following views display individually each component of the label, corresponding to the way in which you enter these values through Oracle Policy Manager.

Table 6–1 Oracle Label Security Views

View	Contents
DBA_SA_USER_LEVELS	Displays the levels assigned to the user: minimum level, maximum level, default level, and level for the row label
DBA_SA_USER_COMPARTMENTS	Displays the compartments assigned to the user
DBA_SA_USER_GROUPS	Displays the groups assigned to the user

7 Implementing Policy Options and Labeling Functions

This chapter explains how to customize the enforcement of Oracle Label Security policies, and how to implement labeling functions.

This chapter contains these sections:

Choosing Policy Options

This section introduces the policy options, and discusses their use.

Overview of Policy Enforcement Options

Of all the enforcement controls which Oracle Label Security permits, the administrator must choose those which meet the needs of the given application. These options can operate at three different levels:

- Policy level
- Schema level
- Table level

For each policy, you can specify a set of default enforcement options that will automatically be used whenever the policy is applied to a table. Alternatively, you can explicitly specify enforcement options by schema or by table.

When you apply a policy to a schema or a table, you can customize its enforcement in accordance with your security requirements, and specify whether the label column should be displayed or hidden. You can chose to enforce some or all of the policy options for any protected table.

Optionally, you can assign each table a *labeling function* and / or a *SQL predicate*.

The Oracle Label Security policy enforcement options are as follows:

Table 7–1 Policy Enforcement Options

Option	Description
LABEL_DEFAULT	If the user does not explicitly specify a label on INSERT, the session's default row label value is used.
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 only enforced if the LABEL_UPDATE option is set.
CHECK_CONTROL	Applies READ_CONTROL policy enforcement to INSERT and UPDATE statements to assure that the new row label is read-accessible.
READ_CONTROL	Applies policy enforcement to all queries; only authorized rows are accessible for SELECT, UPDATE, and DELETE operations.
WRITE_CONTROL	Determines the ability to INSERT, UPDATE, and DELETE data in a row. If this option is set, 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 Figure 3–8.
DELETE_CONTROL	Applies policy enforcement to DELETE operations, according to the algorithm for write access described in Figure 3–8.
UPDATE_CONTROL	Applies policy enforcement to UPDATE operations on the data columns within a row, according to the algorithm for write access described in Figure 3–8.
ALL_CONTROL	Applies all enforcement options.
NO_CONTROL	Applies no enforcement options. A labeling function or a SQL predicate can nonetheless be applied.

Remember: although Oracle Label Security may be applied to a table, not all DML operations will necessarily be governed by the policy. Depending on how the administrator has set the policy enforcement options, there will be differences in 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 set. This means that all enforcement options are in effect. If a user attempts to perform an operation for which he or she is not authorized, an error message is raised and the SQL statement fails.

See Also: "Implementing Inverse Groups with the INVERSE_GROUP Enforcement Option" on page 13-4

The HIDE Policy Column Option

HIDE is a configuration option which you can specify when initially applying an Oracle Label Security policy to a table (that is, when adding the policy column to the table). When HIDE is specified, the column which contains the policy's labels is not displayed.

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.

For INSERT statements, the values for the hidden label columns are not required for all-column inserts. For SELECT statements, the values of hidden label columns are not automatically returned; they must be explicitly retrieved.

Furthermore, the label column may or may not be displayed when you perform a DESCRIBE on the table, depending on how the administrator has set the HIDE option. With the HIDE option off, the label column is displayed in response to a select.

See Also: "Retrieving All Columns from a Table When Policy Label Column Is Hidden" on page 4-9

The Label Management Enforcement Options

This section describes the label enforcement options.

LABEL_DEFAULT: Using the Session's Default Row Label

If set, this option specifies that the user's session default row label should be used on insert as the new row label, if the user does not explicitly specify a value. Note that if a labeling function is in force on the table, it will override the LABEL_DEFAULT option.

LABEL_UPDATE: Changing Data Labels

If this option is not set, any user can change a row's label, within the range of his or her authorizations. If LABEL_UPDATE is set, then the user must have the WRITEUP, WRITEDOWN, and/or WRITEACROSS privilege in order to modify a label.

The LABEL_UPDATE option uses an Oracle after-row trigger. It is only invoked on an update operation which changes the value of the policy label column. Note that any labeling function which is in force on a table will override the LABEL_UPDATE option.

CHECK_CONTROL: Checking Data Labels

After an INSERT or UPDATE, the READ_CONTROL option is enforced on the new label to assure that the user is authorized for read access. In other words, CHECK_CONTROL ensures that the user who modifies a label on a row can still access the row after the operation.

The Access Control Enforcement Options

This section describes the options related to access control:

- READ_CONTROL: Reading Data
- WRITE_CONTROL: Writing Data
- INSERT_CONTROL, UPDATE_CONTROL, and DELETE_CONTROL

READ_CONTROL: Reading Data

The READ_CONTROL option uses Oracle virtual private database (VPD) technology to enforce the read access mediation algorithm, as illustrated in Figure 3–7 on page 3-13.

READ_CONTROL determines the set of records accessible to a session for SELECT, UPDATE and DELETE operations. If READ_CONTROL is off, then all rows in the table protected by the policy are accessible to all users.

WRITE_CONTROL: Writing Data

The WRITE_CONTROL option uses Oracle after-row triggers to enforce the write access mediation algorithm, as illustrated in Figure 3–8 on page 3-16. When an Oracle Label Security policy is applied to a table and the WRITE_CONTROL option is selected, triggers are generated and the algorithm is enforced.

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 separately in INSERT, UPDATE, and DELETE operations using the corresponding policy enforcement option (INSERT_CONTROL, UPDATE_CONTROL, and DELETE_CONTROL).

If WRITE_CONTROL is on but LABEL_UPDATE is not specified, the user can change both data and labels. If you want to control updating the row labels, use the LABEL_UPDATE option in addition to WRITE_CONTROL.

INSERT_CONTROL, UPDATE_CONTROL, and DELETE_CONTROL

These options apply policy enforcement during the corresponding operations on the data columns within a row, according to the algorithm for write access described in Figure 3–8.

The Overriding Enforcement Options

Whereas ALL_CONTROL applies all of the label management and access control enforcement options, NO_CONTROL applies none of them. Labeling functions and SQL predicates can nonetheless be applied. Note that the ALL_CONTROL option can be used on the command line only. Oracle Policy Manager does not provide this as an alternative to selecting individual options.

If you apply a policy with NO_CONTROL specified, a policy label column is added to the table, but the label values are NULL. Since 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 wish.

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.

Guidelines for Using the Policy Enforcement Options

You can customize policy enforcement for a schema or table through the Oracle Policy Manager, or by using the SA_POLICY_ADMIN package. This section documents the supported keywords.

Note that you can specify a string of default options for the policy; these are used if no schema or table options are specified.

If a policy is applied to a table, and subsequently applied to the schema containing that table, the options on the table are not affected. The table retains its original options.

In general, administrators will use the LABEL_DEFAULT policy option. This causes the user's row label to be used to label data. Alternatively, a labeling function can be used to label the data. If neither of these two options is used, a label would have to be specified in every INSERT statement.

The following table suggests certain combinations of policy enforcement options which you may find useful when implementing your Oracle Label Security policy. As the table illustrates, you might typically enforce READ_CONTROL and WRITE_CONTROL, and take one or another approach to setting the label.

Table 7–2 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.
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, but users cannot change labels without privileges. Add CHECK_CONTROL to restrict new labels (on insert or update) to visible range.

Exemptions from Oracle Label Security Policy Enforcement

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 Oracle9i EXEMPT ACCESS POLICY privilege, either directly or through a database role, are completely exempt from Oracle Label Security enforcement—regardless of the export mode, application or utility used to extract data from the database. This is a very powerful privilege and should be carefully managed.

Note that this privilege does not affect the enforcement of standard Oracle9*i* 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.

Viewing Policy Options on Tables and Schemas

You can use the following views to see the policy enforcement options which are currently applied to tables and schemas:

- DBA_SA_TABLE_POLICIES
- DBA_SA_SCHEMA_POLICIES

Using a Labeling Function

Application developers can create labeling functions, programs which contain procedural logic to compute and return a label. The function can use a wide array of resources to compute the label. These include context variables (such as date or username) and data values.

This section describes how to use labeling functions.

- Approaches to Data Labeling
- How Labeling Functions Work
- Creating a Labeling Function
- Specifying a Labeling Function

Approaches to Data Labeling

There are three ways to label data which is being inserted:

- Explicitly specify a label in every INSERT into the table.
- 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 statement.
- Create a labeling function which will be invoked upon every INSERT statement, and which operates 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, Oracle Label Security embeds a call to that function in INSERT and UPDATE triggers to compute a label.

For example, the following labeling function uses the contents of COL1 and COL2 of the new row to compute and return the appropriate label for the row.

```
my_label(:new.col1,:new.col2)
```

If you do not specify a labeling function, then you should use the LABEL_DEFAULT option. Otherwise, you must explicitly specify a label on every INSERT statement.

How Labeling Functions Work

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 is invalid, 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 datatype. The TO_LBAC_DATA_LABEL function can be used to convert a label in character string format to a datatype of LBACSYS.LBAC_LABEL. Note that LBACSYS must have EXECUTE privilege on your labeling function. The owner of the labeling function must have EXECUTE privilege on the TO_LBAC_DATA_LABEL function, with GRANT option.

Creating a Labeling Function

The following example shows how to create a labeling function.

```
SQL> 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;
    /
```

Specifying a Labeling Function

The following example shows how to specify a labeling function.

```
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);
```

Policy Options and Labeling Functions: Inserting Labeled Data

This section explains how enforcement options and labeling functions affect the insertion of labeled data.

- Enforcement Control Options and INSERT
- Inserting Labels When a Labeling Function is Specified
- Inserting Child Rows into Tables with Declarative Referential Integrity Enabled

Enforcement Control Options and INSERT

When you attempt to insert or update data based on your authorizations, the outcome depends upon the way in which the policy enforcement controls are set.

- If INSERT_CONTROL is set, then you can only insert rows labeled within your write authorizations. If you attempt to update data which 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, the statement will fail.
- If INSERT_CONTROL is *not* set, you can insert with any valid label.
- If the CHECK_CONTROL option is set, you can only insert rows with labels which you are authorized to read—even if the labels are generated by a labeling function.

Inserting 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 you enter will have effect (unless the label function enables your label to be considered). New row labels are always determined by the label function.

Note that the security administrator can establish a labeling function which sets the label of a row being inserted to a value outside the range which the user can see. If this is the case, the user can potentially insert a row, but not be authorized to see the row. You can prevent this situation from occurring by using the CHECK_ CONTROL option. This option verifies the user’s read authorization on the new label; if this option is on, she will not be able to perform such an insert.

Inserting Child Rows into Tables with Declarative Referential Integrity Enabled

Declarative referential integrity can enforce parent-child relationships between tables, if the parent is a protected table. If a child row is in a table which has a referential integrity constraint, then the parent row must be visible (the user must be able to see the parent row) for the insert to succeed. Thus, the user must have read access to the parent row.

If the parent table has READ_CONTROL on, 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, cannot insert child rows for department 20. (Note that all records will be visible if the user has FULL or READ privilege.)

Policy Options and Labeling Functions: Updating Labeled Data

Update behavior in Oracle Label Security is similar to that of insert behavior. There is nothing different, as long as the user is authorized to change the rows in question. This section contains these topics:

- Updating Labels Using CHAR_TO_LABEL
- Enforcement Control Options and UPDATE
- Updating Labels When a Labeling Function Is Specified
- Updating Child Rows in Tables with Declarative Referential Integrity Enabled

Updating Labels Using CHAR_TO_LABEL

If you need to change a row's label from SENSITIVE to CONFIDENTIAL, you can change the label as follows:

```
UPDATE emp
SET hr_label = char_to_label ('HR', 'CONFIDENTIAL')
WHERE ename = 'ESTANTON';
```

Enforcement Control Options and UPDATE

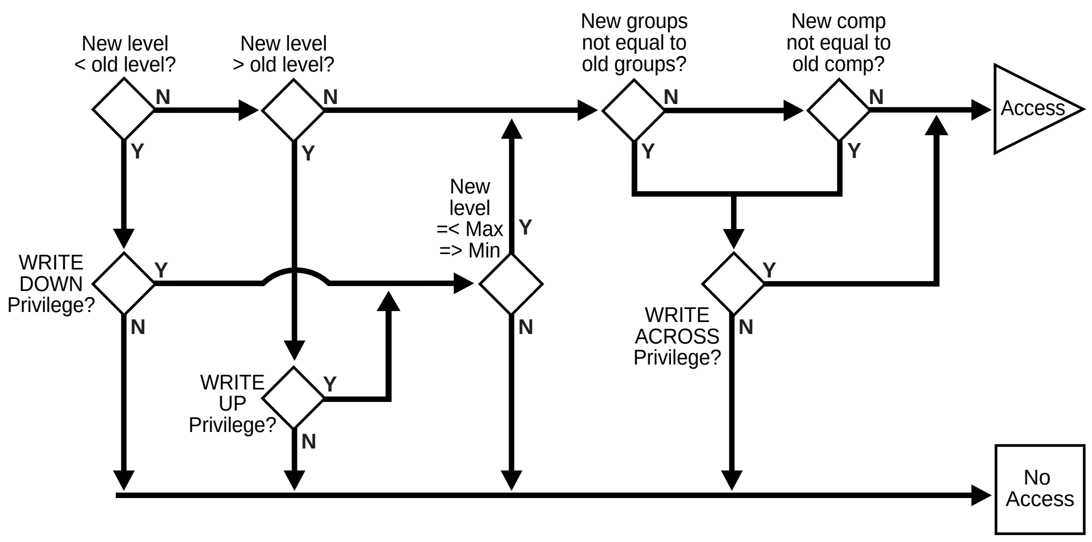
When you attempt to update data based on your authorizations, the outcome depends upon the way in which enforcement controls are set.

