# Oracle® Database

Security Guide

10*g* Release 1 (10.1)

**Part No. B10773-01**

December 2003

Oracle Database Security Guide, 10*g* Release 1 (10.1)

Part No. B10773-01

Primary Authors: Laurel P. Hale, Jeffrey Levinger

Contributing Authors: Ruth Baylis, Michele Cyran, John Russell

Graphic Designer: Valarie Moore

# Contents

# 6 Access Controls on Tables, Views, Synonyms, or Rows

# 7 Security Policies

## List of Figures

## List of Tables

# Send Us Your Comments

**Oracle Database Security Guide, 10*g* Release 1 (10.1)**

**Part No. B10773-01**

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

This document provides a comprehensive overview of security for Oracle Database. It includes conceptual information about security requirements and threats, descriptions of Oracle Database security features, and procedural information that explains how to use those features to secure your database.

This preface contains these topics:

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

## Audience

The Oracle Database Security Guide is intended for database administrators (DBAs), security administrators, application developers, and others tasked with performing the following operations securely and efficiently:

- Designing and implementing security policies to protect the organization's data, users, and applications from accidental, inappropriate, or unauthorized actions
- Creating and enforcing policies and practices of auditing and accountability for any such inappropriate or unauthorized actions
- Creating, maintaining, and terminating user accounts, passwords, roles, and privileges
- Developing applications that provide desired services securely in a variety of computational models, leveraging database and directory services to maximize both efficiency and client ease of use

To use this document, you need a basic understanding of how and why a database is used, as well as at least basic familiarity with SQL queries or programming.

## Organization

This document contains:

### Part I, "Overview of Security Considerations and Requirements"

Part I presents fundamental concepts of data security, and offers checklists and policies to aid in securing your site's data, operations, and users.

### Chapter 1, "Security Requirements, Threats, and Concepts"

This chapter presents fundamental concepts of data security requirements and threats.

### Chapter 2, "Security Checklists and Recommendations"

This chapter presents checklists, with brief explanations, for policies and practices that reduce your installation's vulnerabilities.

### Chapter 3, "Security Policies and Tips"

This chapter presents basic general security policies, with specific chapter references, that apply to every site. These you must understand and apply to the unique considerations of your own site. The chapter also introduces general application design practices regarding roles and privileges.

## Part II, "Security Features, Concepts, and Alternatives"

Part II presents methods and features that address the security requirements, threats, and concepts described in Part I.

### Chapter 4, "Authentication Methods"

This chapter deals with verifying the identity of anyone who wants to use data, resources, or applications. Authentication establishes a trust relationship for further interactions as well as accountability linking access and actions to a specific identity.

### Chapter 5, "Authorization: Privileges, Roles, Profiles, and Resource Limitations"

This chapter describes standard authorization processes that allow an entity to have certain levels of access and action, but which also limit the access, actions, and resources permitted to that entity.

### Chapter 6, "Access Controls on Tables, Views, Synonyms, or Rows"

This chapter discusses protecting objects by using object-level privileges and views, as well as by designing and using policies to restrict access to specific tables, views, synonyms, or rows. Such policies invoke functions that you design to specify dynamic predicates establishing the restrictions.

### Chapter 7, "Security Policies"

This chapter discusses security policies in separate sections dealing with system security, data security, user security, password management, and auditing. It concludes with a more detailed version of the checklist first presented in Chapter 2.

### Chapter 8, "Database Auditing: Security Considerations"

This chapter presents auditing as the monitoring and recording of selected user database actions. Auditing can be based either on individual actions, such as the type of SQL statement executed, or on combinations of factors that can include user name, application, time, and so on. Security policies can trigger auditing when specified elements in an Oracle database are accessed or altered, including the contents within a specified object.

### Part III, "Security Implementation, Configuration, and Administration"

Part III presents the details of setting up, configuring, and administering Oracle Database security features.

### Chapter 9, "Administering Authentication"

This chapter describes the methods for creating and administering authentication by defining users and how they are to be identified and verified before access is granted. Chapter 9 discusses the four primary methods as database, external, global, and proxy authentication.

### Chapter 10, "Administering User Privileges, Roles, and Profiles"

This chapter presents the interwoven tasks and considerations involved in granting, viewing, and revoking database user privileges and roles, and the profiles that contain them.

### Chapter 11, "Configuring and Administering Auditing"

This chapter describes auditing and accountability to protect and preserve privacy for the information stored in databases, detect suspicious activities, and enable finely-tuned security responses.

### Chapter 12, "Introducing Database Security for Application Developers"

This chapter provides an introduction to the security challenges that face application developers and includes an overview of Oracle Database features they can use to develop secure applications.

### Chapter 13, "Using Virtual Private Database to Implement Application Security Policies"

This chapter discusses developing secure applications by using application context, fine-grained access control, or virtual private database to implement security policies.

### Chapter 14, "Implementing Application Context and Fine-Grained Access Control"

This chapter provides several examples of applications developed using application context, fine-grained access control, and virtual private database. It includes code examples and their corresponding explanations.

### Chapter 15, "Preserving User Identity in Multitiered Environments"

This chapter discusses developing secure multiple tier applications.

## Related Documentation

For more information, see these Oracle resources:

- *Oracle Database Concepts*
- *Oracle Database Administrator's Guide*
- *Oracle Data Warehousing Guide*
- *Oracle Streams Advanced Queuing Java API Reference*
- *Oracle Streams Advanced Queuing User's Guide and Reference*