- If UPDATE_CONTROL is set, then you can only update rows labeled within your write authorizations. If you attempt to update data which you can read, but for which you do not have write authorization, 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, the statement will fail.

- If UPDATE_CONTROL is not set, you can update all rows to which you have read access.
- If LABEL_UPDATE is set, you must have the appropriate privilege (WRITEUP, WRITEDOWN, or WRITEACROSS) in order to update a label.
- If the CHECK_CONTROL option is set, you can only specify labels which you are authorized to read.
- If LABEL_UPDATE is *not* set but UPDATE_CONTROL *is* set, then you can update a label to any new label value within your write authorization.

The following figure illustrates the label evaluation process for LABEL_UPDATE.

Figure 7–1 Label Evaluation Process for LABEL_UPDATE



Updating 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 you enter will have effect (unless the label function permits your label to be considered). New row labels are always determined by the label function.

Note that the security administrator can establish a labeling function which sets the label of a row being updated to a value outside the range which you can see. If this is the case, you can update a row, but not be authorized to see the row. If the CHECK_CONTROL option is on, you will not be able to perform such an update. CHECK_CONTROL verifies your read authorization on the new label.

Updating Child Rows in Tables with Declarative Referential Integrity Enabled

If a child row is in a table which has a referential integrity constraint, then the parent row must be visible (the user must be able to see the parent row) for the update to succeed. If the parent table has READ_CONTROL on, 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: *Oracle9i Application Developer's Guide - Fundamentals*

Policy Options and Labeling Functions: Deleting Labeled Data

This section covers the deletion of labeled data.

- If the DELETE_CONTROL option is set, you can only delete rows within your write authorization.
- If the DELETE_CONTROL is *not* set, then you can only delete rows which you can read.
- With DELETE_CONTROL set, and declarative referential integrity defined with cascading deletes, then 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 set 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 DELETE_RESTRICT is set. DELETE_ RESTRICT 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, he still cannot delete the parent row.

See Also: *Oracle9i Application Developer's Guide - Fundamentals*:

Using a SQL Predicate with an Oracle Label Security Policy

You can use a SQL predicate to provide extensibility for selective enforcement of data access rules.

This section contains these topics:

- SQL Predicates Used with an Oracle Label Security Policy
- Effect of Multiple SQL Predicates Under Oracle Label Security

SQL Predicates Used with an Oracle Label Security Policy

A SQL predicate is a condition, with an optional preceding AND or OR. It 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 affects the labels which a user can read. It therefore affects the labels and data which 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. (That is, her user label would have to dominate the label on the employee's row). A SQL predicate like the following could be added, such that an employee could bypass label-based filtering if he wanted to view his 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 his or her own record.

You can use such a predicate in conjunction with READ_CONTROL, 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, 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.

Effect of Multiple SQL Predicates Under Oracle Label Security

A predicate applied to a table by means of an Oracle Label Security policy is appended to any other predicates which may be applied by other Oracle Label Security policies, or by Oracle fine grain access control / VPD policies. The predicates are ANDed 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 ANDed 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
```

8

Applying Policies to Tables and Schemas

This chapter describes the SA_POLICY_ADMIN package, which enables you to administer policies on tables and schemas. It contains these sections:

Policy Administration Terminology

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 labels remain in the table unless you explicitly drop them.

You can *alter* the default policy enforcement options for future tables which 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 *re-apply* the policy to the table.

See Also: "Choosing Policy Options" on page 7-2

Policy Administration Functions for Tables and Schemas

Two sets of functions are available to administer Oracle Label Security policies:

- functions to administer policies at the table level
- functions to administer policies at the schema level

Schema-level functions are provided for convenience. Note, however, that administrative operations which you perform at the table level will override operations performed at the schema level.

Table 8–1 Policy Administration Functions

Purpose	Table-Level Function	Level Function
Apply policy	APPLY_TABLE_POLICY	APPLY_SCHEMA_POLICY
Alter policy	Not applicable	ALTER_SCHEMA_POLICY
Disable policy	DISABLE_TABLE_POLICY	DISABLE_SCHEMA_POLICY
Re-enable policy	ENABLE_TABLE_POLICY	ENABLE_SCHEMA_POLICY
Remove policy	REMOVE_TABLE_POLICY	REMOVE_SCHEMA_POLICY

To perform these functions with Oracle Policy Manager, go to **Oracle Label Security Policies—>** *policyname***—>Protected Objects.** Select either Schemas or Tables, and use the corresponding property sheet.

Note: You should restrict access to application tables when using Oracle Policy Manager to change enforcement options. This is because Oracle Policy Manager must remove the policy in order to make such changes, and then re-apply the policy after the changes have been made.

Administering Policies on Tables Using SA_POLICY_ADMIN

To administer policies on tables, a user must have EXECUTE privilege for the SA_POLICY_ADMIN package, and must have been granted the *policy*_DBA role. Authorized users can also perform these functions with the Oracle Policy Manager. This section contains these topics:

- Applying a Policy with SA_POLICY_ADMIN.APPLY_TABLE_POLICY
- Removing a Policy with SA_POLICY_ADMIN.REMOVE_TABLE_POLICY
- Disabling a Policy with SA_POLICY_ADMIN.DISABLE_TABLE_POLICY
- Re-enabling a Policy with SA_POLICY_ADMIN.ENABLE_TABLE_POLICY

Applying a Policy with SA_POLICY_ADMIN.APPLY_TABLE_POLICY

Use the APPLY_TABLE_POLICY procedure to add 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, then re-apply it.

Syntax:

```
PROCEDURE 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);
```

policy_name	Specifies an existing policy
schema_name	Specifies the schema which contains the table
table_name	Specifies the table to be controlled by the policy
table_options	A comma-separated 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 invoking 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	Specifies an additional predicate to combine (using AND or OR) with the label-based predicate for READ_CONTROL

Example:

The following statement applies the HUMAN_RESOURCES policy to the EMP table in the SA_DEMO schema.

```
SA_POLICY_ADMIN.APPLY_TABLE_POLICY('human_resources',
    'sa_demo','emp','no_control');
```

Removing a Policy with SA_POLICY_ADMIN.REMOVE_TABLE_POLICY

The 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:

```
PROCEDURE REMOVE_TABLE_POLICY (
policy_name       IN VARCHAR2,
schema_name       IN VARCHAR2,
table_name        IN VARCHAR2,
  drop_column     IN BOOLEAN DEFAULT FALSE);
```

policy_name	Specifies an existing policy
schema_name	Specifies the schema which contains the table
table_name	Specifies the table
drop_column	If TRUE, the policy's column will be dropped from the table. Otherwise, the column will remain.

Example:

The following statement removes the HUMAN_RESOURCES policy from the EMP table in the SA_DEMO schema:

```
SA_POLICY_ADMIN.REMOVE_TABLE_POLICY('human_resources','sa_demo','emp');
```

Disabling a Policy with SA_POLICY_ADMIN.DISABLE_TABLE_POLICY

The DISABLE_TABLE_POLICY procedure disables the enforcement of the policy for the specified table without changing the enforcement options, labeling function, or predicate values. It removes the RLS predicate and DML triggers from the table.

Syntax:

```
PROCEDURE DISABLE_TABLE_POLICY (
  policy_name      IN VARCHAR2,
  schema_name      IN VARCHAR2,
  table_name       IN VARCHAR2);
```

policy_name	Specifies an existing policy
schema_name	Specifies the schema which contains the table
table_name	Specifies the table

Example:

The following statement disables the HUMAN_RESOURCES policy on the EMP table in the SA_DEMO schema:

```
SA_POLICY_ADMIN.DISABLE_TABLE_POLICY('human_resources','sa_demo','emp');
```

Re-enabling a Policy with SA_POLICY_ADMIN.ENABLE_TABLE_POLICY

The ENABLE_TABLE_POLICY procedure re-enables the current enforcement options, labeling function, and predicate for the specified table by re-applying the RLS predicate and DML triggers.

Syntax:

```
PROCEDURE ENABLE_TABLE_POLICY (
  policy_name     IN VARCHAR2,
  schema_name     IN VARCHAR2,
  table_name      IN VARCHAR2);
```

policy_name	Specifies an existing policy
schema_name	Specifies the schema which contains the table
table_name	Specifies the table

Example:

The following statement re-enables the HUMAN_RESOURCES policy on the EMP table in the SA_DEMO schema:

```
SA_POLICY_ADMIN.ENABLE_TABLE_POLICY('human_resources','sa_demo','emp');
```

Administering Policies on Schemas with SA_POLICY_ADMIN

To administer policies on schemas, a user must have EXECUTE privilege on the SA_POLICY_ADMIN package, and must have been granted the *policy*_DBA role. Authorized users can also use the Oracle Policy Manager to perform these functions.

This section contains these topics:

Applying a Policy with SA_POLICY_ADMIN.APPLY_SCHEMA_POLICY

In addition to applying a policy to individual tables, you can apply a policy at the schema level. The APPLY_SCHEMA_POLICY procedure applies the specified policy to all of the existing tables in a schema (that is, to those which do not already have the policy applied) and enables the policy for these tables. 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 which already have the policy applied.

Syntax:

```
PROCEDURE APPLY_SCHEMA_POLICY (
  policy_name        IN VARCHAR2,
  schema_name        IN VARCHAR2,
  default_options    IN VARCHAR2 DEFAULT NULL);
```

policy_name	Specifies an existing policy.
schema_name	Specifies the schema name to control with the policy.
default_options	The default options to be used for tables in the schema.

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.

Altering Enforcement Options: SA_POLICY_ADMIN.ALTER_SCHEMA_POLICY

The 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; existing tables in the schema are not affected.

Syntax:

```
PROCEDURE ALTER_SCHEMA_POLICY (
  policy_name        IN VARCHAR2,
  schema_name        IN VARCHAR2,
  default_options    IN VARCHAR2);
```

policy_name	Specifies an existing policy.
schema_name	Specifies the schema name to control with the policy.
default_options	The default options to be used for new tables created in the schema.

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 re-apply the policy.

If you alter the enforcement options on a schema, 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.

Removing a Policy with SA_POLICY_ADMIN.REMOVE_SCHEMA_POLICY

The REMOVE_SCHEMA_POLICY procedure removes the specified policy from a schema. The policy will be removed from all of the tables in the schema and, optionally, the label column for the policy will be dropped from all of the tables.

Syntax:

```
PROCEDURE REMOVE_SCHEMA_POLICY (
  policy_name     IN VARCHAR2,
  schema_name     IN VARCHAR2,
  drop_column     IN BOOLEAN DEFAULT FALSE);
```

policy_name	Specifies an existing policy
schema_name	Specifies the schema name
drop_column	If TRUE, the policy's column will be dropped from the tables; otherwise, the column will remain.

Disabling a Policy with SA_POLICY_ADMIN.DISABLE_SCHEMA_POLICY

The DISABLE_SCHEMA_POLICY procedure disables the enforcement of the policy for all of the tables in the specified schema, without changing the enforcement options, labeling function, or predicate values. It removes the RLS predicate and DML triggers from all the tables in the schema.

Syntax:

```
PROCEDURE DISABLE_SCHEMA_POLICY (
  policy_name    IN VARCHAR2,
  schema_name    IN VARCHAR2);
```

policy_name	Specifies an existing policy
schema_name	Specifies the schema name containing the table

Re-Enabling a Policy with SA_POLICY_ADMIN.ENABLE_SCHEMA_POLICY

The ENABLE_SCHEMA_POLICY procedure re-enables the current enforcement optiofns, labeling function, and predicate for the tables in the specified schema by re-applying the RLS predicate and DML triggers.

Syntax:

```
PROCEDURE ENABLE_TABLE_POLICY (
  policy_name    IN VARCHAR2,
  schema_name    IN VARCHAR2);
```

policy_name	Specifies an existing policy
schema_name	Specifies the schema name containing the table

This is the same as enabling a policy for a table, but it covers all tables in the schema.

Policy Issues for Schemas

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 execute SA_POLICY_ADMIN.ENABLE_SCHEMA_POLICY, the table will remain unprotected. If you wish to protect the table once again, you must apply the policy to the table, or re-apply the policy to the schema.

If you apply a policy to a schema which 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 keep the options they already had.

9 Administering and Using Trusted Stored Program Units

This chapter explains how to use trusted stored program units to enhance system security. It contains these topics:

Introduction to Trusted Stored Program Units

Oracle9*i* *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 just like any other stored program units in Oracle9*i*: 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 which 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; doing so may 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, 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 which can be verified to make sure that it is valid.

How a Trusted Stored Program Unit Executes

A trusted stored program unit executes using its own privileges, and the invoker's labels. It can thus perform privileged operations on the set of rows constrained by the user's labels.

Oracle9*i* 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.

Trusted Stored Program Unit Example

A trusted stored program unit with READ privilege can read all unprotected data, and all data protected by this policy in the database. Consider, for example, a user who is responsible for creating purchasing forecast reports. She must perform a summation operation on the amount of all purchases—regardless of whether or not her 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
DECLARE
  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 execute SUM_PURCHASES, the user would need to be granted the standard Oracle9*i* EXECUTE object privilege upon this procedure.

See Also: Chapter 3, "Understanding Access Controls and Privileges"

Managing Program Unit Privileges with SET_PROG_PRIVS

To grant privileges to a stored program unit, you must have the *policy*_DBA role, and EXECUTE permission on the SA_USER_ADMIN package. You can use either the SA_USER_ADMIN package or the Oracle Policy Manager to manage Oracle Label Security privileges.

Use the SA_USER_ADMIN.SET_PROG_PRIVS procedure to set policy-specific privileges for program units. If the *privileges* parameter is NULL, the program unit's privileges for the policy are removed.

Syntax:

```
PROCEDURE SET_PROG_PRIVS (
  policy_name          IN VARCHAR2,
  schema_name          IN VARCHAR2,
  program_unit_name    IN VARCHAR2,
  privileges           IN VARCHAR2);
```

policy_name	Specifies the policy name of an existing policy.
schema_name	Specifies the schema containing the program unit
program_unit_name	Specifies the program unit to be granted privileges
privileges	A comma-separated character string of policy-specific privileges

For example, to give READ privilege to the SUM_PURCHASES function (described in "Trusted Stored Program Unit Example" on page 9-3), you could enter:

```
EXECUTE sa_user_admin.set_prog_privs (
'HR','myschema','sum_purchases','READ');
```

When the SUM_PURCHASES procedure is then called, it executes 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 corporate payroll, without learning what salary any individual employee receives.

Warning: When you create a trusted stored program unit, have the Oracle Label Security administrator review it carefully and evaluate the privileges you are granting to it. Ensure, for example, that procedures in trusted packages do 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.

Creating and Compiling Trusted Stored Program Units

This section contains these topics:

- Creating Trusted Stored Program Units
- Setting Privileges for Trusted Stored Program Units
- Re-Compiling Trusted Stored Program Units
- Recreating Trusted Stored Program Units
- Executing Trusted Stored Program Units

Creating Trusted Stored Program Units

You create a trusted stored program unit in the same way that you create a standard procedure, function, or package: using the statement CREATE PROCEDURE, CREATE FUNCTION, or CREATE PACKAGE and CREATE PACKAGE BODY. The program unit becomes trusted when you grant it Oracle Label Security privileges.

See Also: *Oracle9i SQL Reference*

Setting Privileges for Trusted Stored Program Units

When a developer creates a stored program unit, the Oracle Label Security administrator can verify the correctness of the code before granting the necessary privileges to the stored program unit. Whenever the trusted stored program unit is recreated or replaced, its privileges are removed. The Oracle Label Security administrator must then re-verify the code and grant the privileges once again.

Re-Compiling Trusted Stored Program Units

Re-compiling a trusted stored program unit, either automatically or manually (using ALTER PROCEDURE), does not affect its Oracle Label Security privileges. You must, however, re-grant the EXECUTE privilege on the program unit after re-compiling.

Recreating 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 execute even if the Oracle Label Security privileges have been removed.

If you re-create a procedure, function, or package, 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 re-grant it the required privileges.

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 which can grant the appropriate Oracle Label Security privileges, as required.

Executing Trusted Stored Program Units

Under Oracle Label Security all of the standard Oracle9*i* controls on procedure invocation (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 executed, 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 executed, 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 execute with 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.

Using SA_UTL Functions to Set and Return Label Information

The SA_UTL package provides several functions for use within PL/SQL programs. These functions return information about the current values of the session security attributes, in the form of numeric label values. While they can be used in program units which are not trusted, these functions 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 which returns the same labels in character string format.

- Viewing Session Label and Row Label Using SA_UTL
- Setting the Session Label and Row Label Using SA_UTL
- Returning Greatest Lower Bound and Least Upper Bound

See Also: "Viewing Session Attributes with SA_SESSION Functions" on page 4-23

Viewing Session Label and Row Label Using SA_UTL

SA_UTL.NUMERIC_LABEL

This procedure returns the current session label. It takes a policy name as the input parameter and returns a NUMBER value.

```
SA_UTL.NUMERIC_LABEL (policy_name) RETURN NUMBER;
```

SA_UTL.NUMERIC_ROW_LABEL

This procedure returns the current row label. It takes a policy name as the input parameter and returns a NUMBER value.

```
SA_UTL.NUMERIC_ROW_LABEL (policy_name) RETURN NUMBER;
```

SA_UTL.DATA_LABEL

This function returns TRUE if the label is a *data* label.

```
FUNCTION DATA_LABEL(label IN NUMBER)
RETURN BOOLEAN;
```

Setting the Session Label and Row Label Using SA_UTL

These procedures use numeric labels instead of character strings as input values. Available SA_SESSION procedures perform the same functions as these, but in character string format.

SA_UTL.SET_LABEL

Use this procedure to set 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.

```
PROCEDURE SET_LABEL (policy_name IN VARCHAR2,
                     label IN NUMBER);
```

policy_name	The name of an existing policy
label	The label to set as the session label

SA_UTL.SET_ROW_LABEL

Use this procedure to set 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.

```
PROCEDURE SET_ROW_LABEL (policy_name IN VARCHAR2,
                         row_label IN NUMBER);
```

policy_name	The name of an existing policy
row_label	The label to set as the session default row label

See Also: "Changing Your Session and Row Labels with SA_SESSION" on page 4-19

Returning Greatest Lower Bound and Least Upper Bound

GREATEST_LBOUND

This function returns a label that is the greatest lower bound of the two label arguments.

Syntax:

```
FUNCTION GREATEST_LBOUND (label1 IN NUMBER,
                          label2 IN NUMBER)
RETURN NUMBER;
```

LEAST_UBOUND

This function returns an Oracle Label Security label that is the least upper bound of the label arguments.

Syntax:

```
FUNCTION LEAST_UBOUND (label1 IN NUMBER,
                       label2 IN NUMBER)
RETURN NUMBER;
```

See Also: "Determining Upper and Lower Bounds of Labels" on page 4-12. The functions above are the same as those described in Chapter 4, except that these return a number instead of a character string.

10 Auditing Under Oracle Label Security

The Oracle9*i* audit facility lets you hold database users accountable for the operations they perform. It can track specific database objects, operations, users, and privileges. Oracle Label Security supplements this by tracking use of its own administrative operations and policy privileges. It provides the SA_AUDIT_ADMIN package to set and change the policy auditing options.

This chapter explains how to use Oracle Label Security auditing. It contains these topics:

Overview of Oracle Label Security Auditing

Oracle Label Security auditing supplements standard Oracle9*i* auditing by tracking use of its own administrative operations, and use of the policy privileges. You can use either the SA_AUDIT_ADMIN package or Oracle Policy Manager to set and change the auditing options for an Oracle Label Security policy.

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 simply records the sensitivity of the row.

The auditing options which you specify apply only to subsequent sessions, not to the current session. You can specify audit options even if auditing is disabled; no overhead is created simply by making these specifications. When you do enable Oracle Label Security auditing, the options come into effect, and overhead is created beyond that created by standard Oracle9*i* auditing.

Note that Oracle Label Security does not provide labels for audit data written to the operating system audit trial. All Oracle Label Security audit records are written directly to the database audit trail, even if operating system auditing is enabled. If auditing is disabled, then no Oracle Label Security audit records are generated.

Enabling Systemwide Auditing: AUDIT_TRAIL Initialization Parameter

For Oracle Label Security to generate audit records, you must first enable systemwide auditing by setting the Oracle9*i* AUDIT_TRAIL initialization parameter in the database's parameter file. The parameter can be set to one of the following values:

DB	Enables database auditing and directs all audit records to the database audit trail. This approach is recommended by Oracle Corporation. Note that even with AUDIT_TRAIL set to DB, some records are always sent to the operating system audit trail. These include STARTUP and SHUTDOWN statements, as well as CONNECT AS SYSOPER or SYSDBA.
OS	Enables operating system auditing. This directs most of your Oracle9*i* audit records to the operating system, rather than to the database; the records will not contain Oracle Label Security labels. By contrast, any Oracle Label Security auditing will go to the database, with labels. If you set AUDIT_TRAIL to OS, the Oracle Label Security-specific audit records are written to the database audit trail and the other Oracle9*i* audit records are written to the operating system audit trail (with no policy column in the operating system data).
NONE	Disables auditing. This is the default.

After you have edited the parameter file, restart the database instance to enable or disable database auditing as specified.

Set the AUDIT_TRAIL parameter before you set audit options. If you do not set this parameter, you are still able to set audit options, however audit records are not written to the database until the parameter is set and the database instance is restarted.

See Also: For information about enabling and disabling systemwide auditing, setting audit options, and managing the audit trail, see *Oracle9i Database Administrator's Guide*

For information about editing initialization parameters, see *Oracle9i Database Reference*

For details about systemwide AUDIT and NOAUDIT functioning, see *Oracle9i SQL Reference*

Enabling Oracle Label Security Auditing with SA_AUDIT_ADMIN

After you have enabled systemwide auditing, you can use SA_AUDIT_ADMIN procedures to enable or disable Oracle Label Security auditing. To use Oracle Label Security auditing, you must have the *policy*_DBA role.

- Auditing Options for Oracle Label Security
- Enabling Oracle Label Security Auditing with SA_AUDIT_ADMIN.AUDIT
- Disabling Oracle Label Security Auditing with SA_AUDIT_ADMIN.NOAUDIT
- Examining Audit Options with the DBA_SA_AUDIT_OPTIONS View

Auditing Options for Oracle Label Security

The AUDIT and NOAUDIT options are as follows:

Table 10–1 Auditing Options for Oracle Label Security

Option	Description
APPLY	Audits application of specified Oracle Label Security policies to tables and schemas
REMOVE	Audits removal of specified Oracle Label Security policies from tables and schemas
SET	Audits the setting of user authorizations, and user and program privileges
PRIVILEGES	Audits use of all policy-specific privileges

Enabling Oracle Label Security Auditing with SA_AUDIT_ADMIN.AUDIT

Use the AUDIT procedure to enable policy-specific auditing.

Syntax:

```
PROCEDURE AUDIT (
     policy_name      IN VARCHAR2,
     users            IN VARCHAR2 DEFAULT NULL,
     option           IN VARCHAR2 DEFAULT NULL,
     type             IN VARCHAR2 DEFAULT NULL,
     success          IN VARCHAR2 DEFAULT NULL);
```

Parameter	Description	Default Behavior
policy_name	Required. Specifies the name of an existing policy. Auditing of each policy is independent of all others.)	None
users	Optional. A comma-separated list of user names to audit. If not specified, all users are audited.	All users
option	Optional. A comma-separated list of options to be audited. See Table 10–1. If not specified, all default options (that is, options not including privileges) are audited. Audit options for privileged operations should be set explicitly by specifying the PRIVILEGES option, which sets audit options for *all* privileges.	All options
type	Optional. BY ACCESS or BY SESSION. If not specified, audit records are written by session.	BY SESSION
success	Optional. SUCCESSFUL or NOT SUCCESSFUL. If not specified, audit is written for both.	SUCCESSFUL and NOT SUCCESSFUL

If the administrator does not specify any audit options, then all options except the privilege-related ones are audited. Auditing of privileges must be specified explicitly. For example, if the administrator enters

```
SA_AUDIT_ADMIN.AUDIT ('HR');
```

then default auditing options are set for the HR policy. When the administrator enables auditing, it will be performed on all users by session, whether successful or not.

When you set auditing parameters and options, the new values apply only to subsequent sessions, not to the current session.

Consider also a case in which one AUDIT call (with no users specified) enables auditing for APPLY operations for all users, and then a second call enables auditing of REMOVE operations for a specific user. For example:

```
SA_AUDIT_ADMIN.AUDIT ('HR', NULL, 'APPLY');
SA_AUDIT_ADMIN.AUDIT ('HR', 'SCOTT', 'REMOVE');
```

In this case, SCOTT is audited for both APPLY and REMOVE operations.

Disabling Oracle Label Security Auditing with SA_AUDIT_ADMIN.NOAUDIT

To disable policy-specific auditing, use the SA_AUDIT_ADMIN.NOAUDIT procedure.

Syntax:

```
PROCEDURE NOAUDIT (
     policy_name     IN VARCHAR2,
     users           IN VARCHAR2 DEFAULT NULL,
     option          IN VARCHAR2 DEFAULT NULL);
```

Parameter	Description	Default Behavior
policy_name	Required. Specifies the name of an existing policy.	None
users	Optional. A comma-separated list of user names to audit. If not specified, auditing is disabled for all users.	All users
option	Optional. A comma-separated list of options to be disabled. See Table 10–1. If not specified, all default options are disabled. Privileges must be disabled explicitly.	All options

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:

```
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:

```
SA_AUDIT_ADMIN.AUDIT ('HR', 'SCOTT');
SA_AUDIT_ADMIN.AUDIT ('HR');
```

In this case, a subsequent call to NOAUDIT with no users specified (such as the following)

```
SA_AUDIT_ADMIN.NOAUDIT ('HR');
```

does not reverse the auditing which was set for SCOTT explicitly in the first call. Auditing thus continues to be performed on SCOTT. In this way, even if 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 execute NOAUDIT with no options, Oracle Label Security will nonetheless continue to audit privileged operations. For example, if auditing is enabled and you enter

```
SA_AUDIT_ADMIN.NOAUDIT ('HR');
```

then auditing will continue to be performed on the privileged operations (such as WRITEDOWN).