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

Printed documentation is available for sale in the Oracle Store at

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

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/membership/
```

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 for Windows Operating Systems

### 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. | *Oracle Database Concepts*<br>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.<br>You can back up the database by using the `BACKUP` command.<br>Query the `TABLE_NAME` column in the `USER_TABLES` data dictionary view.<br>Use the `DBMS_STATS.GENERATE_STATS` procedure. |

| Convention | Meaning | Example |
|---|---|---|
| `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.<br><br>**Note:** Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown. | Enter `sqlplus` to open SQL*Plus.<br><br>The password is specified in the `orapwd` file.<br><br>Back up the datafiles and control files in the `/disk1/oracle/dbs` directory.<br><br>The `department_id`, `department_name`, and `location_id` columns are in the `hr.departments` table.<br><br>Set the `QUERY_REWRITE_ENABLED` initialization parameter to `true.`<br><br>Connect as `oe` user.<br><br>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`*.<br><br>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}`<br>`[COMPRESS \| NOCOMPRESS]` |

| Convention | Meaning | Example |
|---|---|---|
| `. . .` | Horizontal ellipsis points indicate either:<br>▪ That we have omitted parts of the code that are not directly related to the example<br>▪ That you can repeat a portion of the code | `CREATE TABLE ... AS` *`subquery`*`;`<br><br>`SELECT` *`col1`*`,` *`col2`*`, ... ,` *`coln`* `FROM employees;` |
| .<br>.<br>. | Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example. | `SQL> SELECT NAME FROM V$DATAFILE;`<br>`NAME`<br>`------------------------------------`<br>`/fsl/dbs/tbs_01.dbf`<br>`/fs1/dbs/tbs_02.dbf`<br>`.`<br>`.`<br>`.`<br>`/fsl/dbs/tbs_09.dbf`<br>`9 rows selected.` |
| Other notation | You must enter symbols other than brackets, braces, vertical bars, and ellipsis points as shown. | `acctbal NUMBER(11,2);`<br>`acct    CONSTANT NUMBER(4) := 3;` |
| *`Italics`* | Italicized text indicates placeholders or variables for which you must supply particular values. | `CONNECT SYSTEM/`*`system_password`*<br>`DB_NAME =` *`database_name`* |
| `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;`<br>`SELECT * FROM USER_TABLES;`<br>`DROP TABLE hr.employees;` |
| `lowercase` | Lowercase typeface indicates programmatic elements that you supply. For example, lowercase indicates names of tables, columns, or files.<br>**Note:** Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown. | `SELECT last_name, employee_id FROM employees;`<br>`sqlplus hr/hr`<br>`CREATE USER mjones IDENTIFIED BY ty3MU9;` |

## Conventions for Windows Operating Systems

The following table describes conventions for Windows operating systems and provides examples of their use.

| Convention | Meaning | Example |
|---|---|---|
| Choose Start > | How to start a program. | To start the Database Configuration Assistant, choose Start > Programs > Oracle - *HOME_NAME* > Configuration and Migration Tools > Database Configuration Assistant. |
| File and directory names | File and directory names are not case sensitive. The following special characters are not allowed: left angle bracket (<), right angle bracket (>), colon (:), double quotation marks ("), slash (/), pipe (\|), and dash (-). The special character backslash (\) is treated as an element separator, even when it appears in quotes. If the file name begins with \\, then Windows assumes it uses the Universal Naming Convention. | `c:\winnt"\"system32` is the same as `C:\WINNT\SYSTEM32` |
| `C:\>` | Represents the Windows command prompt of the current hard disk drive. The escape character in a command prompt is the caret (^). Your prompt reflects the subdirectory in which you are working. Referred to as the *command prompt* in this manual. | `C:\oracle\oradata>` |
| Special characters | The backslash (\) special character is sometimes required as an escape character for the double quotation mark (") special character at the Windows command prompt. Parentheses and the single quotation mark (') do not require an escape character. Refer to your Windows operating system documentation for more information on escape and special characters. | `C:\>exp scott/tiger TABLES=emp QUERY=\"WHERE job='SALESMAN' and sal<1600\"`<br>`C:\>imp SYSTEM/`*`password`*` FROMUSER=scott TABLES=(emp, dept)` |
| *`HOME_NAME`* | Represents the Oracle home name. The home name can be up to 16 alphanumeric characters. The only special character allowed in the home name is the underscore. | `C:\> net start Oracle`*`HOME_NAME`*`TNSListener` |

| Convention | Meaning | Example |
|---|---|---|
| *ORACLE_HOME* and *ORACLE_BASE* | In releases prior to Oracle8*i* release 8.1.3, when you installed Oracle components, all subdirectories were located under a top level *ORACLE_HOME* directory that by default used one of the following names:<br>▪ `C:\orant` for Windows NT<br>▪ `C:\orawin98` for Windows 98<br>This release complies with Optimal Flexible Architecture (OFA) guidelines. All subdirectories are not under a top level *ORACLE_HOME* directory. There is a top level directory called *ORACLE_BASE* that by default is `C:\oracle`. If you install the latest Oracle release on a computer with no other Oracle software installed, then the default setting for the first Oracle home directory is `C:\oracle\ora`*nn*, where *nn* is the latest release number. The Oracle home directory is located directly under *ORACLE_BASE*.<br>All directory path examples in this guide follow OFA conventions.<br>Refer to *Oracle Database Platform Guide for Windows* for additional information about OFA compliances and for information about installing Oracle products in non-OFA compliant directories. | Go to the *ORACLE_BASE*`\`*ORACLE_HOME*`\rdbms\admin` directory. |

## 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 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 does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

# What's New in Oracle Database Security?

The Oracle Database 10*g* Release 1 (10.1) security features and enhancements described in this section comprise the overall effort to provide superior access control and accountability (privacy) with this release of the database.

The following sections describe new security features of Oracle Database 10*g* Release 1 (10.1) and provide pointers to additional information:

- New Features in Virtual Private Database
- New Features in Auditing
- New PL/SQL Encryption Package: DBMS_CRYPTO

## New Features in Virtual Private Database

To provide enhanced access control, privacy, and performance, the following enhancements have been added to Virtual Private Database (VPD), a feature of the Enterprise Edition:

- Column-level VPD and Column Masking

  *Column-level VPD* policies provides more fine-grained access controls on data. With column-level VPD, security policies can be applied only where a particular column or columns are accessed in the user's query. This means that when a user has rights to access the object itself, VPD can limit the individual rows returned only if the columns the user accesses contain sensitive information, such as salaries, or national identity numbers.

  The default behavior of column-level VPD restricts the number of rows returned when a query addresses columns containing sensitive data. In contrast, *column masking* behavior allows all rows to be returned for a query against data protected by column-level VPD, but the columns that contain sensitive information are returned as `NULL` values. With column masking, users see all the data they are supposed to see, but privacy is not compromised.

  **See Also:**

  - "Column-level VPD" on page 13-4 for conceptual information about these new features
  - "Adding Policies for Column-Level VPD" on page 14-40 for information about how to add column-level VPD to your applications

- Static, Context-Sensitive, and Shared VPD Policy Types

  Static and context-sensitive policy types optimize VPD for significant performance improvements because the policy function does not execute for every SQL query. *Static policies* maintain the same predicate for queries, updates, inserts, and deletes throughout a session. (However application context or attributes such as `SYSDATE` can change the value returned by the predicate.) They are particularly useful for hosting environments where you always need to apply the same predicate. With *context-sensitive policies*, the predicate can change after statement parse time, but VPD re-executes the policy function only if the application context changes. This ensures that any changes to the predicate since the initial parsing are captured. Context-sensitive policies are useful when VPD policies must enforce two or more different predicates for different users or groups.

Both static and context-sensitive policies can be shared across multiple database objects, so that queries on another database object can use the same cached predicate. Shared policies enable you to further decrease the overhead of re-executing policy functions for every query, reducing any performance impact.

> **See Also:** "DBMS_RLS.ADD_POLICY Procedure Policy Types" on page 14-36 for more information about these new policy types and how to use them in applications.

- Application context support for parallel queries

  In this release, if you use `SYS_CONTEXT` inside a SQL function which is embedded in a parallel query, the function picks up the application context.

  > **See Also:** "Using SYS_CONTEXT in a Parallel Query" on page 14-5 for information about using this enhancement.

## New Features in Auditing

Oracle Database 10*g* Release 1 (10.1) expands upon standard and fine-grained auditing for enhanced user accountability, providing the following new features:

- Fine-grained Auditing Support for DML

  In the previous release, fine-grained auditing support was only available for `SELECT` statements. In this release, fine-grained auditing support is expanded to include DMLs (`INSERT`, `UPDATE`, `DELETE`).

  > **See Also:** "Fine-Grained Auditing" on page 11-29 for more information about this new feature

- Uniform Audit Trail

  In this release the `DBA_COMMON_AUDIT_TRAIL` view has been added, which presents both the standard and the fine-grained audit log records in a single view.

  > **See Also:** "Database Audit Trail Contents" on page 11-8 for more information about this new view.

- Extensions to Standard Audit and Fine-Grained Auditing

  Fields have been added to the standard and the fine-grained audit trails in this release. New fields capture the exact SQL text of audited statements, the date and time stamp in UTC (Coordinated Universal Time) format, and enhanced auditing for enterprise users. Enterprise users are global database users, who are stored in and LDAP directory. In this release, the audit trail includes enterprise users' full distinguished names (DNs) and global user identifiers (GUIDs).

  > **See Also:** "What Information is Contained in the Audit Trail?" on page 11-7 for more information about the extensions to standard and fine-grained audit trails

## New PL/SQL Encryption Package: DBMS_CRYPTO

In this release, a new flexible interface, `DBMS_CRYPTO`, is provided to encrypt especially sensitive stored data, or it can also be used in conjunction with PL/SQL programs running network communications. This new interface provides support for the following features:

- Encryption algorithms as follows:
  - AES (Advanced Encryption Standard)
  - Triple DES (112- and 168-bits)
  - DES
  - RC4
- Cryptographic hash algorithms (SHA-1, MD5, and MD4)
- Keyed hash (MAC, or Message Authentication Code) algorithms (SHA-1, MD5)
- Padding forms (PKCS #5, zeroes)
- Block cipher chaining mode modifiers (CBC, CFB, ECB, OFB)

`DBMS_CRYPTO` is intended to replace the `DBMS_OBFUSCATION_TOOLKIT`, providing greater ease of use and support for a range of algorithms to accommodate new and existing systems.

> **See Also:** Chapter 16, "Developing Applications Using Data Encryption" for information about how to use this package

# Part I

# Overview of Security Considerations and Requirements

Part I presents fundamental concepts of data security requirements and threats that pertain to connecting to a database, accessing and altering tables, and using applications. In addition, security checklists are provided for DBAs and application developers, which cover installation preparation, database administration best practices, and recommendations for developing secure applications.

This part contains the following chapters:

This part also contains high-level security checklists for DBAs and application developers, covering preparations for installation, best practices for administration, and recommended practices for developing secure applications. References are included, pointing to the explanations and alternatives presented in Part II and the examples described in Part III.

# 1 Security Requirements, Threats, and Concepts

Database security requirements arise from the need to protect data: first, from accidental loss and corruption, and second, from deliberate unauthorized attempts to access or alter that data. Secondary concerns include protecting against undue delays in accessing or using data, or even against interference to the point of denial of service. The global costs of such security breaches run to billions of dollars annually, and the cost to individual companies can be severe, sometimes catastrophic.

These requirements are dynamic. New technologies and practices continually provide new arenas for unauthorized exploitation, as well as new ways for accidental or deliberate misuse to affect even stable products and environments. Today's evolution involves a globally changing technological and cultural environment, in which security concerns necessarily affect both the use of existing solutions and the development of new ones.

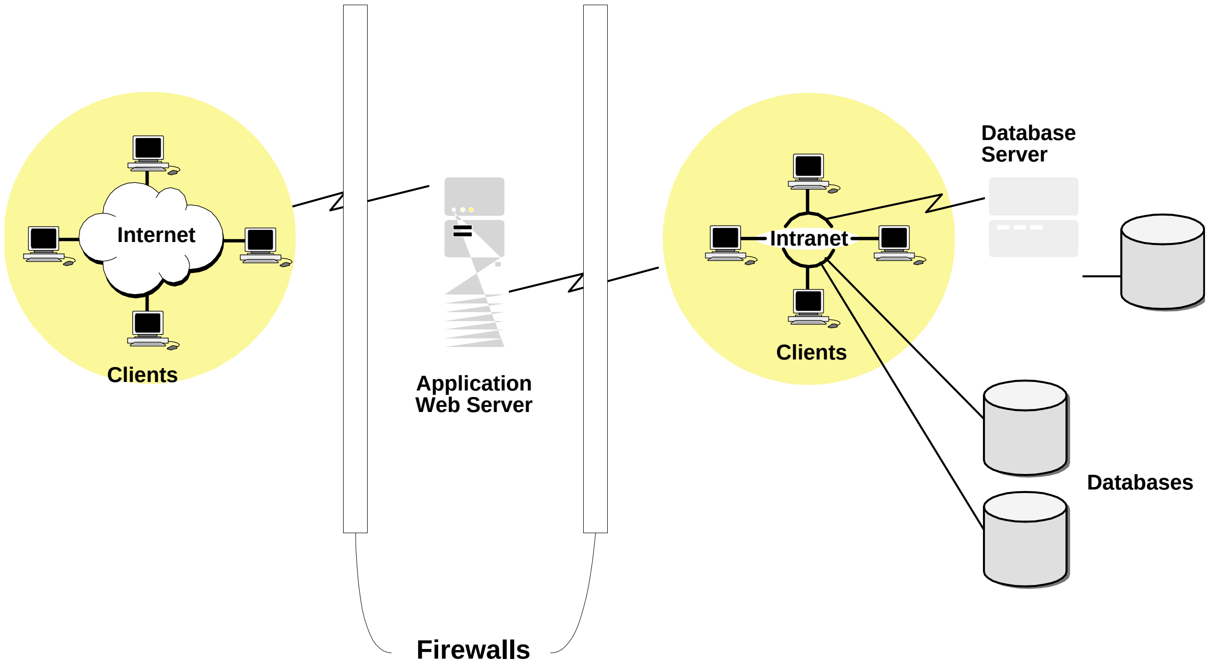
As security requirements are understood with increasing clarity, certain general principles can be developed for satisfying them and for disabling the threats against them deriving from Internet vulnerabilities. Of course, principles can vary in effectiveness. Their implementations usually vary in cost: hardware and software acquisition and maintenance, administrative and programming personnel, and the impact of security measures on processing time and response time. Total cost also includes the costs of managing each of these areas — hardware, software, personnel, efficiency, responsiveness, and so on — management costs that can escalate with increasing volumes of users, transactions, and data types.

This book will address most of those issues, conceptually at first, and with increasing detail as it moves toward discussing specific choices and implementations.

The basic elements and operations of the database environment include connection to a server or a schema, table access and alteration, and application usage. Securing these against accidental or deliberate misuse is the responsibility of security officers, administrators, and application programmers. They must also administer and protect the rights of internal database users, and guarantee electronic commerce confidentiality as customers access databases from anywhere the Internet reaches.

In the Internet age, the full spectrum of risks to valuable and sensitive data, and to user access and confidentiality, is broader than ever before. Figure 1–1 shows the complex computing environment that security plans must protect.

***Figure 1–1 Realms Needing Protection in an Internet World***



The diagram shows several important parts of the security picture, illustrating client communities, connections, databases, and servers, all of which must be secured against inappropriate access or use. These different areas can require different techniques to achieve good security, and they must integrate so as to preclude or minimize security gaps or vulnerabilities.

Table 1–1 provides separate categories that are usable in focusing efforts to create secure operations in a secure environment. When these necessary components are consistent in their security focus, coherent in the ways they work together, and made complete by closing all known channels of attack and misuse, your security is as good as it gets.

These categories reappear in later chapters in discussing checklists and best practices.

***Table 1–1 Security Issues by Category***

| Dimension | Security Issues |
|---|---|
| Physical | Your computers must be made physically inaccessible to unauthorized users by keeping them in a secure physical environment. |
| Personnel | The people responsible for your site's physical security, system administration, and data security must be reliable. Performing background checks on DBAs before making hiring decisions is a wise protective measure. |
| Procedural | The procedures and policies used in the operation of your system must assure reliable data. It is often wise to separate out users' functional roles in data management.<br><br>For example, one person might be responsible for database backups. Her only role is to be sure the database is up and running.<br><br>Another person might be responsible for generating application reports involving payroll or sales data. His role is to examine the data and verify its integrity.<br><br>Further, you can establish policies that protect tables and schemas against unauthorized, accidental, or malicious usage. |
| Technical | Storage, access, manipulation, and transmission of data must be safeguarded by technology that enforces your particular information control policies. |

When you think carefully about security risks, the solutions you adopt will apply well to the actual situation you're addressing; not all security problems have a technical fix. For example, employees must occasionally leave their desks unattended. Depending on the sensitivity of their work and on your required level of security, your security procedures could require them to do any of the following

- have another person cover for them while they're away
- clear the desk surface, locking all sensitive materials away, before leaving
- lock their doors, if they have private offices
- explicitly lock their computer screens before leaving the desk

No technical solution can fix a physically insecure work environment or a corrupt or disaffected employee. It is true, though, that procedural and technical protections might be able to limit the damage that a physical breach or a disgruntled employee (or ex-employee) can inflict.

## Identity Management: Security in Complex, High Volume Environments

In addition to the general issues and categories of security, the sheer numbers of people and activities requiring a security focus add the dimension of complexity.

As the number of users, databases, applications, and networks grows from "a few" to dozens, hundreds, or thousands, the complexity of their interactions rises exponentially. So, too, do the risks, and the management tasks required to maintain security and efficiency.

For example, the number of interactions for ten users accessing any of five databases is potentially 50. Add 90 users and 45 databases, and you have 100 users accessing any of 50 databases, with potential interactions at 5,000. When you add to this example some number of applications and of networks, you begin to see extreme complexity, with directly proportional security risks. A security breach anywhere in a network can threaten the security of its databases and users, and that of other connected networks, databases, and users as well.