NOAUDIT parameters and options which you set apply only to subsequent sessions, not to current sessions.

If you try to enable an audit option which has already been set, or if you try to disable an audit option which has not been set, Oracle Label Security processes the statement without indicating an error. An attempt to specify an invalid option results in an error message.

Examining Audit Options with the DBA_SA_AUDIT_OPTIONS View

This section describes the view which displays the Oracle Label Security auditing options and privileges.

The DBA_SA_AUDIT_OPTIONS view contains the following columns:

```
Name                                      Null?    Type
----------------------------------------- -------- ------------
POLICY_NAME                               NOT NULL VARCHAR2(30)
USER_NAME                                 NOT NULL VARCHAR2(30)
APY                                                VARCHAR2(3)
REM                                                VARCHAR2(3)
SET_                                               VARCHAR2(3)
PRV                                                VARCHAR2(3)
```

Output is similar to the following:

Table 10–2 DBA_SA_AUDIT_OPTIONS Sample Output

POLICY_ NAME	USER_NAME	APY	REM	SET	PRV
HR	SCOTT	-/-	-/-	-/-	A/A
HR	LBACSYS	S/S	S/S	S/S	-/-

See Also: *Oracle9i Database Administrator's Guide*

Managing Policy Label Auditing

This section describes procedures available to manage policy label auditing:

- Policy Label Auditing with SA_AUDIT_ADMIN.AUDIT_LABEL
- Disabling Policy Label Auditing with SA_AUDIT_ADMIN.NOAUDIT_LABEL
- Finding Label Audit Status with AUDIT_LABEL_ENABLED

Policy Label Auditing with SA_AUDIT_ADMIN.AUDIT_LABEL

Use the AUDIT_LABEL procedure to record policy labels during auditing. It causes the user's session label to be stored in the audit table.

Syntax:

```
PROCEDURE AUDIT_LABEL (
     policy_name     IN VARCHAR2);
```

Parameter	Description	Default
policy_name	Required. Specifies the name of an existing policy.	None

Disabling Policy Label Auditing with SA_AUDIT_ADMIN.NOAUDIT_LABEL

Use the NOAUDIT_LABEL procedure to disable auditing of policy labels.

Syntax:

```
PROCEDURE NOAUDIT_LABEL (
     policy_name     IN VARCHAR2);
```

Parameter	Description	Default
policy_name	Required. Specifies the name of an existing policy.	None

Finding Label Audit Status with AUDIT_LABEL_ENABLED

Use the AUDIT_LABEL_ENABLED function to show whether labels are being recorded in audit records for the policy.

Syntax:

```
FUNCTION AUDIT_LABEL_ENABLED (policy_name IN VARCHAR2)
     RETURN boolean;
```

Creating and Dropping an Audit Trail View for Oracle Label Security

This section contains these topics:

- Creating a View with SA_AUDIT_ADMIN.CREATE_VIEW
- Dropping the View with SA_AUDIT_ADMIN.DROP_VIEW

Creating a View with SA_AUDIT_ADMIN.CREATE_VIEW

The CREATE_VIEW procedure creates an audit trail view named DBA_*policyname*_AUDIT_TRAIL, which 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, the user can optionally specify a shorter view name.

Syntax:

```
PROCEDURE CREATE_VIEW (
     policy_name     IN VARCHAR2);
     view_name       IN VARCHAR2    DEFAULT NULL);
```

where *policy_name* specifies the name of an existing policy

Dropping the View with SA_AUDIT_ADMIN.DROP_VIEW

The DROP_VIEW procedure drops the audit trail view for the specified policy.

Syntax:

```
PROCEDURE DROP_VIEW (
     policy_name     IN VARCHAR2);
     view_name       IN VARCHAR2    DEFAULT NULL);
```

where *policy_name* specifies the name of a policy. *View_name* is an optional parameter which can have a maximum of 14 characters.

Oracle Label Security Auditing Tips

This section contains these topics:

- Strategy for Setting SA_AUDIT_ADMIN Options
- Auditing Privileged Operations

Strategy for Setting SA_AUDIT_ADMIN Options

Before setting any audit options, you must devise an auditing strategy which monitors events of interest, without recording extraneous events. You should periodically review this strategy, because applications, user base, configurations, and other external factors can change.

The Oracle Label Security options, and those provided by the Oracle9*i* audit facility, might not directly address all of your specific or application-dependent auditing requirements. However, through use of database triggers, you can audit specific events and record specific information that you cannot audit and record using the more generic audit facility.

> **See Also:** For more information about using triggers for auditing, see *Oracle9i Database Concepts*

Auditing Privileged Operations

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.

11 Using Oracle Label Security with a Distributed Database

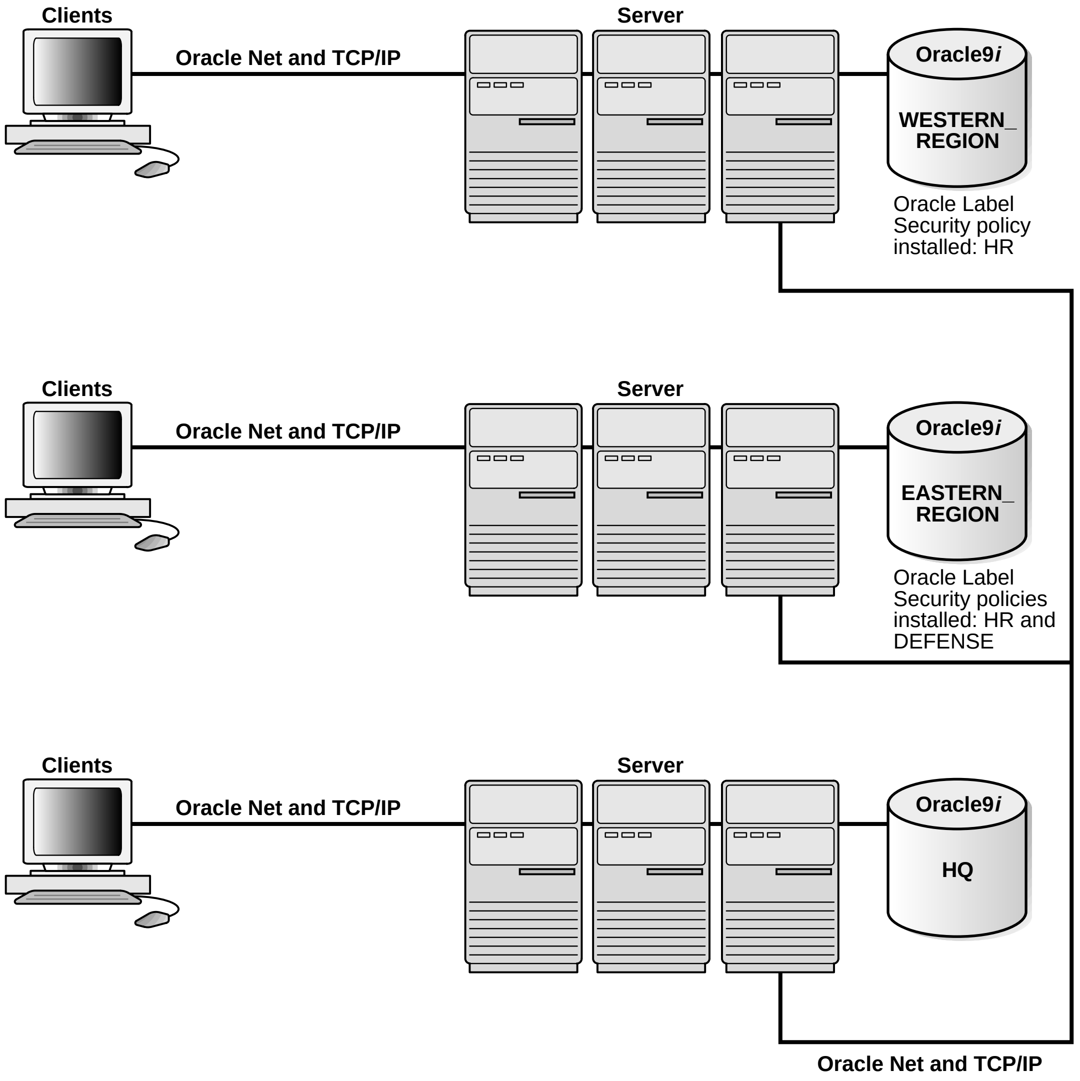
This chapter describes special considerations for using Oracle Label Security in a distributed configuration. It contains the following sections:

An Oracle Label Security Distributed Configuration

A network configuration that supports distributed databases can include multiple Oracle9*i* servers, or other database servers, running on the same or different operating systems. Each cooperative server in a distributed system communicates with other clients and servers over a network.

Figure 11–1 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, you can access data if your userid on EASTERN_REGION is authorized to see it, even if locally (on WESTERN_REGION) you do not have this access.

Figure 11–1 Using Oracle Label Security with a Distributed Database



Connecting to a Remote Database Under Oracle Label Security

Distributed databases behave 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 appropriate labels, you can access the data. If not, you cannot access the data.

The database link sets up the connection to the remote database and identifies the user that will be associated with the remote session. Your Oracle Label Security authorizations on the remote database are based upon 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, her authorizations are based on the labels and privileges of remote user AUSTEN, since AUSTEN is the user identified in the database link. When JANE issues 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.

Establishing Session Label and Row Label for a Remote Session

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 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 the local user’s session label is S when the database link is established, the S label is passed over. This is a valid label; the user can connect and read SENSITIVE data.
- If the local user’s session label is HS when the database link is established, the HS level is passed across, but it is not valid for the remote user. The local user 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. Afterward you can change the label, but to do so you must execute the SA_SESSION.SET_LABEL procedure on the remote database.

The local user described above 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. Her default label is S on the remote database.

On the local database, the user can set her session label to her maximum level of HS, but if the label of the remote user is set to S, then she can only retrieve S data from the remote database. If she performs a distributed query, she will get HS data from the local database, and S data from the remote database.

Setting Up Labels in a Distributed Environment

It is advisable to use the same label component definitions and label tags on any database which is to be protected by the policy.

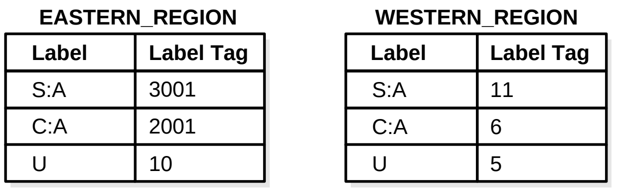
- Setting Label Tags in a Distributed Environment
- Setting Numeric Form of Label Components in a Distributed Environment

Setting 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, 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; the label tag, however, 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, the tags have no meaning. Serious consequences can result.

Figure 11–2 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, 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.

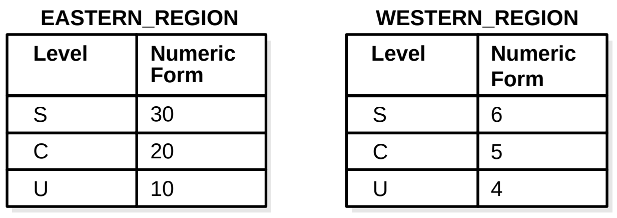
To avoid the complexities of label tags, you can simply convert labels to strings upon retrieval (using LABEL_TO_CHAR) and use CHAR_TO_LABEL when you store labels. Operations will succeed as long as the component names are the same.

Setting Numeric Form of Label Components in a Distributed Environment

In a distributed environment you should use the same relative ranking of the numeric form of the level component, in order to ensure proper sorting of the labels.

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 11–3 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

Using Oracle Label Security Policies in a Distributed Environment

Oracle Label Security supports all standard Oracle9*i* 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, you must apply the same Oracle Label Security policy to both the local and the remote database.
- 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 from 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—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, this user can obtain data from Database 3 as well.