This type of complex environment demands speed and flexibility in granting or revoking access rights for any user and any resource. Delays in administrative processes, or their implementation on the corresponding databases, translate either to legitimate access delayed or to access granted when it should have been denied.

For example, when an employee with access to multiple applications and accounts on multiple databases leaves the company, his access to those applications and accounts should stop instantly. Achieving this can be difficult when administrative control and responsibility is distributed across nodes in the network and among different administrators and groups.

But what if an intelligent central repository could efficiently control data regarding identity, accounts, authentication, and authorization and rapidly communicate any needed information to any node or application? Then one change (or few) in one place could control the entirety of the departing employee's access rights and privileges. The assumption is that the intelligence imputed to this central repository and its software takes care of all such considerations and connections. Of course, all related system, database, and application routines would need to adjust to relying upon that central repository as the "single source of truth" regarding such information.

These considerations are the basis for an integrated Identity Management solution. Its components and integration are described in subsequent chapters, as the need for them becomes explicit in the context of general security functionality and dimensions.

> **See Also:** For a full discussion of Oracle's Identity Management Infrastructure, see the *Oracle Identity Management Concepts and Deployment Planning Guide*

As an overview, however, the following subsections provide an introduction:

- Desired Benefits of Identity Management
- Components of Oracle's Identity Management Infrastructure

## Desired Benefits of Identity Management

The goal of identity management is to create

- greater security, because a single point of control is inherently easier to secure and more responsive than multiple such points, and
- greater efficiency, because a single point of control automatically eliminates the duplication and delays inherent in the need for multiple actions to handle dispersed administrative responsibilities regarding the same accounts.

However, cost and complexity remain relevant measures of the viability of any such solution, which ideally should provide the following reductions in resource allocations:

- **One-time cost:** Planning and implementing the identity management infrastructure should be a one-time cost, rather than a necessary part of each enterprise application deployment. In this way, new applications can be rapidly deployed, automatically benefitting from the infrastructure, but without having to re-create it. Examples could include portals, J2EE applications, and e-business applications.

- **Centralized management with distributable tools:** Provisioning and managing identities should be done centrally, even if administered in multiple places using tools that can handle any account alteration needed.
- **Seamless timely distribution:** Changes to user accounts, profiles, or privileges should be instantly available to all enterprise applications and rapidly communicated to distributed databases.
- **User single sign-on:** The centralized security infrastructure should make it possible to realize user single sign-on across enterprise applications. Single sign-on makes it unnecessary for users to remember multiple passwords and for security administrators to protect multiple password repositories and provisioning mechanisms.
- **Single point of integration:** The centralized identity management infrastructure should provide a single point of integration between the enterprise environment and other identity management systems. It should eliminate any need for multiple custom "point-to-point" integration solutions.

An identity management solution meeting all these criteria at a high level would provide an enterprise with high availability, information localization, and delegated component administration. Further, each additional application deployed in that enterprise would then leverage the shared infrastructure for identity management services.

As a real-world example, the Oracle Identity Management Infrastructure uses integrated components to provide those benefits, as described in the next section.

## Components of Oracle's Identity Management Infrastructure

The Oracle Identity Management infrastructure includes the following components and capabilities:

- **Oracle Internet Directory**, a scalable, robust LDAP V3-compliant directory service implemented on the Oracle9i Database.
- **Oracle Directory Integration and Provisioning**, part of Oracle Internet Directory, which enables synchronization between Oracle Internet Directory and other directories and user repositories. This service also provides automatic provisioning services for Oracle components and applications and, through standard interfaces, for third-party applications.

- **Oracle Delegated Administration Service**, part of Oracle Internet Directory, which provides trusted proxy-based administration of directory information by users and application administrators.
- **OracleAS Single Sign-On**, which provides single sign-on access to Oracle and third party web applications.
- **OracleAS Certificate Authority**, which generates and publishes X.509 V3 PKI certificates to support strong authentication methods, secure messaging, and so on.

  In addition to its use of SSL (Secure Socket Layer), OracleAS Containers for J2EE, and Oracle HTTP Server, Oracle's Identity Management infrastructure has a built-in reliance on OracleAS Single Sign-On and Oracle Internet Directory. When OracleAS Certificate Authority is in use, it publishes each valid certificate in a directory entry for the DN in use. The Single Sign-On server and other components can rely on these directory entries because the certificate authority removes revoked and expired certificates from the directory on a regular basis. Users who are authenticated by the Single Sign-On server, and who lack a certificate can rapidly acquire one directly from OracleAS Certificate Authority, enabling them to authenticate to any Oracle component or application that is configured to authenticate users with the Single Sign-On server.

In a typical enterprise application deployment, a single Oracle Identity Management infrastructure is deployed, consisting of multiple server and component instances. Such a configuration in fact provides the high availability, information localization, and delegated component administration benefits mentioned earlier.

2

# Security Checklists and Recommendations

This chapter gives you a broad overview of the many types of tasks you must confront in order to build good security. Understanding the diverse categories of such tasks improves your likelihood of preventing security gaps. Such gaps, whether exploited accidentally or intentionally, can undermine or overwhelm otherwise tight security that you have created in other areas.

Chapter 1 introduced the requirements for good security, the threats against it, and concepts that have proven useful in creating practical methods for developing and sustaining it.

The overview presented here, in this chapter, identifies categories of tasks useful in meeting those requirements and threats. This chapter presents brief descriptions of these categories and tasks, with cross-references to Parts 2 and 3, where the important details necessary to their implementation are described.

Good security requires physical access control, reliable personnel, trustworthy installation and configuration procedures, secure communications, and control of database operations such as selection, viewing, updating, or deleting database records. Since some of these requirements involve applications or stored procedures as well as human action, security procedures must also account for how these programs are developed and dealt with.

Practical concerns must also be met: minimizing the costs of equipment, personnel, and training; minimizing delays and errors; and maximizing rapid and thorough accountability. Scalability, too, is an important and independent practical criterion that should be assessed for each proposed solution.

These, then, are the categories with which this overview is concerned. They are discussed in the following sections:

- Physical Access Control Checklist
- Personnel Checklist

- Secure Installation and Configuration Checklist
- Networking Security Checklists

## Physical Access Control Checklist

It shouldn't be easy to walk into your facility without a key or badge, or without being required to show identity or authorization. Controlling physical access is your first line of defense, protecting your data (and your staff) against the simplest of inadvertent or malicious intrusions and interferences.

Lack of such control can make it easier to observe, copy, or steal your other security controls, including internal keys, key codes, badge numbers or badges, and so on. Of course, the security of these measures, too, depends on how alert and security conscious each of your staff is, but physical access control stops a variety of potential problems before they even get started.

Each organization must evaluate its own risks and budget. Elaborate measures may well not be needed, depending on many factors: company size, risk of loss, internal access controls, quantity and frequency of outside visitors, and so on. Preparing for accountability and recovery are additional considerations, possibly prompting alarms or video surveillance of entryways. The visibility of these preparations can also act as deterrence.

Improving your facility's physical access control can add to your security. Make it hard to get in, hard to remain or leave unobserved or unidentified, hard to get at sensitive or secure areas inside, and hard not to leave a trace.

## Personnel Checklist

Your staff makes your organization work, well or poorly depending on who they are and how they are managed. Your security is critically dependent on them: first, on how honest and trustworthy they are, and second, on how aware and alert they are to security concerns and considerations. The first issue is a matter of selection, interviewing, observation, and reference checking. Done well, these skills can prevent your hiring people who are (or are likely to become) inappropriate for tasks or environments that depend on establishing and maintaining security. To a very large degree, security depends on individuals: when they get careless, resentful, or larcenous, tight security loosens or disappears. Your other measures won't matter if they are carelessly or deliberately undermined or sabotaged.

The second issue is how aware and alert your staff is to security concerns and considerations. Such consciousness is only partly a matter of background: the environment and training you provide are the most significant influences, given basic honesty and intent to cooperate. When an organization both shows and says that security is important, by establishing and enforcing security procedures and by providing training and bulletins about it, people learn and adapt. The result is better security and safety for them as well as for the organization's data and products.

## Secure Installation and Configuration Checklist

Information security, privacy, and protection of corporate assets and data are of critical importance to every business. For databases, establishing a secure configuration is a very strong first line of defense, using industry-standard "best security practices" for operational database deployments. The following list of such practices is deliberately general to remain brief. Additional details for each recommendation as it applies to Oracle Database appear in Chapter 7, "Security Policies". Further specific descriptions of database-related tasks and actions can be found throughout the Oracle documentation set.

Implementing the following ten recommendations provides the basis for a secure configuration:

1. **Install Only What Is Required.**

   Do a custom installation. Avoid installing options and products you don't need. Choose to install only those additional products and options, in addition to the database server, that you do clearly need. Or, if you choose to do a "typical" installation instead, improve your security after the installation processes finish, by deinstalling the options and products you don't need.

2. **Lock And Expire Default User Accounts.**

   The Oracle Database installs with many default (preset) database server user accounts. Upon the successful creation of a database server instance, the Database Configuration Assistant automatically locks and expires most default database user accounts.

   ---

   **Note:** If you use Oracle Universal Installer or Database Configuration Assistant, they will prompt for new SYS and SYSTEM passwords, and will not accept the defaults "change_on_install" or "manager", respectively.

   ---

Once the database is installed, lock SYS and SYSTEM as well, and use AS SYSDBA for administrator access. Specify administrative passwords individually.

This account (AS SYSDBA) tracks the operating system username, maintaining accountability. If you only need access for database startup and shutdown, use AS SYSOPER instead. SYSOPER has fewer administrative privileges than SYS, but enough to perform basic operations such as startup/shutdown, mount, backup, archive, and recover.

Database Configuration Assistant is not used during a manual installation, so all default database users remain unlocked and able to gain unauthorized access to data or to disrupt database operations. Therefore, after a manual installation, use SQL to *lock* and *expire* all default database user accounts except `SYS`, `SYSTEM`, `SCOTT`, and `DBSNMP`. (Do it to SCOTT, too, unless it is being actively used. Also lock SYS and SYSTEM as described earlier.) If a locked account is later needed, a database administrator can simply unlock and activate that account with a new, meaningful password.

3. **Change Default User Passwords.**

   Security is most easily broken when a default database server user account still has a default password *even after installation*. Three steps fix this:

   a. Change the default passwords of administrative users immediately after installing the database server.

      In any Oracle environment (production or test), assign strong, meaningful passwords to the SYS and SYSTEM user accounts immediately upon successful installation of the database server. Under no circumstances should the passwords for SYS and SYSTEM remain in their default states. Similarly, for production environments, do not use default passwords for any administrative accounts, including SYSMAN and DBSNMP.

   b. Change the default passwords of all users immediately after installation.

      Lock and expire all default accounts after installation. If any such account is later activated, change its default password to a new meaningful password.

   c. Enforce password management.

      Apply basic password management rules, such as password length, history, and complexity, to all user passwords.

      Require all users to change their passwords regularly, such as every eight weeks.

If possible, use Oracle Advanced Security (an option to the Enterprise Edition of Oracle Database) with network authentication services (such as Kerberos), token cards, smart cards, or X.509 certificates. These services provide strong user authentication and enable better protection against unauthorized access.

4. **Enable Data Dictionary Protection.**

   Implement data dictionary protection to prevent users who have the `ANY` system privilege from using it on the data dictionary. Oracle Database sets the `O7_DICTIONARY_ACCESSIBILITY` to `FALSE`. This setting prevents using the `ANY` system privilege on the data dictionary, except for authorized users making DBA-privileged connections (for example `CONNECT/AS SYSDBA`).

5. **Practice The Principle Of Least Privilege.**

   Three practices implement this principle:

   a. Grant necessary privileges only.

      Do not provide database users more privileges than necessary. Enable only those privileges actually required to perform necessary jobs efficiently:

      1) Restrict the number of system and object privileges granted to database users, and

      2) Restrict the number of `SYS`-privileged connections to the database as much as possible. For example, there is generally no need to grant `CREATE ANY TABLE` to any non DBA-privileged user.

   b. Revoke unnecessary privileges and roles from the database server user group `PUBLIC`.

      This default role, granted to every user in an Oracle database, enables unrestricted use of its privileges, such as `EXECUTE` on various PL/SQL packages. If unnecessary privileges and roles are not revoked from PUBLIC, a minimally privileged user could access and execute packages otherwise inaccessible to him. The more powerful packages that may potentially be misused are listed in Chapter 7, "Security Policies".

   c. Restrict permissions on run-time facilities.

      Do not assign "all permissions" to any database server run-time facility, such as the Oracle Java Virtual Machine (OJVM).

      Instead, grant specific permissions to the explicit document root file paths for such facilities that may execute files and packages outside the database server. Examples are listed in Chapter 7, "Security Policies".

6. **Enforce Access Controls Effectively.**

   Authenticate clients properly.

   Although remote authentication can be turned on (TRUE), your installation is more secure with it off (FALSE, which is the default). With remote authentication turned on, the database implicitly trusts every client, because it assumes every client was authenticated by the remote authenticating system. However, clients in general (such as remote PCs) cannot be trusted to perform proper operating system authentication, so turning on this feature is a very poor security practice. To enforce proper server-based authentication of clients connecting to an Oracle database, leave or turn this feature off (remote_os_authentication=FALSE, which is the default).

7. **Restrict Operating System Access.**

   Four practices implement appropriate restrictions on operating system access:

   a. Limit the number of operating system users.

   b. Limit the privileges of the operating system accounts (administrative, root-privileged or DBA) on the Oracle Database host (physical machine) to the fewest and least powerful privileges required for each user.

   c. Disallow modifying the default permissions for the Oracle Database home (installation) directory or its contents, even by privileged operating system users or the Oracle owner.

   d. Restrict symbolic links. Ensure that when any path or file to the database is provided, neither that file nor any part of that path is modifiable by an untrusted user. The file and all components of the path should be owned by the DBA or some trusted account, such as root. This recommendation applies to all types of files: data files, log files, trace files, external tables, bfiles, and so on.

8. **Restrict Network Access.**

   (See Networking Security Checklists later in this chapter for appropriate practices.)

9. **Apply All Security Patches And Workarounds.**

   Plug every security hole or flaw as soon as corrective action is identified. Always apply all relevant and current security patches for both the host operating system and Oracle Database itself, and for all installed Oracle Database options and components.

Periodically check the security site on Oracle Technology Network for details on security alerts released by Oracle Corporation:

```
http://otn.oracle.com/deploy/security/alerts.htm
```

Also check Oracle Worldwide Support Service's site, Metalink, for details on available and upcoming security-related patches:

```
http://metalink.oracle.com
```

10. **Contact Oracle Security Products.**

    If you believe that you have found a security vulnerability in Oracle Database, submit an iTAR to Oracle Worldwide Support Services using Metalink, or e-mail a complete description of the problem, including product version and platform, together with any exploit scripts and examples, to the following address:

    ```
    secalert_us@oracle.com
    ```

## Networking Security Checklists

Security for network communications is improved by using client, listener, and network checklists to create thoroughgoing protection. Using SSL (Secure Sockets Layer) is an essential element in these lists, enabling top security for authentication and communications.

### SSL (Secure Sockets Layer) Checklist

SSL is the Internet standard protocol for secure communication, providing mechanisms for data integrity and data encryption. These mechanisms can protect the messages sent and received by you or by applications and servers, supporting secure authentication, authorization, and messaging by means of certificates and, if necessary, encryption. Good security practices maximize all these protections and minimize gaps or disclosures that threaten them. While the primary documentation for Oracle's SSL configuration and practices is *Oracle Advanced Security Administrator's Guide*, the following basic list illustrates the cautious attention to detail necessary for successful, secure use of SSL:

1. Ensure that configuration files (such as for clients and listeners) use the correct port for SSL, which is the port configured upon installation. You can run HTTPS on any port, but the standards specify port 443, where any HTTPS-compliant browser looks by default. Or the port can be specified in the URL, for example, (for port 4445): `https://secure.server.dom:4445/`

If a firewall is in use, it too must use the same port(s) for secure (SSL) communication.

2. Ensure that tcps is specified as the PROTOCOL in the ADDRESS parameter in the tnsnames.ora file (typically on the client or in the LDAP directory). The identical specification must appear in the listener.ora file (typically in the $ORACLE_HOME/network/admin directory).
3. Ensure that the SSL mode is consistent for both ends of every communication. For example, between the database on one side and the user or application on the other. The mode can specify that there be client or server authentication only (one-way), both client and server authentication (two-way), or no authentication.
4. Ensure that the server supports the client cipher suites and the certificate key algorithm that will be in use.
5. Do not remove the encryption from your RSA private key inside your `server.key` file, which requires that you enter your pass-phrase to read and parse this file. (A non-SSL-aware server does not require such a phrase.) However, if you were to decide your server is secure enough, you could remove the encryption from the RSA private key while preserving the original file. This would enable system boot scripts to start the server, because no pass-phrase would be needed. However, be very sure that permissions on the `server.key` file allow only root or the web server user to read it. Ideally, restrict permissions to root alone, and have the web server start as root but run as another server. Otherwise, anyone who gets this key can impersonate you on the net.

**See Also:**

- For general SSL information, including configuration, see the *Oracle Advanced Security Administrator's Guide*.
- For tcps information in particular, see *Oracle Net Services Administrator's Guide* and *Oracle Net Services Reference Guide*.

## Client Checklist

Since authenticating client computers is problematic over the Internet, user authentication is typically done instead. This approach avoids client system issues that include falsified IP addresses, hacked operating systems or applications, and falsified or stolen client system identities. Nevertheless, the following steps improve the security of client connections:

1. Configure the connection to use SSL.

Using SSL (Secure Sockets Layer) communication makes eavesdropping unfruitful and enables the use of certificates for user and server authentication.

2. Set up certificate authentication for clients and servers.

## Listener Checklist

Because the listener acts as the database's gateway to the network, it is important to limit the consequences of malicious interference:

1. Restrict the privileges of the listener, so that it cannot read or write files in the database or the Oracle server address space.

   This restriction prevents external procedure agents spawned by the listener (or procedures executed by such an agent) from inheriting the ability to do such reads or writes. The owner of this sepasrate listener process should not be the owner that installed Oracle or executes the Oracle instance (such as ORACLE, the default owner).

2. Secure administration by the following four steps:

   **See Also:** See the section A Security Checklist in Chapter 7, "Security Policies", for more specific details.

   a. Password protect the listener.

   b. Prevent on-line administration.

   c. Use SSL when administering the listener.

   d. Remove the external procedure configuration from the listener.ora file if you do not intend to use such procedures.

3. Monitor listener activity.

## Network Checklist

Protecting the network and its traffic from inappropriate access or modification is the essence of network security. The following practices improve network security:

a. Restrict physical access to the network. Make it hard to attach devices for listening to, interfering with, or creating communications.

b. Protect the network access points from unauthorized access. This goal includes protecting the network-related software on the computers, bridges, and routers used in communication.

c. Since you cannot protect physical addresses when transferring data over the Internet, use encryption when this data needs to be secure.

d. Use firewalls.

   Appropriately placed and configured firewalls can prevent outsider access to your organization's intranet when you allow internal users to have Internet access.

   * Keep the database server behind a firewall. Oracle Database's network infrastructure supports a variety of firewalls from various vendors; examples are listed in Chapter 7, "Security Policies".

   * Ensure that the firewall is placed outside the network to be protected.

   * Configure the firewall to accept only those protocols, applications, or client/server sources that you know are safe.

   * Use a product like Oracle Connection Manager to multiplex multiple client network sessions through a single network connection to the database. It can filter on source, destination, and host name. This functionality enables you to ensure that connections are accepted only from physically secure terminals or from application Web servers with known IP addresses. (Filtering on IP address alone is not enough for authentication, because it can be faked.)

e. Never poke a hole through a firewall.

   For example, do not leave open Oracle Listener's 1521 port, allowing the database to connect to the Internet or the Internet to connect with the database. Such a hole introduces significant security vulnerabilities that hackers are likely to exploit. They can enable even more port openings through the firewall, create multi-threaded operating system server problems, and gain access to crucial information on databases behind the firewall. If the Listener is running without an established password, they can probe for critical details about the databases on which it is listening. These details include trace and logging information, banner information, and database descriptors and service names, enabling malicious and damaging attacks on the target databases.

f. Prevent unauthorized administration of the Oracle Listener.

   Always establish a meaningful, well-formed password for the Oracle Listener, to prevent remote configuration of the Oracle Listener. Further, prevent unauthorized administration of the Oracle Listener, as described in Chapter 7, "Security Policies".

g. Check network IP addresses.

   Use the Oracle Net "valid node checking" security feature to allow or deny access to Oracle server processes from network clients with specified IP addresses. Set parameters in the `protocol.ora` file (Oracle Net configuration file) to specify client IP addresses respectively denied or allowed connections to the Oracle Listener. This action prevents potential Denial of Service attacks.

h. Encrypt network traffic.

   If possible, utilize Oracle Advanced Security to encrypt network traffic between clients, databases, and application servers. (Note that Oracle Advanced Security is available only with the Enterprise Edition of the Oracle database).

i. Harden the host operating system (the system on which Oracle Database resides).

   Disable all unnecessary operating system services. Many UNIX and Windows services are not necessary for most deployments. Such services include FTP, TFTP, TELNET, and so forth.

   For each disabled service, be sure to close both the UDP and TCP ports. Leaving either type of port enabled leaves the operating system vulnerable.

In summary, consider all paths the data travels and assess the threats that impinge on each path and node. Then take steps to lessen or eliminate those threats and the consequences of a successful breach of security. Also monitor and audit to detect either increased threat levels or successful penetration.

# 3 Security Policies and Tips

The idea of security policies includes many dimensions. Broad considerations include requiring regular backups to be done and stored off-site. Narrow table or data considerations include ensuring that unauthorized access to employee salaries is precluded by built-in restrictions on every type of access to the table that contains them.

This chapter introduces ideas about security policies and offers tips about recommend practices that can tighten security, in the following sections:

- Introduction to Database Security Policies
- Recommended Application Design Practices to Reduce Risk

## Introduction to Database Security Policies

This section briefly introduces security policies. It covers:

- Security Threats and Countermeasures
- What Information Security Policies Can Cover

### Security Threats and Countermeasures

An organization should create a written security policy to enumerate the security threats it is trying to guard against, and the specific measures the organization must take. Security threats can be addressed with different types of measures:

- Procedural, such as requiring data center employees to display security badges
- Physical, such as securing computers in restricted-access facilities
- Technical, such as implementing strong authentication requirements for critical business systems

- Personnel-related, such as performing background checks or "vetting" key personnel

Consider whether the appropriate response to a threat is procedural, physical, technical, personnel-related, or a combination of the such measures.

For example, one possible security threat is disruption of critical business systems caused by a malicious person damaging a computer. A physical response to this threat is to secure key business computers in a locked facility. A procedural response is to create system backups at regular intervals. Personnel measures could include background checks on employees who access or manage key business systems.

Oracle Database offers many mechanisms you can use to implement the *technical* measures of a good security policy. Chapter 7, "Security Policies" and Chapter 13, "Using Virtual Private Database to Implement Application Security Policies".

## What Information Security Policies Can Cover

In addition to addressing requirements unique to your environment, you should also design and implement technical measures in your information security policies to address important generic issues, such as the following concerns:

***Table 3–1 Issues and Actions for Policies to Address***

| **Security Concern/Practice** | **Recommended Actions** | **References** |
|---|---|---|
| Establish & maintain application-level security | Attach privileges and roles to each application. | See Ref [1] |
| | Ensure that users cannot misuse those roles and privileges when they are not using the application. | |
| | Base use of roles on user-defined criteria, such as a user connecting only from a particular IP address, or only through a particular middle tier. | |
| Manage privileges & attributes (system / object / user) | Permit only certain users to access, process, or alter data, including the rights to execute a particular type of SQL statement or to access another user's object. | See Ref [2] |
| | Apply varying limitations on users' access to or actions on objects, such as schemas, tables, or rows, or resources, such as time (CPU, connect, or idle times). | |
| Create, manage, and control roles (database, enterprise) | Create named groups of privileges to facilitate granting them to users, including previously named groups (roles). | See Ref [3] |

***Table 3–1 (Cont.) Issues and Actions for Policies to Address***

| Security Concern/Practice | Recommended Actions | References |
|---|---|---|
| Establish the granularity of access control desired | Set up session-based attributes securely. For example, store user attributes (username, employee number, and so on) to be retrieved later in the session, enabling fine-grained access control. | See Ref [4] |
| | Create security policy functions & attach them to critical or sensitive tables, views, or synonyms used by an application. DML statements on such objects are then modified dynamically, and transparently to the user, to preclude inappropriate access. | |
| | Enforce fine-grained or label-based access control automatically with policy functions or data & user labels, quickly limiting access to sensitive data, often without additional programming | |
| Establish & manage the use of encryption | Use Secure Socket Layer (SSL) connections, well-established encryption suites, or PKI certificates for critical / sensitive transmissions / applications. | See Ref [5] |
| Establish & maintain security in 3-tier applications | Preserve user identity through a middle tier to the database. | See Ref [6] |
| | Avoid the overhead of separate database connections by proxying user identities (and credentials like a password or certificate) through the middle tier to the database. | |
| Control query access, data misuse, and intrusions | Monitor query access based on specific content or row to detect data misuse or intrusions. | See Ref [7] |
| | Use proxy authentication to support auditing of proxied user connections. | |
| | Use Regular Auditing and Fine Grained Auditing to detect unauthorized or inappropriate access or actions. | |

[1] <<<role independence and Application Security>>> "Creating Secure Application Roles" on page 12-5

[2] <<<privileges & attributes (system / object / user)>>> Introduction to Privileges on page 5-2, Understanding User Privileges and Roles on page 10-15, & Granting User Privileges and Roles on page 10-24

[3] <<<Create, manage, and control roles (database, enterprise)>>> "Introduction to Roles" on page 5-19 & Managing User Roles on page 10-20

[4] <<<Establish the granularity of access control desired)>>>Chapter 7, "Security Policies", Chapter 13, "Using Virtual Private Database to Implement Application Security Policies" & Chapter 14, "Implementing Application Context and Fine-Grained Access Control"

[5] <<<Establish & manage the use of encryption>>>Chapter 4, "Authentication Methods"

[6] Insert Reference for <<<Establish & maintain security in 3-tier applications>>>Chapter 14, "Implementing Application Context and Fine-Grained Access Control" & Chapter 15, "Preserving User Identity in Multitiered Environments"

[7] <<<Control query access, data misuse, and intrusions>>>Chapter 14, "Implementing Application Context and Fine-Grained Access Control" & Chapter 15, "Preserving User Identity in Multitiered Environments"; for auditing, Chapter 8, "Database Auditing: Security Considerations"and Chapter 11, "Configuring and Administering Auditing"

The security practices and recommended actions of Table 3–1 are readily implemented using the Oracle features, facilities, and products listed in Table 3–2, in alphabetical order. Discussions of these terms and products appear in the corresponding chapters (or book) listed in that table:

*Table 3–2 References Terms and Chapters for Oracle Features and Products*

| Reference Terms | Reference Chapters |
|---|---|
| Application Context | Chapter 14, "Implementing Application Context and Fine-Grained Access Control" |
| Data Encryption | Chapter 16, "Developing Applications Using Data Encryption" |
| Fine-Grained Access Control | Chapter 14, "Implementing Application Context and Fine-Grained Access Control" |
| Fine-Grained Auditing | Chapter 8, "Database Auditing: Security Considerations", and<br>Chapter 11, "Configuring and Administering Auditing" |
| Oracle Label Security | *Oracle Label Security Administrator's Guide* |
| Proxy Authentication | Chapter 15, "Preserving User Identity in Multitiered Environments" |
| End-User Identity Propagation | Chapter 15, "Preserving User Identity in Multitiered Environments" |
| Secure Application Roles | Chapter 5, "Authorization: Privileges, Roles, Profiles, and Resource Limitations", and<br>Chapter 12, "Introducing Database Security for Application Developers" |

**See Also:** *Oracle Security Overview*

## Recommended Application Design Practices to Reduce Risk

To avoid or minimize potential problems, use the following recommended practices for database roles and privileges. Each practice is explained in the following sections in detail:

- Tip 1: Enable and Disable Roles Promptly

- Tip 2: Encapsulate Privileges in Stored Procedures
- Tip 3: Use Role Passwords Unknown to the User
- Tip 4: Use Proxy Authentication and a Secure Application Role
- Tip 5: Use Secure Application Role to Verify IP Address
- Tip 6: Use Application Context and Fine-Grained Access Control

### Tip 1: Enable and Disable Roles Promptly

Enable the proper role only when the application starts, and disable it as soon as the application terminates. To do this, you must take the following approach:

- Give each application distinct roles.
- For each application, establish one role that contains *all* privileges necessary to use the application successfully.
- If needed, establish several additional roles that contain only some of these privileges, to provide tighter or less restrictive security to different users or uses of the application
- Protect each database role by a password (or by operating system authentication) to prevent unauthorized use.

  One role should contain only non-destructive privileges associated with the application (SELECT privileges for specific tables or views associated with the application). This read-only role allows an application user to generate custom reports using ad hoc tools, such as SQL*Plus, but disallows modifying table data. A role designed for an ad hoc query tool may or may not be protected by a password (or by operating system authentication).
- Use the SET ROLE statement at application startup to enable one of the database roles associated with that application. For a role authorized by a password, the SET ROLE statement within the application must include that password, preferably encrypted by the application. If a role is authorized by the operating system, the system administrator must set up accounts in advance to provide application users with appropriate operating system privileges.
- Disable a previously enabled database role upon application termination.
- Grant application users database roles as needed.

**Note:** Database roles granted to users can nonetheless be enabled by users *outside the application*. Such use is not controlled by application-based security, but it can be controlled by virtual private database. In three-tier systems, using a secure application role prevents users from acquiring the role outside the application.

Additionally, you can use the PRODUCT_USER_PROFILE table to:

- Specify what roles to enable when a user starts SQL*Plus. This functionality is similar to that of a precompiler or Oracle Call Interface (OCI) application that issues a SET ROLE statement to enable specific roles upon application startup.
- Disable the use of the SET ROLE statement for SQL*Plus users, thereby restricting such users to only the privileges associated with the roles enabled when SQL*Plus started.
- Enable other ad hoc query and reporting tools to restrict the roles and commands that each user can use while running that product.

**See Also:**

- "Ways to Use Application Context with Fine-Grained Access Control" on page 13-16
- "Creating Secure Application Roles" on page 12-5
- The appropriate tool manual, such as the *SQL*Plus User's Guide and Reference*, which contains information about how to create and how to use the PRODUCT_USER_PROFILE table

### Tip 2: Encapsulate Privileges in Stored Procedures

Restrict users' ad hoc query tools from exercising application privileges, by encapsulating those privileges into stored procedures. Grant users execute privileges on those procedures, rather than issuing direct privilege grants to the users, so that the privileges cannot be used outside the appropriate procedure.

Users can then exercise privileges only in the context of well-formed business applications. For example, consider authorizing users to update a table only by executing a stored procedure, rather than by updating the table directly. By doing this, you avoid the problem of the user having the SELECT privilege and using it outside the application.

> **See Also:** "Example 3: Event Triggers, Application Context, Fine-Grained Access Control, and Encapsulation of Privileges" on page 14-13

### Tip 3: Use Role Passwords Unknown to the User

Grant privileges through roles that require a password unknown to the user.

For privileges that the user should exercise only within an application, enable the role by a password *known only by the creator of the role*. Use the application to issue a SET ROLE statement. Since the user does not have the role password, it must either be embedded in the application or retrievable from a database table by a stored procedure.

Hiding the password discourages users from trying to use the privileges without using the application, which does improve security, but it is not foolproof.

*Security by obscurity* is not a good security practice. It protects against lazy users who merely want to bypass the application, even though they could, with access to the application code, potentially find the password. It does not protect against users who want deliberately to misuse privileges without using the application code (malicious users). Since malicious users can decompile client code and recover embedded passwords, you should only use the *embedded password* method to protect against lazy users.

Retrieving the role password from a database table is a bit more secure. It requires that the user uncover what stored procedure to use, gain EXECUTE permission on that procedure, execute it, and retrieve the password. Only then could the user use the role outside of the application.

### Tip 4: Use Proxy Authentication and a Secure Application Role

In three-tier systems, enabling a role is possible only when the user accesses the database through a middle-tier application. This requires the use of proxy authentication and a secure application role.

Proxy authentication distinguishes between a middle tier creating a session on behalf of a user and the user connecting directly. Both the proxy user (the middle tier) and the *real* user information are captured in the user session.

A secure application role is implemented by a package, performing desired validation before allowing a user to assume the privileges in the role. When an application uses proxy authentication, the secure application role package validates that the user session was created by proxy. If the user is connecting to the database through an application, the role can be set, but if the user is connecting directly, it cannot.

Consider a situation in which you want to restrict use of an HR administration role to users accessing the database (by proxy) through the middle tier HRSERVER. You could create the following secure access role:

```
CREATE ROLE admin_role IDENTIFIED USING hr.admin;
```

Here, the `hr.admin` package performs the desired validation, permitting the role to be set only if it determines that the user is connected by proxy. The `hr.admin` package can use `SYS_CONTEXT ('userenv', 'proxy_userid')`, or `SYS_CONTEXT (userenv', 'proxy_user')`. Both return the ID and name of the proxy user (HRSERVER, in this case). If the user attempts to connect directly to the database, the `hr.admin` package will not allow the role to be set.

**See Also:**

- Re proxy authentication, Chapter 15, "Preserving User Identity in Multitiered Environments"
- Re secure application roles, "Creating Secure Application Roles" on page 12-5

Re application context:

- Application Context on page 6-6.
- Chapter 13, "Using Virtual Private Database to Implement Application Security Policies"
- Chapter 14, "Implementing Application Context and Fine-Grained Access Control"
- *Oracle Database SQL Reference* for information about `SYS_CONTEXT ('userenv', 'proxy_user')` and `SYS_CONTEXT ('userenv', proxy_userid')`

## Tip 5: Use Secure Application Role to Verify IP Address

The secure application role package can use additional information in the user session to restrict access, such as the user's original IP address.

You should never use IP addresses to make primary access control decisions, because IP addresses can be spoofed.

You can, however, use an IP address to increase access restrictions after using other criteria for the primary access control decision.

For example, you may want to ensure that a user session was created by proxy for a middle tier user connecting from a particular IP address. Of course, the middle tier must authenticate itself to the database before creating a lightweight session, and the database ensures that the middle tier has privilege to create a session on behalf of the user.

Your secure application role package could validate the IP address of the incoming connection. Before allowing `SET ROLE` to succeed, you can to ensure that the `HRSERVER` connection (or the lightweight user session) is coming from the appropriate IP address by using `SYS_CONTEXT (userenv',' 'ip_address').` Doing so provides an additional layer of security.

## Tip 6: Use Application Context and Fine-Grained Access Control

In this scenario, you combine server-enforced fine-grained access control and, through application context, session-based attributes.

**See Also:**

- Chapter 13, "Using Virtual Private Database to Implement Application Security Policies", and
- Chapter 14, "Implementing Application Context and Fine-Grained Access Control"

# Part II

# Security Features, Concepts, and Alternatives

Part II presents methods and Oracle Database features that address the security requirements, threats, and concepts presented in Part I.

This part contains the following chapters:

# 4 Authentication Methods

Authentication means verifying the identity of someone (a user, device, or other entity) who wants to use data, resources, or applications. Validating that identity establishes a trust relationship for further interactions. Authentication also enables accountability by making it possible to link access and actions to specific identities. After authentication, authorization processes can allow or limit the levels of access and action permitted to that entity, as described in Chapter 5, "Authorization: Privileges, Roles, Profiles, and Resource Limitations".

Oracle allows a single database instance to use any or all methods. Oracle requires special authentication procedures for database administrators, because they perform special database operations. Oracle also encrypts passwords during transmission to ensure the security of network authentication.

To validate the identity of database users and prevent unauthorized use of a database username, you can authenticate using any combination of the methods described in the following sections:

| Authentication Considerations About ... | Links to Topics |
|---|---|
| Operating Systems | Authentication by the Operating System |
| Networks and LDAP Directories | Authentication by the Network |
| Databases | Authentication by the Oracle Database |
| Multitier Systems | Multitier Authentication and Authorization |
| Secure Socket Layer Usage | Authentication of Database Administrators |
| Database Administrators | Authentication of Database Administrators |

**See Also:**

- Chapter 9, "Administering Authentication", discusses how to configure and administer these authentication methods.
- Chapter 15, "Preserving User Identity in Multitiered Environments", discusses proxy authentication.

## Authentication by the Operating System

Some operating systems permit Oracle to use information they maintain to authenticate users, with the following benefits:

- Once authenticated by the operating system, users can connect to Oracle more conveniently, without specifying a username or password. For example, an operating-system-authenticated user can invoke SQL*Plus and skip the username and password prompts by entering