Using Replication with Oracle Label Security

This section explains how to use the replication option with tables protected by Oracle Label Security policies. It contains these topics:

- Introduction to Replication Under Oracle Label Security
- Contents of a Materialized View
- Requirements for Creating Materialized Views Under Oracle Label Security
- How to Refresh Materialized Views

See Also: For a complete explanation of replication in Oracle9*i*, and how to set up the replication environment, see *Oracle9i Replication*

For general information about using materialized views, see *Oracle9i Database Concepts*

Oracle9i Data Warehousing Guide

Introduction to Replication Under Oracle Label Security

This section introduces the use of replication under Oracle Label Security. It contains the following topics:

- Replication Functionality Supported by Oracle Label Security
- Row Level Security Restriction on Replication Under Oracle Label Security

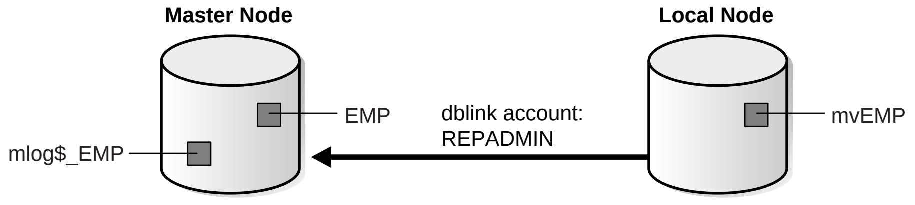
Replication Functionality Supported by Oracle Label Security

Oracle Label Security supports standard replication and Advanced Replication, including multimaster replication and updatable materialized views (snapshots).

Oracle9*i* 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 Figure 11–4, a master table is a table you wish to replicate, on a node that you designate as the master node. Using a dblink account (such as REPADMIN), 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 machine.) 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 11–4 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.

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 thus 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 / or the materialized view owner) must have READ or FULL privilege. As a result, the queries used to perform the replication will not be modified by RLS.

See Also: *Oracle9i Database Concepts*

Contents of a Materialized View

This section discusses the contents of materialized views.

- How Materialized View Contents Are Determined
- Complete Materialized Views
- Partial Materialized Views

How Materialized View Contents Are Determined

Oracle Label Security performs the following steps when creating materialized views. These 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 which 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 only writes 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

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, the result is a complete materialized view of the master table. To ensure that the materialized view is complete, ensure that you 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, you may not be able to perform a replication refresh.

Partial Materialized Views

A partial materialized view is created when you specify a WHERE clause in the materialized view definition. This is a convenient way to pass subsets of data to a remote database.

Note: To create a partial materialized view you must have write access to all the rows being replicated.

Requirements for Creating Materialized Views Under Oracle Label Security

Requirements for creating a materialized view depend upon the type of materialized view you are creating.

- Requirements for the REPADMIN Account
- Requirements for the Owner of the Materialized View
- Requirements for Creating Partial Multilevel Materialized Views
- Requirements for Creating Complete Multilevel Materialized Views

Requirements for the REPADMIN Account

Requirements for the 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 SELECT privilege on the master table.
- It must be the account which establishes the database link from the remote node to the database containing the master table.

See Also: *Oracle9i Replication*

Requirements for the Owner of the Materialized View

Remember that the privileges belonging 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 READ privilege, he can select all rows, regardless of policy options.

Requirements for Creating Partial Multilevel Materialized Views

To create a partial materialized view which includes only some of the rows in a remote master table protected by Oracle Label Security, you must have sufficient privileges to WRITE in the local database at every label retrieved by your query.

Requirements for Creating Complete Multilevel Materialized Views

To create a complete materialized view which includes every row in a remote master table protected by Oracle Label Security, you must be able to WRITE in the local database at the labels of all of the rows retrieved by the defined materialized view query.

How to Refresh Materialized Views

If the contents or definition of a master table changes, refresh the materialized view so that it 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 of 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, use the optional materialized view background processes, SNP*n*, 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: For information about SNP*n* background processes, see *Oracle9i Database Administrator's Guide*

12

Performing DBA Functions Under Oracle Label Security

The standard Oracle9*i* utilities can be used under Oracle Label Security, but certain restrictions apply, and extra steps may be required to get the expected results. This chapter describes these special considerations. It assumes you are using policy label columns of the NUMBER datatype.

The chapter contains these sections:

Using the Export Utility with Oracle Label Security

The 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.

- For any tables protected by an Oracle Label Security policy, only rows with labels authorized for read access will be exported; unauthorized rows will not be included in the export file. Consequently, to export *all* the data in protected tables, you must have a privilege (such as FULL or READ) which gives you complete access.
- SQL statements to reapply policies are exported along with tables and schemas that are exported. These statements are executed 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*, it is exported as a normal, unhidden column.
- The LBACSYS schema cannot be exported due to the use of opaque types in Oracle Label Security. To export an entire database, you must individually specify all of the schemas and/or tables (except for the LBACSYS schema). Use standard backup techniques to back up the LBACSYS schema.

See Also: *Oracle9i Database Utilities*

Using the Import Utility with Oracle Label Security

This section explains how the Import utility functions under Oracle Label Security:

- Requirements for Import Under Oracle Label Security
- Defining Data Labels for Import
- Importing Labeled Data Without Installing Oracle Label Security
- Importing Unlabeled Data
- Importing Tables with Hidden Columns

See Also: *Oracle9i Database Utilities*

Requirements for Import Under Oracle Label Security

To use the Import utility 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, as follows:

1. Install Oracle Label Security.
2. Create any Oracle Label Security policies which protect the data to be imported. The policies must 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. Tag values assigned to the policy labels in each database must be the same. (Note that if you are importing into a database from which you exported, the components are most likely already defined.)

Verifying Import User Authorizations

To successfully import data under Oracle Label Security, the user running the import operation must be authorized for all of the labels required to insert the data and labels contained in the export file. Errors will be raised upon import if the following requirements are not met:

Requirement 1: To assure that all rows can be imported, the user 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.

Requirement 2: The user must also have the ability to write all rows that have been exported. This can be accomplished by one of the following methods:

- The user can be granted the FULL privilege.
- A user-defined labeling function can be applied to the table.
- The user can be given sufficient authorization to write all labels contained in the import file.

Defining Data Labels for Import

The label definitions at the time of import must include all of the policy labels used in the export file. You can use the views DBA_SA_LEVELS, DBA_SA_COMPARTMENTS, DBA_SA_GROUPS, and DBA_SA_LABELS 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, the operation will fail.

Importing Labeled Data Without Installing Oracle Label Security

When policy label columns are defined as a NUMBER datatype, 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, since the SQL statements to reapply the policy to the imported tables and schemas will fail.

Importing 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.

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, you must pre-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 causes the policy label column to be added to the table as a hidden column.
2. Then remove the policy from the table, so that the enforcement options specified in the export file can be re-applied to the table during the import operation.
3. Perform the import. In this way, the hidden property of the label column is preserved.

Using SQL*Loader with Oracle Label Security

SQL*Loader moves data from external files into tables in an Oracle9*i* database. This section contains these topics:

- Requirements for Using SQL*Loader Under Oracle Label Security
- Oracle Label Security Input to SQL*Loader

See Also: For information about SQL*Loader, including log files, discard files, and bad files, see *Oracle9i Database Utilities*

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. Since SQL*Loader performs INSERT operations, all of the standard requirements apply when using SQL*Loader on tables protected by Oracle Label Security policies.

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, 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.

You can use the following variations when loading Oracle Label Security data with SQL*Loader:

`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 col2 above; fieldtype 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 POSITION (1:22) HIDDEN 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 datafile specified by this control file:

```
HS:FN                 231 ACCOUNTING  REDWOOD SHORES
```

Performance Tips for Oracle Label Security

This section explains how to achieve optimal performance with Oracle Label Security.

- Using ANALYZE to Improve Oracle Label Security Performance
- Creating Indexes on the Policy Label Column
- Planning a Label Tag Strategy to Enhance Performance
- Partitioning Data Based on Numeric Label Tags

Using ANALYZE to Improve Oracle Label Security Performance

Run the ANALYZE command on the Oracle Label Security data dictionary tables in the LBACSYS schema, so that the cost-based optimizer can improve execution plans on queries. This will improve Oracle Label Security performance.

Running ANALYZE on application tables improves the application SQL performance.

Creating Indexes on the Policy Label Column

By creating the appropriate type of index on the policy label column, you can improve the performance of user-issued queries on protected tables.

If you have applied an Oracle Label Security policy on a database table in a particular schema, 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, consider creating a bitmapped index.
- If the cardinality of data in the policy label column is high, consider creating a B-tree index.

Example 1:

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 issues 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 which are processed, based on the user's authorizations. To improve performance of this query, you could create an index on the HR_LABEL column.

Example 2:

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.

See Also: *Oracle9i Database Performance Tuning Guide and Reference*

Planning a Label Tag Strategy 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; 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.

Consider, for example, label tags assigned as follows:

Table 12–1 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 issues the following query:

```
SELECT COUNT (*) FROM scott.emp;
```

Oracle Label Security adds a predicate which includes a BETWEEN clause (based on the user’s maximum and minimum authorizations) to restrict the set of rows this user can see:

```
SELECT COUNT (*) FROM scott.emp
  AND 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.

By contrast, unnecessary work would be performed if tag values were assigned as follows:

Table 12–2 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 he 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.

Partitioning Data Based on Numeric Label Tags

If you are using a numeric ordering strategy with the numeric label tags which you have applied to the labels, you can use this as a basis for Oracle9*i* data partitioning. Depending upon 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, and 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 issued, 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 how to do this. 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) );
```

Creating Additional Databases After Installation

When you install the Oracle9*i* Enterprise Edition and Oracle Label Security, an initial Oracle8i database is created. You can then install Oracle Label Security, as described in the *Oracle Label Security Installation Notes* for your platform.

If you wish to create additional databases, Oracle Corporation recommends that you do this using the Oracle Database Configuration Assistant. Alternatively, you can create additional databases by following the steps listed in Chapter 2 of the *Oracle9i Database Administrator's Guide*

Each time you create a new database, you must install into it the Oracle Label Security data dictionary tables, views, and packages, and create the LBACSYS account. For the first database, this is done automatically when you install Oracle Label Security. For additional databases, you must perform the following tasks manually.

Note: If you have not installed Oracle Label Security at least once in your target Oracle environment, you must first do so using the Oracle Universal Installer.

---

1. In your init*sid*.ora file, set the COMPATIBLE parameter to the current Oracle9*i* release which you are running. (This must be no lower than 8.1.7.)

   Shut down and restart your database so that this change will take effect.
2. Connect to the Oracle9*i* instance as user SYS, using the `AS SYSDBA` syntax.
3. Run the script `$ORACLE_HOME/rdbms/admin/catols.sql`.

   This script installs the label-based framework, data dictionary, datatypes, 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.
4. Change the default password of the LBACSYS user.

Now you can proceed to create an Oracle Label Security policy.

See Also: For a complete discussion of Oracle database creation, see *Oracle9i Database Administrator's Guide*

13 Releasability Using Inverse Groups

This chapter discusses the Oracle Label Security implementation of releasability using inverse groups. It contains the following sections:

Note: The Oracle Policy Manager graphical user interface is not supported for policies which contain inverse groups.

Introduction to 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, US cannot access data which only has inverse group UK. Adding US to that data makes it accessible to all users with the inverse groups UK, 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.

Comparing Standard Groups and Inverse Groups

Groups in Oracle Label Security identify organizations which 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 a label with inverse groups, it checks to see whether the data, not the user, has the appropriate group authorizations: does the data have *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 which contains 3 standard groups: Eastern, Western, and Southern. User1’s label authorizations include the groups Eastern and Western. Assuming 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. She *cannot* view records that have only the Eastern group, only the Western group, or that have no groups at all.

Table 13–1 shows all the rows which User1 can potentially access, given the type of group which is used in the policy.

Table 13–1 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: thus 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 which 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: "Groups" on page 2-8

How Inverse Groups Work

This section explains how inverse groups are implemented, and how they work. It contains these topics:

Implementing Inverse Groups with the INVERSE_GROUP Enforcement Option

When creating an Oracle Label Security policy, the administrator can specify whether the policy can use inverse group functionality to implement releasability. To do this, he specifies INVERSE_GROUP as one of the *default_options* in the CREATE_POLICY statement.

The INVERSE_GROUP option can only be set at policy creation time. Once a policy is created, this option cannot be changed.

The INVERSE_GROUP option is thus policy-wide. 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.

Whereas 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.

The administrator can give individual users authorization for one or more inverse groups.

Inverse Groups and Label Components

When an Oracle Label Security policy is created with the inverse group option, the components in the policy label (levels, compartments, and groups) are the same as with standard 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:

There are three levels:

UNCLASSIFIED	UN
CONFIDENTIAL	CON
SECRET	SE

One compartment:

FINANCIAL	FIN

Three 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.

Computed Labels with Inverse Groups

This section explains how inverse groups affect computed label values. It contains these topics:

- Computed Session Labels with Inverse Groups
- Inverse Groups and Computed Max Read Groups and Max Write Groups

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 13–2 Computed Session Labels with Inverse Groups

Computed Label	Definition
Max Read Label	The user's maximum level combined with his or her 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 which 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 which 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.

See Also: "Computed Session Labels" on page 3-9

Inverse Groups and Computed Max Read Groups and Max Write Groups

From the computed values in Table 13–2, two sets of groups are identified for label evaluation of read and write access:

Max Read Groups	This is the set of groups contained in the Max Read Label. It identifies the *minimum* set of inverse groups that can be set in any user label.
Max Write Groups	This is the set of groups contained in the Max Write Label. It identifies 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 13–3, 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 13–3 Read and Write Authorizations for Standard Groups and Inverse Groups

	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 which contains at least the 2 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.

Inverse Groups and Hierarchical Structure

Standard groups in Oracle Label Security are hierarchical, such 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.

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 groups and inverse groups. The differences are described below, in "Algorithm for Read Access with Inverse Groups" on page 13-9 and "Algorithm for Write Access with Inverse Groups" on page 13-11.

The effect of inverse groups on the COMPACCESS privilege is described below, in "Algorithms for COMPACCESS Privilege with Inverse Groups" on page 13-13.

Inverse groups have no impact upon the following user privileges:

- PROFILE_ACCESS
- WRITEUP
- WRITEDOWN
- WRITEACROSS

Algorithm for Read Access with Inverse Groups

This section describes the algorithm for read access with inverse groups.

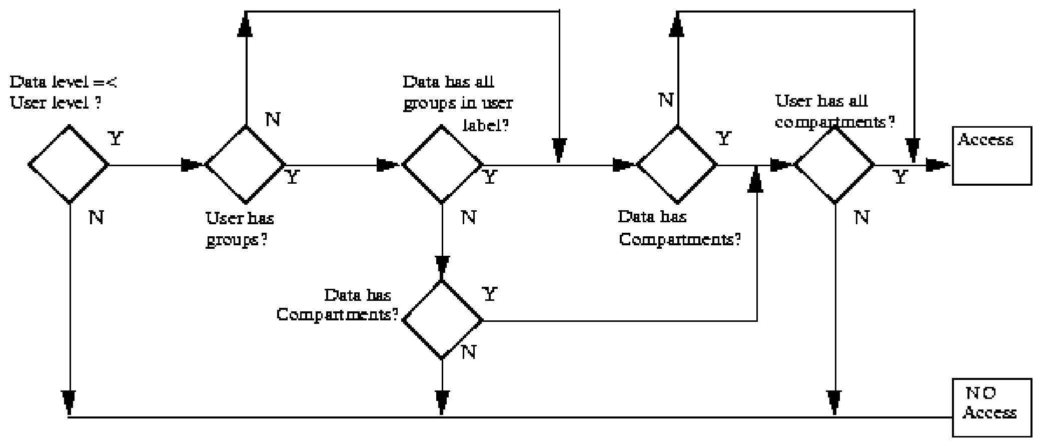
To read data in a table with the INVERSE GROUP option in effect, the label evaluation process proceeds from levels to groups to compartments, as illustrated in Figure 13–1. (Note that the current session label is the label being evaluated.)

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 he can access the data. If not, he 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 13–1 Label Evaluation Process for Read Access with Inverse Groups



See Also: "The Oracle Label Security Algorithm for Read Access" on page 3-13

Algorithm for Write Access with Inverse Groups

This section describes 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 13–2. (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.

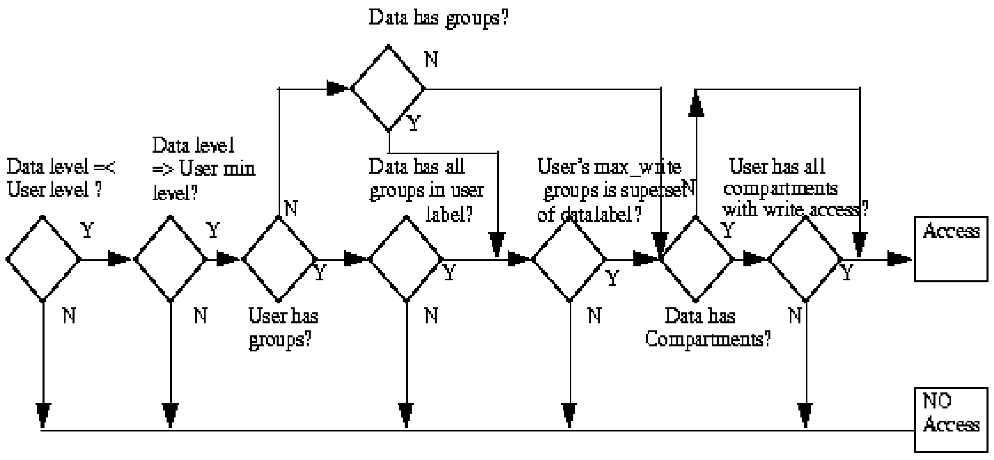
   or

   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 13–2 Label Evaluation Process for Write Access with Inverse Groups


Data has groups?
N
Y
Data level =< User level ?
Y
N
Data level => User min level?
Y
N
N
Y
User has groups?
Data has all groups in user label?
Y
N
User's max_write groups is superset of data label ?
Y
N
N
Data has Compartments?
Y
User has all compartments with write access?
Y
N
Access
NO Access

See Also: "The Oracle Label Security Algorithm for Write Access" on page 3-15

Algorithms for COMPACCESS Privilege with Inverse Groups

This section describes the 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 13–3 and Figure 13–4 show the label evaluation process for read access and write access for a user with 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.)

Note: These flow diagrams are true only when the user has no special privileges.

See Also: "COMPACCESS" on page 3-20

Figure 13–3 Label Evaluation for Read Access with COMPACCESS Privilege and Inverse Groups

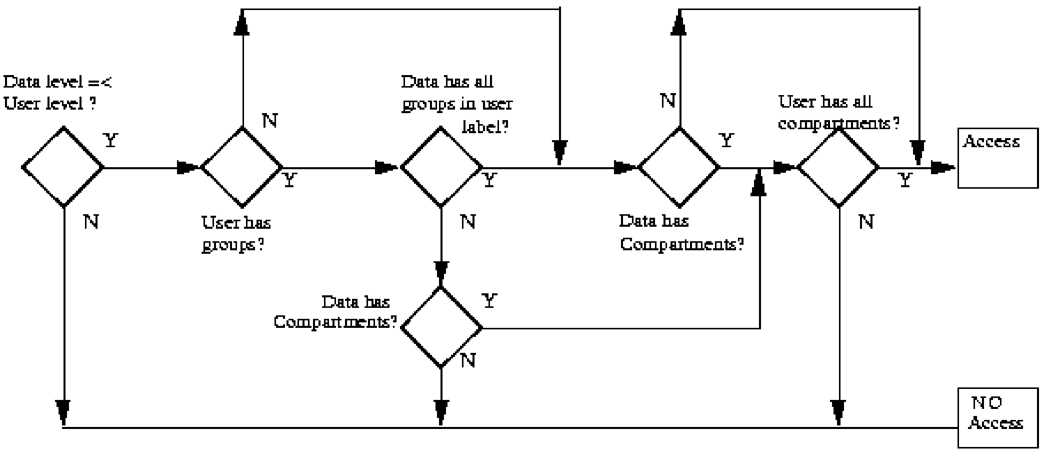
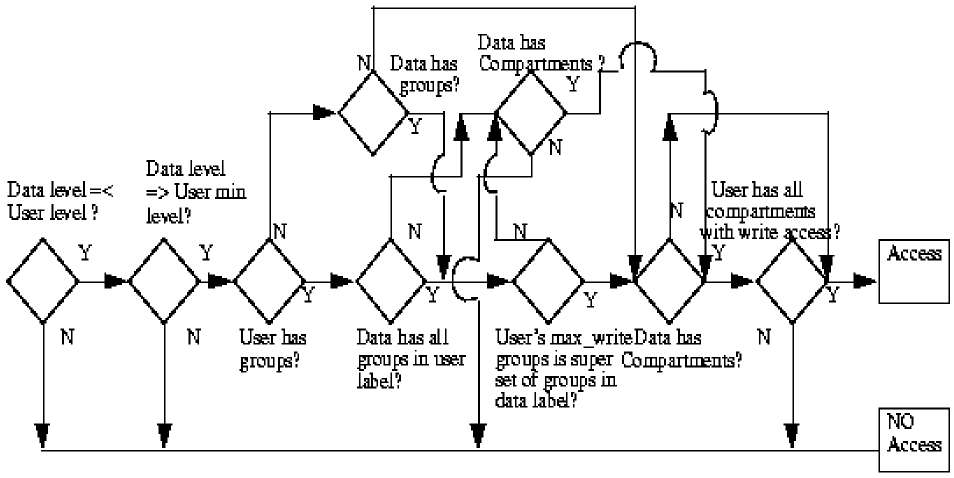


Figure 13–4 Label Evaluation for Write Access with COMPACCESS Privilege and Inverse Groups



Session Labels and Inverse Groups

This section describes how inverse groups affect session labels and and row labels.

- Inverse Groups with SA_USER_ADMIN.SET_DEFAULT_LABEL and SA_USER_ADMIN.SET_ROW_LABEL
- Inverse Groups with SA_SESSION.SET_ROW_LABEL and SA_SESSION.SET_LABEL
- Examples of Session Labels and Inverse Groups

Inverse Groups with SA_USER_ADMIN.SET_DEFAULT_LABEL and SA_USER_ADMIN.SET_ROW_LABEL

The use of inverse groups affects the behavior of Oracle Label Security procedures which 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.

Rules for Changing Default Labels with Standard Groups

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.

Rules for Changing Default Labels with Inverse Groups

In the case of inverse 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 more inverse groups* than the new write label.

The same rule applies for SA_USER_ADMIN.SET_ROW_LABEL.

See Also: "SA_USER_ADMIN.SET_DEFAULT_LABEL" on page 6-13

"SA_USER_ADMIN.SET_ROW_LABEL" on page 6-14

"Dominance Rules for Labels with Inverse Groups" on page 13-27

Inverse Groups with SA_SESSION.SET_ROW_LABEL and SA_SESSION.SET_LABEL

The use of inverse groups affects the behavior of the SA_SESSION.SET_LABEL and SA_SESSION.SET_ROW_LABEL procedures, which can be used to set the user's current session label and row label, respectively.

Rules for Changing Session Label with Standard Groups

With standard groups, the SA_SESSION.SET_LABEL procedure can be used to set the session label to include any 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, 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.

Rules for Changing Session Label and Row Label with Inverse Groups

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 him to access *more* sensitive information. The user should thus 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 his 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, US as his Max Read Groups, and UK,US,CAN as his Max Write Groups. He can set his session label to C:ALPHA:UK,US,CAN but not to C:ALPHA:UK.
- If the user has the inverse group UK as his Max Read Groups, and UK,CAN as his Max Write Groups.assigned to him. He can set his session label to

C:ALPHA:UK,CAN but cannot change it to either C:ALPHA or C:ALPHA:UK,US,CAN.

> **See Also:** "Changing the Session Label with SA_SESSION.SET_LABEL" on page 4-20
>
> "Changing the Row Label with SA_SESSION.SET_ROW_LABEL" on page 4-21

Examples of Session Labels and Inverse Groups

This section presents examples to illustrate the use of inverse groups.

Inverse Groups Example 1

Consider a User1, of a policy that implements inverse groups. The user has the following labels:

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

These values are derived from the foregoing labels:

Max Read Groups	G1,G2
Max Write Groups	G1,G2,G3

The following conclusions can be drawn:

- User1 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 which User1 inserts has the label SE:ALPHA:G1,G2. (This is the same as with standard groups.)
- If User1 leaves the default label as is, and sets his row label to SE:ALPHA:G1,G2,G3, then he will insert SE:ALPHA:G1,G2,G3 in new rows of

data he writes. (In standard groups, he can never set more groups in the row label than in the default label.)

Inverse Groups Example 2

Consider a User1, of a policy that implements inverse groups. The user has the following labels:

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:

These values are derived from the foregoing labels:

Max Read Groups	(an empty set)
Max Write Groups	G1,G2,G3

The following conclusions can be drawn:

- User1 can update any data with level C, compartment ALPHA, and any combination of groups G1, G2, G3, or no groups. He inserts the label C:ALPHA: in new data he writes.
- User2, who has Max Read Groups of G1,G2 or G1,G3, and so on, will not be able to view the data written by User1. This is because User1's Default Row Label contains no groups.
- User1 can choose to set inverse groups in his row label, as long as the inverse groups in the session label dominates the row label (that is, his 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 User1 sets his session label and row label to C:ALPHA:G1:G2:G3, then his data becomes accessible to anyone who has any combination of G1,G2,G3 in his Max Read Groups.