  ```
  SQLPLUS /
  ```

- With control over user authentication centralized in the operating system, Oracle need not store or manage user passwords, though it still maintains usernames in the database.
- Audit trails in the database and operating system use the same usernames.

When an operating system is used to authenticate database users, managing distributed database environments and database links requires special care.

**See Also:**

- *Oracle Database Administrator's Guide* sections on (and index entries for) authentication, operating systems, distributed database concepts, and distributed data management.
- Your Oracle operating system-specific documentation for more information about authenticating by way of your operating system

## Authentication by the Network

Authentication capabilities at the network layer are handled by the SSL protocol or by third-party services, as described in the following subsections:

- Authentication by the Secure Socket Layer Protocol

- Authentication Using Third-Party Services
  - DCE Authentication
  - Kerberos Authentication
  - Public Key Infrastructure-Based Authentication
  - Authentication with RADIUS
  - Directory-based Services

## Authentication by the Secure Socket Layer Protocol

The Secure Socket Layer (SSL) protocol is an application layer protocol. It can be used for user authentication to a database, independent of global user management in Oracle Internet Directory. That is, users can use SSL to authenticate to the database even without a directory server in place.

## Authentication Using Third-Party Services

Authentication at the network layer makes use of third-party network authentication services. Prominent examples include the Distributed Computing Environment (DCE), Kerberos, public key infrastructure, the Remote Authentication Dial-In User Service (RADIUS), and directory-based services, as described in subsequent subsections.

If network authentication services are available to you, Oracle can accept authentication from the network service. To use a network authentication service with Oracle, you need Oracle Enterprise Edition with the Oracle Advanced Security option.

**See Also:**

- *Oracle Database Administrator's Guide* for more information about network authentication. If you use a network authentication service, some special considerations arise for network roles and database links: see the sections on (and index entries for) distributed database concepts and managing a distributed database.
- *Oracle Advanced Security Administrator's Guide* for information about Oracle Enterprise Edition with the Oracle Advanced Security option

### DCE Authentication

The Distributed Computing Environment (DCE) from the Open Group is a set of integrated network services that works across multiple systems to provide a distributed environment. The network services include remote procedure calls (RPCs), directory service, security service, threads, distributed file service, diskless support, and distributed time service.

DCE is the middleware between distributed applications and the operating system / network services and is based on a client / server model of computing. By using the services and tools that DCE provides, users can create, use, and maintain distributed applications that run across a heterogeneous environment.

### Kerberos Authentication

Kerberos is a trusted third-party authentication system that relies on shared secrets. It presumes that the third party is secure, and provides single sign-on capabilities, centralized password storage, database link authentication, and enhanced PC security. It does this through a Kerberos authentication server, or through Cybersafe Active Trust, a commercial Kerberos-based authentication server.

> **See Also:** *Oracle Advanced Security Administrator's Guide* for more information about DCE and Kerberos.

### Public Key Infrastructure-Based Authentication

Authentication systems based on public key infrastructure issue digital certificates to user clients, which use them to authenticate directly to servers in the enterprise without directly involving an authentication server. Oracle provides a public key infrastructure (PKI) for using public keys and certificates, consisting of the following components:

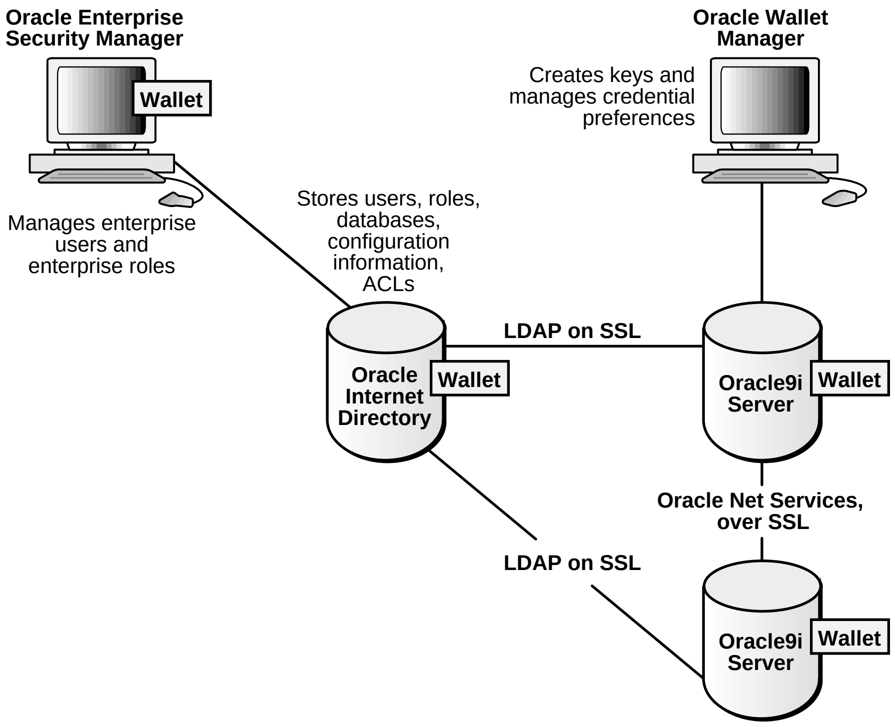
- Authentication and secure session key management using Secure Sockets Layer (SSL).
- Oracle Call Interface (OCI) and PL / SQL functions to sign user-specified data using a private key and certificate, and verify the signature on data using a trusted certificate.
- **Trusted certificates**, identifying third-party entities that are trusted as signers of user certificates when an identity is being validated as the entity it claims to be. When the user's certificate is being validated, the signer is one of the factors checked, using trust points or a trusted certificate chain of certificate authorities stored in the validating system. If there are several levels of trusted certificates

in that chain, a trusted certificate at a lower level is simply trusted without needing to have all its higher level certificates reverified.

- **Oracle wallets**, which are data structures that contain a user private key, a user certificate, and the user's set of trust points (trusted certificate authorities).
- **OracleAS Certificate Authority**, a component of the Oracle Identity Management infrastructure, which provides an integrated solution for provisioning X.509v3 certificates for use by individuals, applications, and servers, which require certificates for PKI-based operations such as authentication, SSL, S/MIME, and so on.
- **Oracle Wallet Manager**, a standalone Java application used to manage and edit the security credentials in Oracle wallets, providing the following operations:
  - Protects user keys
  - Manages X.509 Version 3 certificates on Oracle clients and servers
  - Generates a public-private key pair and creates a certificate request for submission to a certificate authority
  - Installs a certificate for the entity
  - Configures trusted certificates for the entity
  - Creates wallets
  - Opens a wallet to enable access to PKI-based services
- X.509 Version 3 certificates obtained from (and signed by) a trusted entity, a certificate authority. Such a certificate certifies, because the certificate authority is trusted, that the requesting entity's information is correct and that the public key on the certificate belongs to the identified entity. The certificate is loaded into an Oracle wallet to enable future authentication.

Oracle's public key infrastructure is illustrated in Figure 4–1.

***Figure 4–1 Oracle Public Key Infrastructure***


Oracle Enterprise Security Manager
Wallet
Manages enterprise users and enterprise roles
Oracle Wallet Manager
Creates keys and manages credential preferences
Stores users, roles, databases, configuration information, ACLs
LDAP on SSL
Oracle Internet Directory
Wallet
Oracle9i Server
Wallet
Oracle Net Services, over SSL
LDAP on SSL
Oracle9i Server
Wallet

---

**Note:** To use public key infrastructure-based authentication with Oracle, you need Oracle Enterprise Edition with the Oracle Advanced Security option.

---

## Authentication with RADIUS

Oracle supports remote authentication of users through the Remote Authentication Dial-In User Service (RADIUS), a standard lightweight protocol used for user authentication, authorization, and accounting.

---

**Note:** To use remote authentication of users through RADIUS with Oracle, you need Oracle Enterprise Edition with the Advanced Security option.

---

**See Also:** *Oracle Advanced Security Administrator's Guide* for information about Oracle Advanced Security

### Directory-based Services

Using a central directory can make authentication and its administration extremely efficient.

Oracle Internet Directory, which uses the Lightweight Directory Access Protocol (LDAP), enables information about users (called enterprise users) to be stored and managed centrally. Whereas database users must be created (with passwords) in each database they need to access, enterprise user information is accessible centrally in the Oracle Internet Directory. This directory is readily integrated with Active Directory and iPlanet.

- **Oracle Internet Directory**, built on the Oracle database and complying with the Lightweight Directory Access Protocol (LDAP v3). Oracle Internet Directory lets you manage the security attributes and privileges for users, including users authenticated by X.509 certificates. Oracle Internet Directory also enforces attribute-level access control. This feature enables read, write, or update privileges on specific attributes to be restricted to specific named users, such as an enterprise security administrator. Directory queries and responses can use SSL encryption for enhanced protection during authentication and other interactions.
- **Oracle Enterprise Security Manager**, which provides centralized privilege management to make administration easier and increase your level of security. Oracle Enterprise Security Manager lets you store and retrieve roles from Oracle Internet Directory. Oracle Enterprise Security Manager may also allow you to store roles in other directory servers that comply with LDAP v3 if those servers can also support the installation of the Oracle schema and related Access Control Lists.

**See Also:**

- *Oracle Internet Directory Administrator's Guide*
- *Oracle 2 Day DBA*
- *Oracle Enterprise Manager Concepts*
- *Oracle Enterprise Manager Advanced Configuration*

## Authentication by the Oracle Database

Oracle can authenticate users attempting to connect to a database, by using information stored in that database.

To set up Oracle to use database authentication, you create each user with an associated password that must be supplied when the user attempts to establish a connection. This process prevents unauthorized use of the database, since the connection will be denied if the user provides an incorrect password. Oracle stores a user's password in the data dictionary in an encrypted format to prevent unauthorized alteration, but a user can change his own password at any time.

To establish which authentication protocols are allowed by the client or database, a DBA can explicitly set the SQLNET.ALLOWED_LOGON_VERSION parameter in the server sqlnet.ora file. Then each connection attempt is tested, and if the client or server does not meet the minimum version specified by its partner, authentication fails with an ORA-28040 error. The parameter can take the values 10, 9, or 8, the default, representing database server versions. Oracle recommends the value 10.

Database authentication includes the following facilities:

- Password Encryption While Connecting. This protection is always in force, by default.
- Account Locking
- Password Lifetime and Expiration
- Password History
- Password Complexity Verification

### Password Encryption While Connecting

Passwords are always automatically and transparently encrypted during network (client/server and server/server) connections, using a modified DES (Data Encryption Standard) or 3DES algorithm, before sending them across the network.

**See Also:** For more information about encrypting passwords in network systems:

- Chapter 7, "Security Policies"
- *Oracle Database Administrator's Guide*
- *Oracle Advanced Security Administrator's Guide*

## Account Locking

Oracle can lock a user's account after a specified number of consecutive failed log-in attempts. You can configure the account to unlock automatically after a specified time interval or to require database administrator intervention to be unlocked.

Use the CREATE PROFILE statement to establish how many failed logins a user can attempt before the account locks, and how long it remains locked before it unlocks automatically.

The database administrator can also lock accounts manually, so that they cannot unlock automatically but must be unlocked explicitly by the database administrator.

**See Also:** "Profiles" on page 5-32

## Password Lifetime and Expiration

The database administrator can specify a lifetime for passwords, after which they expire and must be changed before account login is again permitted. A grace period can be established, during which each attempt to login to the database account receives a warning message to change the password. If it is not changed by the end of that period, the account is locked. No further logins to that account are allowed without assistance by the database administrator.

The database administrator can also set the password state to expired, causing the user's account status to change to expired. The user or the database administrator must then change the password before the user can log in to the database.

**See Also:** Password Management Policy in Chapter 7, "Security Policies"

## Password History

The password history option checks each newly specified password to ensure that a password is not reused for a specified amount of time or for a specified number of password changes. The database administrator can configure the rules for password reuse with CREATE PROFILE statements.

**See Also:**

- For the complete syntax of the CREATE PROFILE command, see the *Oracle Database SQL Reference* .
- For a more complete discussion of password history policies, see the Password History section in Chapter 7, "Security Policies"

## Password Complexity Verification

Complexity verification checks that each password is complex enough to provide reasonable protection against intruders who try to break into the system by guessing passwords.

The sample Oracle password complexity verification routine (PL/SQL script `UTLPWDMG.SQL`, which sets the default profile parameters) checks that each password meet the following requirements:

- Be a minimum of four characters in length
- Not equal the userid
- Include at least one alphabet character, one numeric character, and one punctuation mark
- Not match any word on an internal list of simple words like welcome, account, database, user, and so on
- Differ from the previous password by at least three characters

**See Also:** Chapter 7's discussion of Password Complexity Verification, on page 7-16.

## Multitier Authentication and Authorization

In a multitier environment, Oracle controls the security of middle-tier applications by limiting their privileges, preserving client identities through all tiers, and auditing actions taken on behalf of clients. In applications that use a heavy middle tier, such as a transaction processing monitor, the identity of the client connecting to the middle tier must be preserved. Yet one advantage of a middle tier is **connection pooling**, which allows multiple users to access a data server without each of them needing a separate connection. In such environments, you need to be able to set up and break down connections very quickly.