See Also: "Computed Session Labels" on page 3-9

Changes in Behavior of Procedures with Inverse Groups

When the INVERSE_GROUP option is specified at the time the policy is created, a change occurs in the algorithms which determine the read and write access of the user to labeled data. This section describes how inverse groups affect the behavior of the following procedures:

- SYSDBA.CREATE_POLICY with Inverse Groups
- SYSDBA.ALTER_POLICY with Inverse Groups
- SA_USER_ADMIN.ADD_GROUPS with Inverse Groups
- SA_USER_ADMIN.ALTER_GROUPS with Inverse Groups
- SA_USER_ADMIN.SET_GROUPS with Inverse Groups
- SA_USER_ADMIN.SET_USER_LABELS with Inverse Groups
- SA_USER_ADMIN.SET_DEFAULT_LABEL with Inverse Groups
- SA_USER_ADMIN.SET_ROW_LABEL with Inverse Groups
- SA_COMPONENTS.CREATE_GROUP with Inverse Groups
- SA_COMPONENTS.ALTER_GROUP_PARENT with Inverse Groups
- SA_SESSION.SET_LABEL with Inverse Groups
- SA_SESSION.SET_ROW_LABEL with Inverse Groups
- LEAST_UBOUND with Inverse Groups
- GREATEST_LBOUND with Inverse Groups

SYSDBA.CREATE_POLICY with Inverse Groups

The CREATE_POLICY procedure under the SYSDBA package creates the policy, defines an optional policy-specific column name, and specifies a set of default 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);
```

See Also: "Creating a Policy with SA_SYSDBA.CREATE_POLICY" on page 5-9

"Overview of Policy Enforcement Options" on page 7-2

SYSDBA.ALTER_POLICY with Inverse Groups

The ALTER_POLICY procedure under the SYSDBA package enables you to change a policy's default enforcement options, *except for* the INVERSE_GROUP option. Once a policy is configured for inverse groups, it cannot be changed.

See Also: "Modifying Policy Options with SA_SYSDBA.ALTER_POLICY" on page 5-10

SA_USER_ADMIN.ADD_GROUPS with Inverse Groups

The ADD_GROUPS procedure adds groups to a user, indicating whether the groups are authorized for write as well as read.

The *access_mode* is one of two variables which specify the type of access authorized.

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
in_def	Specifies whether these groups should be in the default groups (Y/N)
in_row	Specifies whether these groups should be in the row label (Y/N)

Note that if in_def is Y in a row, then in_row must also be set to Y, but not vice versa.

If the access mode is set to READ_WRITE, the group is added to Max Read Groups, and Max Write Groups. If the group should be added only to the Max Write Groups, then the access mode should be set to SA_UTL.WRITE_ONLY. If not specified, access_mode is set to SA_UTL.READ_WRITE. If in_def is not specfied, then it will be set to Y or N depending on whether the access mode is READ_WRITE or WRITE_ONLY, respectively. The same is the case with the in_row field.

See Also: "Inverse Groups and Computed Max Read Groups and Max Write Groups" on page 13-7

SA_USER_ADMIN.ALTER_GROUPS with Inverse Groups

The ALTER_GROUPS procedure changes the write access, the default label indicator, and/or the row label indicator for each of the groups in the list.

The behavior of inverse groups is the same as described in the case of ADD_GROUPS.

SA_USER_ADMIN.SET_GROUPS with Inverse Groups

The 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 from standard groups, as follows:

read_groups	A comma-separated list of groups which would be Max Read Groups.
write_groups	A comma-separated list of groups which would be Max Write Groups. It must be a superset of read_groups.
def_groups	Specifies the default groups. It should at least have the *read_groups* and *write_groups* should be a superset of *def_groups*.
row_groups	Specifies the row groups. It should at least have the *def_groups* and should be a subset of max write groups.

SA_USER_ADMIN.SET_USER_LABELS with Inverse Groups

The 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 from standard groups, as follows:

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)
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 the max_write_label is not specified, 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 the default_label is not specified, it is set to the 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, it is set to the def_label, with only the compartments and groups authorized for write access. The inverse groups component is set to 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.

SA_USER_ADMIN.SET_DEFAULT_LABEL with Inverse Groups

The SET_DEFAULT_LABEL procedure sets the user's initial session label to the one specified.

All the rules mentioned for setting inverse groups component of session label mentioned in "Session Labels and Inverse Groups" are applicable here.

SA_USER_ADMIN.SET_ROW_LABEL with Inverse Groups

Use the SET_ROW_LABEL procedure to set the user's initial row label to the one specified.

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: "Rules for Changing Default Labels with Inverse Groups" on page 13-16

SA_COMPONENTS.CREATE_GROUP with Inverse Groups

Use the CREATE_GROUP procedure to create a group and specify 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.

SA_COMPONENTS.ALTER_GROUP_PARENT with Inverse Groups

This function is disabled for policies with the inverse group option. An error message is displayed if the user invokes this function.

SA_SESSION.SET_LABEL with Inverse Groups

Use the SET_LABEL procedure to set 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: "Rules for Changing Session Label and Row Label with Inverse Groups" on page 13-17

SA_SESSION.SET_ROW_LABEL with Inverse Groups

Use the SET_ROW_LABEL procedure to set 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: "Rules for Changing Session Label and Row Label with Inverse Groups" on page 13-17

LEAST_UBOUND with Inverse Groups

The LEAST_UBOUND (LUBD) function returns a character string label that is the least upper bound of label1 and label2: that is, the one label which dominates both.

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

GREATEST_LBOUND with Inverse Groups

The GREATEST_LBOUND (GLBD) function can be used to determine the lowest label of the data that can be involved in an operation, given two different labels. It 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

See: "Determining Upper and Lower Bounds of Labels" on page 4-12

Dominance Rules for Labels with 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 (has 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

See Also: "Dominant and Dominated Labels" on page A-2

Part IV

Appendix

A

Advanced Topics in Oracle Label Security

This appendix covers topics of interest to advanced users of Oracle Label Security. It contains these sections:

Analyzing the Relationships Between Labels

This section describes relationships between labels. It contains these topics:

- Dominant and Dominated Labels
- Non-Comparable Labels
- Using Dominance Functions

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, the user is not allowed to access the object.

Label dominance is analyzed in terms of all its components: levels, compartments, and groups.

Table A–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* of 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.

See Also: "Dominance Rules for Labels with Inverse Groups" on page 13-27

Non-Comparable Labels

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.

Using 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 A–2 Functions to Determine Dominance

Function	Meaning
STRICTLY_DOMINATES	The value of *label1* dominates that of *label2*, and is not equal to it.
DOMINATES	The value of *label1* dominates, or is equal to, that of *label2*.
DOMINATED_BY	The value of *label1* is dominated by that of *label2*.
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. Whereas the SA_UTL dominance functions return BOOLEAN values, the standalone dominance functions return integers.

- DOMINATES Standalone Function
- STRICTLY_DOMINATES Standalone Function
- DOMINATED_BY Standalone Function
- STRICTLY_DOMINATED_BY Standalone Function
- SA_UTL.DOMINATES
- SA_UTL.STRICTLY_DOMINATES
- SA_UTL.DOMINATED_BY
- SA_UTL.STRICTLY_DOMINATED_BY

See Also: "Ordering Labeled Data Rows" on page 4-11

DOMINATES Standalone Function

The DOMINATES (DOM) function returns 1 (TRUE) if *label1* dominates *label2*, or 0 (FALSE) if it does not.

Syntax:

```
FUNCTION DOMINATES (
  label1         IN NUMBER,
  label2         IN NUMBER)
RETURN INTEGER;
```

STRICTLY_DOMINATES Standalone Function

The STRICTLY_DOMINATES (SDOM) function returns 1 (TRUE) if *label1* dominates *label2* and is not equal to it.

Syntax:

```
FUNCTION STRICTLY_DOMINATES (
  label1         IN NUMBER,
  label2         IN NUMBER)
RETURN INTEGER;
```

DOMINATED_BY Standalone Function

The DOMINATED_BY (DOM_BY) function returns 1 (TRUE) if *label1* is dominated by *label2*.

Syntax:

```
FUNCTION DOMINATED_BY (
  label1         IN NUMBER,
  label2         IN NUMBER
RETURN INTEGER;
```

STRICTLY_DOMINATED_BY Standalone Function

The STRICTLY_DOMINATED_BY (SDOM_BY) function returns 1 (TRUE) if *label1* is dominated by *label2* and is not equal to it.

Syntax:

```
FUNCTION STRICTLY_DOMINATED_BY (
  label1         IN NUMBER,
  label2         IN NUMBER)
RETURN INTEGER;
```

SA_UTL.DOMINATES

The SA_UTL.DOMINATES function returns TRUE if *label1* dominates *label2*.

Syntax:

```
FUNCTION DOMINATES (
  label1         IN NUMBER,
  label2         IN NUMBER)
RETURN BOOLEAN;
```

SA_UTL.STRICTLY_DOMINATES

The SA_UTL.STRICTLY_DOMINATES function returns TRUE if *label1* dominates *label2* and is not equal to it.

Syntax:

```
FUNCTION STRICTLY_DOMINATES (
  label1         IN NUMBER,
  label2         IN NUMBER)
RETURN BOOLEAN;
```

SA_UTL.DOMINATED_BY

The SA_UTL.DOMINATED_BY function returns TRUE if *label1* is dominated by *label2*.

Syntax:

```
FUNCTION DOMINATED_BY (
  label1         IN NUMBER,
  label2         IN NUMBER)
RETURN BOOLEAN;
```

SA_UTL.STRICTLY_DOMINATED_BY

The SA_UTL.STRICTLY_DOMINATED_BY function returns TRUE if *label1* is dominated by *label2* and is not equal to it.

Syntax:

```
FUNCTION STRICTLY_DOMINATED_BY (
  label1          IN NUMBER,
  label2          IN NUMBER)
RETURN BOOLEAN;
```

OCI Interface for Setting Session Labels

When using OCI to connect, the policy's SYS_CONTEXT variables can be used to initialize the session label and the row label. The variables are set using the OCIAttrSet function to initialize "externally initialized" SYS_CONTEXT variables. These are available in Release 8.1.7 only when Oracle Label Security is installed.

Each policy has a SYS_CONTEXT named SA$*policy_name*_X. There are two variables that can be set: INITIAL_LABEL and INITIAL_ROW_LABEL.

When set to valid labels within the user's authorizations, the new values will be used instead of the default values stored for the user. This is the same mechanism used for remote connections

See Also: Chapter 11, "Using Oracle Label Security with a Distributed Database"

OCIAttrSet

Additional attributes are defined for OCIAttrSet to insert context. Use 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);
```

Note that size is ub4 type.

OCIAttrGet

Then call OCIAttrGet with OCI_ATTR_APPCTX_LIST to get a handle on the application context list descriptor for the session:

```
(session, OCI_HTYPE_SESSION,
               (dvoid *)&ctxl_desc, (ub4)0, OCI_ATTR_APPCTX_LIST, error_handle);
```

Note that ctxl_desc is (OCIParam *) type[

OCIParamGet

Then use 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.

OCIAttrSet

Set the appropriate values in the application context using the three new attributes OCI_ATTR_APPCTX_NAME, OCI_ATTR_APPCTX_ATTR, and OCI_ATTR_APPCTX_VALUE:

```
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 VARCHAR2 type.

OCI Example

The following example shows how to use externalized SYS_CONTEXT with Oracle Label Security.

```
#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 - <one-line expansion of the name>

   DESCRIPTION
     <short description of component this file declares/defines>

   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
     <function return values, for .c file with single function>

   NOTES
     <other useful comments, qualifications, and so on>

   MODIFIED   (MM/DD/YY)
   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 */
```

B Reference

This appendix provides the following reference information:

- Oracle Label Security Data Dictionary Tables and Views
- Restrictions in Oracle Label Security

Oracle Label Security Data Dictionary Tables and Views

- Oracle9i Data Dictionary Tables
- Oracle Label Security Data Dictionary Views
- Oracle Label Security Auditing Views

Oracle9*i* Data Dictionary Tables

Oracle Label Security does not in any way label the Oracle9*i* data dictionary tables. Access is controlled by standard Oracle9*i* system and object privileges. For a description of all data dictionary tables and views, see the *Oracle9i Database Reference*

Oracle Label Security Data Dictionary Views

Oracle Label Security maintains an independent set of data dictionary tables. These tables are exempt from any policy enforcement. This section lists the views which can display information related to Oracle Label Security.

Note that access to the DBA views is granted by default to the SELECT_CATALOG_ROLE, a standard Oracle9*i* role which lets you examine the Oracle9*i* data dictionary.

ALL_SA_AUDIT_OPTIONS

Name	Null?	Type
POLICY_NAME	NOT NULL	VARCHAR2(30)
USER_NAME	NOT NULL	VARCHAR2(30)
APY		VARCHAR2(3)
REM		VARCHAR2(3)
SET_		VARCHAR2(3)
PRV		VARCHAR2(3)

ALL_SA_COMPARTMENTS

Name	Null?	Type
POLICY_NAME	NOT NULL	VARCHAR2(30)
COMP_NUM	NOT NULL	NUMBER(4)
SHORT_NAME	NOT NULL	VARCHAR2(30)
LONG_NAME	NOT NULL	VARCHAR2(80)

ALL_SA_DATA_LABELS

Name	Null?	Type
POLICY_NAME	NOT NULL	VARCHAR2(30)
LABEL		VARCHAR2(4000)
LABEL_TAG		NUMBER

ALL_SA_GROUPS

NAME	Null?	Type
POLICY_NAME	NOT NULL	VARCHAR2(30)
GROUP_NUM	NOT NULL	NUMBER(4)
SHORT_NAME	NOT NULL	VARCHAR2(30)
LONG_NAME	NOT NULL	VARCHAR2(80)
PARENT_NUM		NUMBER(4)
PARENT_NAME		VARCHAR2(30)

ALL_SA_LABELS

Access to ALL_SA_LABELS is PUBLIC, however only the labels authorized for read access by the session are visible.

Name	Null?	Type
POLICY_NAME	NOT NULL	VARCHAR2(30)
LABEL		VARCHAR2(4000)
LABEL_TAG		NUMBER
LABEL_TYPE		VARCHAR2(15)

ALL_SA_LEVELS

Name	Null?	Type
POLICY_NAME		VARCHAR2(30)
LEVEL_NUM		NUMBER(4)
SHORT_NAME		VARCHAR2(30)
LONG_NAME		VARCHAR2(80)

ALL_SA_POLICIES

Name	Null?	Type
POLICY_NAME	NOT NULL	VARCHAR2(30)
COLUMN_NAME	NOT NULL	VARCHAR2(30)
STATUS		VARCHAR2(8)
POLICY_OPTIONS		VARCHAR2(4000)

ALL_SA_PROG_PRIVS

Name	Null?	Type
SCHEMA_NAME	NOT NULL	VARCHAR2(30)
PROGRAM_NAME	NOT NULL	VARCHAR(30)
POLICY_NAME	NOT NULL	VARCHAR2(30)
PROGRAM_PRIVILEGES		VARCHAR2(4000)

ALL_SA_SCHEMA_POLICIES

Name	Null?	Type
POLICY_NAME	NOT NULL	VARCHAR2(30)
SCHEMA_NAME	NOT NULL	VARCHAR2(30)
STATUS		VARCHAR2(8)
SCHEMA_OPTIONS		VARCHAR2(4000)

ALL_SA_TABLE_POLICIES

Name	Null?	Type
POLICY_NAME	NOT NULL	VARCHAR2(30)
SCHEMA_NAME	NOT NULL	VARCHAR2(30)
TABLE_NAME	NOT NULL	VARCHAR2(30)
STATUS		VARCHAR2(8)
TABLE_OPTIONS		VARCHAR2(4000)
FUNCTION		VARCHAR2(1024)
PREDICATE		VARCHAR2(256)

ALL_SA_USERS

Name	Null?	Type
USER_NAME	NOT NULL	VARCHAR2(30)
POLICY_NAME	NOT NULL	VARCHAR2(30)
USER_PRIVILEGES		VARCHAR2(4000)
MAX_READ_LABEL		VARCHAR2(4000)
MAX_WRITE_LABEL		VARCHAR2(4000)
MIN_WRITE_LABEL		VARCHAR2(4000)
DEFAULT_READ_LABEL		VARCHAR2(4000)
DEFAULT_WRITE_LABEL		VARCHAR2(4000)
DEFAULT_ROW_LABEL		VARCHAR2(4000)
USER_LABELS		VARCHAR2(4000)

ALL_SA_USER_LABELS

Name	Null?	Type
USER_NAME	NOT NULL	VARCHAR2(30)
POLICY_NAME	NOT NULL	VARCHAR2(30)
MAX_READ_LABEL	NOT NULL	VARCHAR2(4000)
MAX_WRITE_LABEL		VARCHAR2(4000)
MIN_WRITE_LABEL		VARCHAR2(4000)
DEFAULT_READ_LABEL		VARCHAR2(4000)
DEFAULT_WRITE_LABEL		VARCHAR2(4000)
DEFAULT_ROW_LABEL		VARCHAR2(4000)
LABELS		VARCHAR2(4000)

Note: The field USER_LABELS in ALL_SA_USERS and the field LABELS in ALL_SA_USER_LABELS are retained solely for backward compatibility and will be removed in the next release.

ALL_SA_USER_LEVELS

Name	Null?	Type
POLICY_NAME	NOT NULL	VARCHAR2(30)
USER_NAME	NOT NULL	VARCHAR2(30)
MAX_LEVEL	NOT NULL	VARCHAR2(30)
MIN_LEVEL	NOT NULL	VARCHAR2(30)
DEF_LEVEL	NOT NULL	VARCHAR2(30)
ROW_LEVEL	NOT NULL	VARCHAR2(30)

ALL_SA_USER_PRIVS

Name	Null?	Type
USER_NAME	NOT NULL	VARCHAR2(30)
POLICY_NAME	NOT NULL	VARCHAR2(30)
USER_PRIVILEGES		VARCHAR2(4000)

DBA_SA_AUDIT_OPTIONS

Name	Null?	Type
POLICY_NAME	NOT NULL	VARCHAR2(30)
USER_NAME	NOT NULL	VARCHAR2(30)
APY		VARCHAR2(3)
REM		VARCHAR2(3)
SET_		VARCHAR2(3)
PRV		VARCHAR2(3)

DBA_SA_COMPARTMENTS

Name	Null?	Type
POLICY_NAME	NOT NULL	VARCHAR2(30)
COMP_NUM	NOT NULL	NUMBER(4)
SHORT_NAME	NOT NULL	VARCHAR2(30)
LONG_NAME	NOT NULL	VARCHAR2(80)

DBA_SA_DATA_LABELS

Name	Null?	Type
POLICY_NAME	NOT NULL	VARCHAR2(30)
LABEL		VARCHAR2(4000)
LABEL_TAG		NUMBER

DBA_SA_GROUPS

Name	Null?	Type
POLICY_NAME	NOT NULL	VARCHAR2(30)
GROUP_NUM	NOT NULL	NUMBER(4)
SHORT_NAME	NOT NULL	VARCHAR2(30)
LONG_NAME	NOT NULL	VARCHAR2(80)
PARENT_NUM		NUMBER(4)
PARENT_NAME		VARCHAR2(30)

DBA_SA_GROUP_HIERARCHY

Name	Null?	Type
POLICY_NAME	NOT NULL	VARCHAR2(30)
HIERARCHY_LEVEL		NUMBER
GROUP_NAME		VARCHAR2(4000)

DBA_SA_LABELS

Name	Null?	Type
POLICY_NAME	NOT NULL	VARCHAR2(30)
LABEL		VARCHAR2(4000)
LABEL_TAG		NUMBER
LABEL_TYPE		VARCHAR2(15)

DBA_SA_LEVELS

Name	Null?	Type
POLICY_NAME	NOT NULL	VARCHAR2(30)
LEVEL_NUM	NOT NULL	NUMBER(4)
SHORT_NAME	NOT NULL	VARCHAR2(30)
LONG_NAME	NOT NULL	VARCHAR2(80)

DBA_SA_POLICIES

Name	Null?	Type
POLICY_NAME	NOT NULL	VARCHAR2(30)
COLUMN_NAME	NOT NULL	VARCHAR2(30)
STATUS		VARCHAR2(8)
POLICY_OPTIONS		VARCHAR2(4000)

DBA_SA_PROG_PRIVS

Name	Null?	Type
SCHEMA_NAME	NOT NULL	VARCHAR2(30)
PROGRAM_NAME	NOT NULL	VARCHAR2(30)
POLICY_NAME	NOT NULL	VARCHAR2(30)
PROGRAM_PRIVILEGES		VARCHAR2(4000)

DBA_SA_SCHEMA_POLICIES

Name	Null?	Type
POLICY_NAME	NOT NULL	VARCHAR2(30)
SCHEMA_NAME	NOT NULL	VARCHAR2(30)
STATUS		VARCHAR2(8)
SCHEMA_OPTIONS		VARCHAR2(4000)

DBA_SA_TABLE_POLICIES

Name	Null?	Type
POLICY_NAME	NOT NULL	VARCHAR2(30)
SCHEMA_NAME	NOT NULL	VARCHAR2(30)
TABLE_NAME	NOT NULL	VARCHAR2(30)
STATUS		VARCHAR2(8)
TABLE_OPTIONS		VARCHAR2(4000)
FUNCTION		VARCHAR2(1024)
PREDICATE		VARCHAR2(256)

DBA_SA_USERS

Name	Null?	Type
USER_NAME	NOT NULL	VARCHAR2(30)
POLICY_NAME	NOT NULL	VARCHAR2(30)
USER_PRIVILEGES		VARCHAR2(4000)
MAX_READ_LABEL		VARCHAR2(4000)
MAX_WRITE_LABEL		VARCHAR2(4000)
MIN_WRITE_LABEL		VARCHAR2(4000)
DEFAULT_READ_LABEL		VARCHAR2(4000)
DEFAULT_WRITE_LABEL		VARCHAR2(4000)
DEFAULT_ROW_LABEL		VARCHAR2(4000)
USER_LABELS		VARCHAR2(4000)

Note: The field USER_LABELS in DBA_SA_USERS is retained solely for backward compatibility and will be removed in the next release.

DBA_SA_USER_COMPARTMENTS

Name	Null?	Type
POLICY_NAME	NOT NULL	VARCHAR2(30)
USER_NAME	NOT NULL	VARCHAR2(30)
COMP	NOT NULL	VARCHAR2(30)
RW_ACCESS		VARCHAR2(5)
DEF_COMP	NOT NULL	VARCHAR2(1)
ROW_COMP	NOT NULL	VARCHAR2(1)

DBA_SA_USER_GROUPS

Name	Null?	Type
POLICY_NAME	NOT NULL	VARCHAR2(30)
USER_NAME	NOT NULL	VARCHAR2(30)
GRP	NOT NULL	VARCHAR2(30)
RW_ACCESS		VARCHAR2(5)
DEF_GROUP	NOT NULL	VARCHAR2(1)
ROW_GROUP	NOT NULL	VARCHAR2(1)

DBA_SA_USER_LABELS

Name	Null?	Type
USER_NAME	NOT NULL	VARCHAR2(30)
POLICY_NAME	NOT NULL	VARCHAR2(30)
MAX_READ_LABEL	NOT NULL	VARCHAR2(4000)
MAX_WRITE_LABEL		VARCHAR2(4000)
MIN_WRITE_LABEL		VARCHAR2(4000)
DEFAULT_READ_LABEL		VARCHAR2(4000)
DEFAULT_WRITE_LABEL		VARCHAR2(4000)
DEFAULT_ROW_LABEL		VARCHAR2(4000)
LABELS		VARCHAR2(4000)

Note: The field LABELS in DBA_SA_USER_LABELS is retained solely for backward compatibility and will be removed in the next release.

DBA_SA_USER_LEVELS

Name	Null?	Type
POLICY_NAME	NOT NULL	VARCHAR2(30)
USER_NAME	NOT NULL	VARCHAR2(30)
MAX_LEVEL	NOT NULL	VARCHAR2(30)
MIN_LEVEL	NOT NULL	VARCHAR2(30)
DEF_LEVEL	NOT NULL	VARCHAR2(30)
ROW_LEVEL	NOT NULL	VARCHAR2(30)

DBA_SA_USER_PRIVS

Name	Null?	Type
USER_NAME	NOT NULL	VARCHAR2(30)
POLICY_NAME	NOT NULL	VARCHAR2(30)
USER_PRIVILEGES		VARCHAR2(4000)

Oracle Label Security Auditing Views

Using the SA_AUDIT_ADMIN.CREATE_VIEW procedure, you can create an audit trail view for the specified policy. By default, this view is named DBA_*policyname*_AUDIT_TRAIL.

The DBA_SA_AUDIT_OPTIONS view contains the columns POLICY_NAME, USER_NAME, APY, SET_, and PRV.

See Also: "Creating and Dropping an Audit Trail View for Oracle Label Security" on page 10-11

Restrictions in Oracle Label Security

The following restrictions exist in this Oracle Label Security release:

- CREATE TABLE AS SELECT Restriction in Oracle Label Security
- Label Tag Restriction
- Export Restriction in Oracle Label Security
- Oracle Label Security Deinstallation Restriction
- Shared Schema Support
- Hidden Columns Restriction

CREATE TABLE AS SELECT Restriction in Oracle Label Security

If you attempt to perform CREATE TABLE AS SELECT in a schema which is protected by an Oracle Label Security policy, the statement will fail.

Label Tag Restriction

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.

Export Restriction in Oracle Label Security

The LBACSYS schema cannot be exported due to the use of opaque types in Oracle Label Security. To export an entire database, you must individually specify all of the schemas and/or tables (except for the LBACSYS schema). Use standard backup techniques to back up the LBACSYS schema.

Oracle Label Security Deinstallation 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 deinstall Oracle Label Security.

> **See Also:** Your platform-specific Oracle installation documentation

Shared Schema Support

User accounts defined in the Oracle Internet Directory cannot be given individual Oracle Label Security authorizations. However, authorizations can be given to the shared schema to which the directory users are mapped.

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.

Index