For these environments, Oracle database administrators can use the Oracle Call Interface to create **lightweight sessions**, allowing database password authentication for each user. This method preserves the identity of the real user through the middle tier without the overhead of a separate database connection for each user.

You can create lightweight sessions with or without passwords. However, if a middle tier is outside or on a firewall, security is better when each lightweight session has its own password. For an internal application server, lightweight sessions without passwords might be appropriate.

Issues of administration and security in multitier environments are discussed in the following sections:

- Clients, Application Servers, and Database Servers
- Security Issues for Middle-Tier Applications
- Identity Issues in a Multitier Environment
- Restricted Privileges in a Multitier Environment

## Clients, Application Servers, and Database Servers

In a multitier environment, an application server provides data for clients and serves as an interface from them to one or more database servers. The application server can validate the credentials of a client, such as a web browser, and the database server can audit operations performed by the application server. These auditable operations include actions performed by the application server on behalf of clients, such as requests that information be displayed on the client. A request to connect to the database server is an example of an application server operation not related to a specific client.

**Note:** While client-side authentication is possible, Oracle strongly recommends disallowing it by setting the remote_os_authentication parameter to FALSE.

Authentication in a multitier environment is based on trust regions. Client authentication is the province of the application server, which itself is authenticated by the database server. The following operations are performed:

- The client provides proof of authentication to the application server, typically using a password or an X.509 certificate.

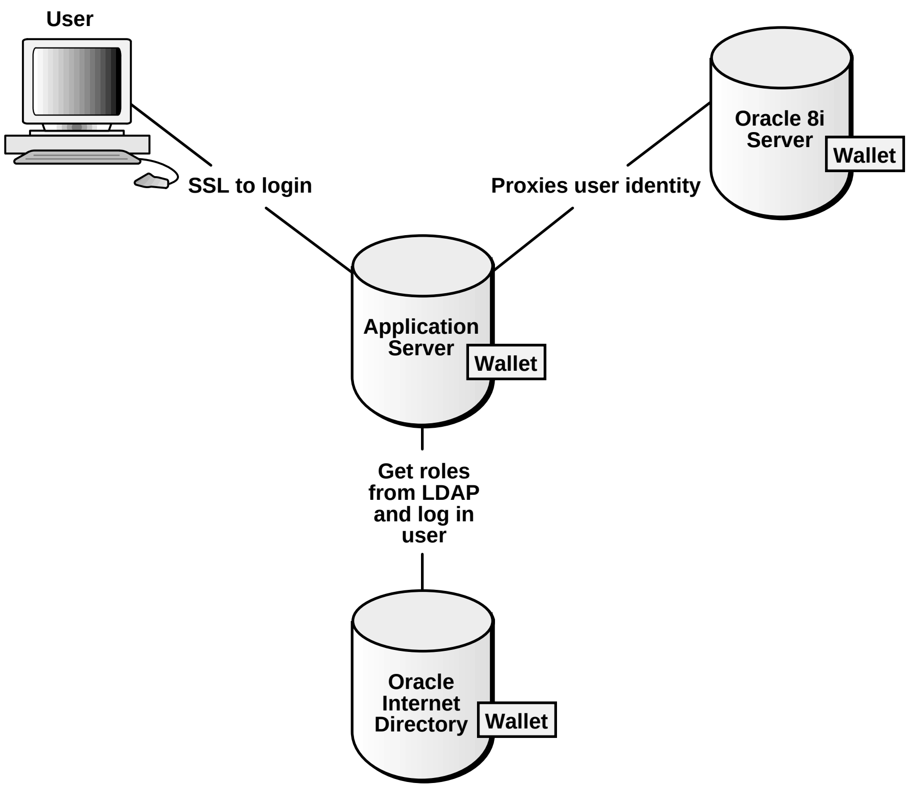
- The application server verifies the client authentication and then authenticates itself to the database server.
- The database server checks the application server authentication, verifies that the client exists, and verifies that the application server has the privilege to connect for this client.

Application servers can also enable roles for a client on whose behalf they connect. The application server can obtain these roles from a directory, which thus serves as an authorization repository. The application server can only request that these roles be enabled. The database verifies the following requirements:

- That the client has these roles by checking its internal role repository
- That the application server has the privilege to connect on behalf of the user, and thus to use these roles as the user could

Figure 4–2 shows an example of multitier authentication.

***Figure 4–2 Multitier Authentication***



## Security Issues for Middle-Tier Applications

Security for middle-tier applications must address the following key issues:

- Accountability: The database server must be able to distinguish between the actions of a client and the actions an application takes on behalf of a client. It must be possible to audit both kinds of actions.
- Differentiation: The database server must be able to distinguish between a client accessing the database directly and a web server acting either for itself or on behalf of a browser client.
- Least privilege: Users and middle tiers should be given the fewest privileges necessary to do their jobs, to minimize the danger of inadvertent or malicious unauthorized activities.

## Identity Issues in a Multitier Environment

Multitier authentication maintains the identity of the client through all tiers of the connection in order to maintain useful audit records. If the originating client's identity is lost, specific accountability of that client is lost. It becomes impossible to distinguish operations performed by the application server on behalf of the client from those done by the application server for itself.

## Restricted Privileges in a Multitier Environment

Privileges in a multitier environment are limited to those necessary to perform the requested operation.

### Client Privileges

Client privileges are as limited as possible in a multitier environment. Operations are performed on behalf of the client by the application server.

### Application Server Privileges

Application server privileges in a multitier environment are also limited, so that the application server cannot perform unwanted or unneeded operations while performing a client operation.

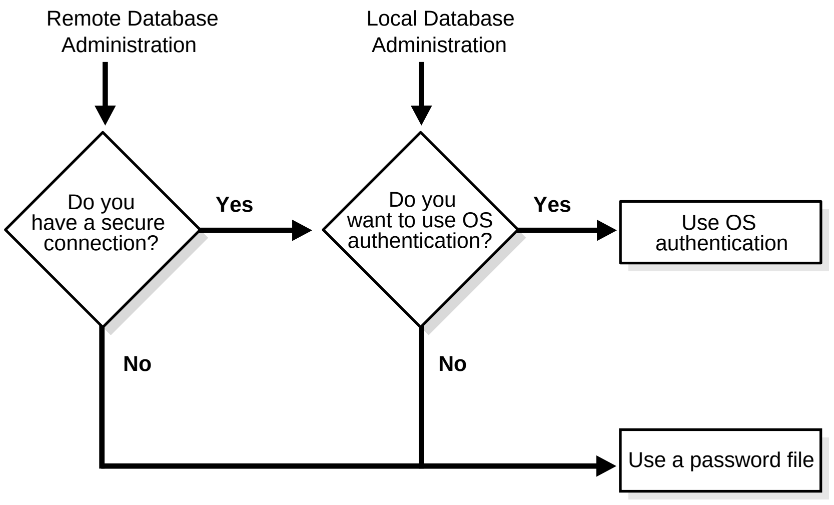
**See Also:** Chapter 9, "Administering Authentication", and the *Oracle Database Administrator's Guide*, for more information about multitier authentication

## Authentication of Database Administrators

Database administrators perform special operations (such as shutting down or starting up a database) that should not be performed by normal database users. Oracle provides for secure authentication of database administrator usernames, for which you can choose either operating system authentication or password files.

Figure 4–3 illustrates the choices you have for database administrator authentication schemes. Different choices apply to administering your database locally (on the machine where the database resides) and to administering many different database machines from a single remote client.

***Figure 4–3 Database Administrator Authentication Methods***


Remote Database Administration
Local Database Administration
Do you have a secure connection?
Yes
Do you want to use OS authentication?
Yes
Use OS authentication
No
No
Use a password file

Operating system authentication for a database administrator typically involves establishing a group on the operating system, assigning the DBA privileges to that group, and then adding to that group the names of persons who should have those privileges. (On UNIX systems, the special group is called the **dba** group.)

The database uses password files to keep track of those database usernames that have been granted the SYSDBA and SYSOPER privileges. These privileges enable the following operations and capabilities:

- SYSOPER lets database administrators perform STARTUP, SHUTDOWN, ALTER DATABASE OPEN/MOUNT, ALTER DATABASE BACKUP, ARCHIVE LOG, and RECOVER. SYSOPER also includes the RESTRICTED SESSION privilege.
- SYSDBA has all system privileges with ADMIN OPTION, including the SYSOPER system privilege, and permits CREATE DATABASE and time-based recovery.

---

**Note:** Connections requested AS SYSDBA or AS SYSOPER must use these phrases; without them, the connection fails. The Oracle parameter 07_DICTIONARY_ACCESSIBILITY is set to FALSE by default, to limit sensitive data dictionary access only to those authorized. The parameter also enforces the required AS SYSDBA or AS SYSOPER syntax. See also Administrator Security in Chapter 7, "Security Policies"

---

**See Also:**

- Your Oracle operating system-specific documentation for information about operating system authentication of database administrator*s*
- *Oracle Database Administrator's Guide*

# 5 Authorization: Privileges, Roles, Profiles, and Resource Limitations

Authorization includes primarily two processes:

- Permitting only certain users to access, process, or alter data
- Applying varying limitations on users' access or actions. The limitations placed on (or removed from) users can apply to objects, such as schemas, tables, or rows; or to resources, such as time (CPU, connect, or idle times).

This chapter introduces the basic concepts and mechanisms for placing or removing such limitations on users, individually or in groups, in the following sections:

| Topic Category | Links to Topics |
|---|---|
| How Privileges Are Acquired and Used | Introduction to Privileges, including system, schema, object, table, procedure, and other privileges |
| How Roles Are Acquired, Used, and Restricted | Introduction to Roles |
| How and Why Resource Limits Are Applied to Users | User Resource Limits |
| How Profiles Are Determined and Used | Profiles |

**See Also:** Chapter 10, "Administering User Privileges, Roles, and Profiles", discusses how to configure and administer privileges, roles, and profiles for users, including DBAs and application programmers.

## Introduction to Privileges

A **privilege** is a right to execute a particular type of SQL statement or to access another user's object. Some examples of privileges include the right to:

- Connect to the database (create a session)
- Create a table
- Select rows from another user's table
- Execute another user's stored procedure

You grant privileges to users so these users can accomplish tasks required for their jobs. You should grant a privilege only to a user who absolutely requires that privilege to accomplish necessary work. Excessive granting of unnecessary privileges can compromise security. A user can receive a privilege in two different ways:

- You can grant privileges to users explicitly. For example, you can explicitly grant to user `SCOTT` the privilege to insert records into the `employees` table.
- You can also grant privileges to a role (a named group of privileges), and then grant the role to one or more users. For example, you can grant the privileges to select, insert, update, and delete records from the `employees` table to the role named `clerk`, which in turn you can grant to users `scott` and `brian`.

Because roles allow for easier and better management of privileges, you should normally grant privileges to roles and not to specific users.

**See Also:**

- *Chapter 10, "Administering User Privileges, Roles, and Profiles"*
- *Oracle Database Administrator's Guide* for discussions of managing and using system and schema object privileges.
- *Oracle Database SQL Reference* for the complete list of system privileges and their descriptions.

There are six major categories of privileges, some with significant subcategories:

- System Privileges
- Schema Object Privileges
- Table Privileges
- View Privileges

- Procedure Privileges
- Type Privileges

## System Privileges

A system privilege is the right to perform a particular action, or to perform an action on any schema objects of a particular type. For example, the privileges to create tablespaces and to delete the rows of any table in a database are system privileges. There are over 100 distinct system privileges to manage, as described in the following subsections:

- Granting and Revoking System Privileges
- Who Can Grant or Revoke System Privileges?

### Granting and Revoking System Privileges

You can grant or revoke system privileges to users and roles. If you grant system privileges to roles, then you can use the roles to manage system privileges. For example, roles permit privileges to be made selectively available.

**Note:** In general, you grant system privileges only to administrative personnel and application developers. End users normally do not require and should not have the associated capabilities.

Use either of the following to grant or revoke system privileges to users and roles:

- The Oracle Enterprise Manager 10g Database Control
- The SQL statements `GRANT` and `REVOKE`

**See Also:**

- For more information about the Database Control, see *Oracle 2 Day DBA*.
- For information about modifying users with the Database Control, see the topic "Creating, Editing, and Deleting Users" in the Enterprise Manager online help.

### Who Can Grant or Revoke System Privileges?

Only two types of users can grant system privileges to other users or revoke such privileges from them:

- Users who have been granted a specific system privilege with the `ADMIN OPTION`
- Users with the system privilege `GRANT ANY PRIVILEGE`

## Schema Object Privileges

A **schema object privilege** is a privilege or right to perform a particular action on a specific schema object.

Different object privileges are available for different types of schema objects. The privilege to delete rows from the `departments` table is an example of an object privilege.

Some schema objects, such as clusters, indexes, triggers, and database links, do not have associated object privileges. Their use is controlled with system privileges. For example, to alter a cluster, a user must own the cluster or have the `ALTER ANY CLUSTER` system privilege.

The following subsections discuss granting and revoking such privileges:

- Granting and Revoking Schema Object Privileges
- Who Can Grant Schema Object Privileges?
- Using Privileges with Synonyms

Object privileges that apply to specific schema objects are discussed in the following sections:

- Table Privileges
- View Privileges
- Sequences (see "Managing Sequences" in *Oracle Database Administrator's Guide*
- Procedure Privileges
- Functions and Packages ("Managing Object Dependencies" in *Oracle Database Administrator's Guide*)
- Type Privileges

### Granting and Revoking Schema Object Privileges

Schema object privileges can be granted to and revoked from users and roles. If you grant object privileges to roles, you can make the privileges selectively available. Object privileges for users and roles can be granted or revoked using the following:

- The SQL statements `GRANT` and `REVOKE`, respectively
- The Oracle Enterprise Manager 10g Database Control

> **See Also:**
>
> - For more information about the Database Control, see *Oracle 2 Day DBA*.
> - For information about modifying privileges with the Database Control, see the Enterprise Manager online help.

### Who Can Grant Schema Object Privileges?

A user automatically has all object privileges for schema objects contained in his or her schema. A user can grant any object privilege on any schema object he or she owns to any other user or role. A user with the `GRANT ANY OBJECT PRIVILEGE` can grant or revoke any specified object privilege to another user with or without the `GRANT OPTION` of the `GRANT` statement. Otherwise, the grantee can use the privilege, but cannot grant it to other users.

For example, assume user `SCOTT` has a table named `t2`:

```
SQL>GRANT grant any object privilege TO U1;
SQL> connect u1/u1
Connected.
SQL> GRANT select on scott.t2 \TO U2;
SQL> SELECT GRANTEE, OWNER, GRANTOR, PRIVILEGE, GRANTABLE FROM DBA_TAB_PRIVS
 WHERE TABLE_NAME = 'employees';

GRANTEE                        OWNER
------------------------------ ------------------------------
GRANTOR                        PRIVILEGE                                GRA
------------------------------ ---------------------------------------- ---
U2                             SCOTT
SCOTT                          SELECT                                   NO
```

> **See Also:** *Oracle Database SQL Reference*

### Using Privileges with Synonyms

A schema object and its synonym are equivalent with respect to privileges. That is, the object privileges granted for a table, view, sequence, procedure, function, or package apply whether referencing the base object by name or using a synonym.

For example, assume there is a table `jward.emp` with a synonym named `jward.employee`, and the user `jward` issues the following statement:

```
GRANT SELECT ON emp TO swilliams;
```

The user `swilliams` can query `jward.emp` by referencing the table by name or using the synonym `jward.employee`:

```
SELECT * FROM jward.emp;
SELECT * FROM jward.employee;
```

If you grant object privileges on a table, view, sequence, procedure, function, or package to a **synonym** for the object, the effect is the same as if no synonym were used. For example, if `jward` wanted to grant the `SELECT` privilege for the `emp` table to `swilliams`, `jward` could issue either of the following statements:

```
GRANT SELECT ON emp TO swilliams;
GRANT SELECT ON employee TO swilliams;
```

If a synonym is dropped, all grants for the underlying schema object remain in effect, even if the privileges were granted by specifying the dropped synonym.

## Table Privileges

Schema object privileges for tables enable table security at the DML or DDL level of operation, as discussed in the following subsections:

- Data Manipulation Language (DML) Operations
- Data Definition Language (DDL) Operations

### Data Manipulation Language (DML) Operations

You can grant privileges to use the `DELETE`, `INSERT`, `SELECT`, and `UPDATE` DML operations on a table or view. Grant these privileges only to users and roles that need to query or manipulate a table's data.

You can restrict `INSERT` and `UPDATE` privileges for a table to specific columns of the table. With selective `INSERT`, a privileged user can insert a row with values for the selected columns. All other columns receive `NULL` or the column's default value. With selective `UPDATE`, a user can update only specific column values of a row.

Selective INSERT and UPDATE privileges are used to restrict a user's access to sensitive data.

For example, if you do not want data entry users to alter the salary column of the employees table, selective INSERT or UPDATE privileges can be granted that exclude the salary column. Alternatively, a view that excludes the salary column could satisfy this need for additional security.

> **See Also:** *Oracle Database SQL Reference* for more information about these DML operations

### Data Definition Language (DDL) Operations

The ALTER, INDEX, and REFERENCES privileges allow DDL operations to be performed on a table. Because these privileges allow other users to alter or create dependencies on a table, you should grant privileges conservatively. A user attempting to perform a DDL operation on a table may need additional system or object privileges. For example, to create a trigger on a table, the user requires both the ALTER TABLE object privilege for the table and the CREATE TRIGGER system privilege.

As with the INSERT and UPDATE privileges, the REFERENCES privilege can be granted on specific columns of a table. The REFERENCES privilege enables the grantee to use the table on which the grant is made as a parent key to any foreign keys that the grantee wishes to create in his or her own tables. This action is controlled with a special privilege because the presence of foreign keys restricts the data manipulation and table alterations that can be done to the parent key.
A column-specific REFERENCES privilege restricts the grantee to using the named columns (which, of course, must include at least one primary or unique key of the parent table).

> **See Also:** "Data Integrity" in *Oracle Database Concepts* for more information about primary keys, unique keys, and integrity constraints

## View Privileges

A view is a presentation of data selected from one or more tables (possibly including other views). A view shows the structure of the underlying tables as well as the selected data, and can be thought of as the result of a stored query. The view contains no actual data but rather derives what it shows from the tables and views on which it is based. A view can be queried, and the data it represents can be changed. Data in a view can be updated or deleted, and new data inserted. These operations directly alter the tables on which the view is based and are subject to the integrity constraints and triggers of the base tables.

Data Manipulation Language (DML) object privileges for tables can be applied similarly to views. Schema object privileges for a view allow various DML operations, which as noted affect the base tables from which the view is derived. These privileges are discussed in the following subsections:

- Privileges Required to Create Views
- Increasing Table Security with Views

## Privileges Required to Create Views

To create a view, you must meet the following requirements:

- You must have been granted one of the following system privileges, either explicitly or through a role:
  - The `CREATE VIEW` system privilege (to create a view in your schema)
  - The `CREATE ANY VIEW` system privilege (to create a view in another user's schema)
- You must have been explicitly granted one of the following privileges:
  - The `SELECT`, `INSERT`, `UPDATE`, or `DELETE` object privileges on all base objects underlying the view
  - The `SELECT ANY TABLE`, `INSERT ANY TABLE`, `UPDATE ANY TABLE`, or `DELETE ANY TABLE` system privileges
- Additionally, in order to grant other users access to your view, you must have received object privileges to the base objects with the `GRANT OPTION` clause or appropriate system privileges with the `ADMIN OPTION` clause. If you have not, grantees cannot access your view.

**See Also:** *Oracle Database SQL Reference*

## Increasing Table Security with Views

To use a view, you require appropriate privileges only for the view itself. You do not require privileges on base objects underlying the view.

Views add two more levels of security for tables, column-level security and value-based security:

- A view can provide access to selected columns of base tables. For example, you can define a view on the `employees` table to show only the `employee_id`, `last_name`, and `manager_id` columns:

  ```
  CREATE VIEW employees_manager AS
      SELECT last_name, employee_id, manager_id FROM employees;
  ```

- A view can provide value-based security for the information in a table. A `WHERE` clause in the definition of a view displays only selected rows of base tables. Consider the following two examples:

  ```
  CREATE VIEW lowsal AS
      SELECT * FROM employees
      WHERE salary < 10000;
  ```

  The `LOWSAL` view allows access to all rows of the `employees` table that have a salary value less than 10000. Notice that all columns of the `employees` table are accessible in the `LOWSAL` view.

  ```
  CREATE VIEW own_salary AS
      SELECT last_name, salary
      FROM employees
      WHERE last_name = USER;
  ```

  In the `own_salary` view, only the rows with an `last_name` that matches the current user of the view are accessible. The `own_salary` view uses the `user` pseudocolumn, whose values always refer to the current user. This view combines both column-level security and value-based security.

## Procedure Privileges

`EXECUTE` is the only **schema object privilege** for procedures, including standalone procedures and functions as well as packages. Grant this privilege only to users who need to execute a procedure or to compile another procedure that calls a desired procedure. To create and manage secure and effective use of procedure privileges, you need to understand the following subsections:

- Procedure Execution and Security Domains
- System Privileges Needed to Create or Alter a Procedure
- Packages and Package Objects

### Procedure Execution and Security Domains

A user with the EXECUTE object privilege for a specific procedure can execute the procedure or compile a program unit that references the procedure. No runtime privilege check is made when the procedure is called. A user with the EXECUTE ANY PROCEDURE system privilege can execute any procedure in the database.

Privileges to execute procedures can be granted to a user through roles.

A procedure's owner, called the "definer," must have all the necessary object privileges for referenced objects. If the owner grants to another user the right to use that procedure, the owner's object privileges for the objects referenced by the procedure apply to that user's exercise of the procedure. These are termed "definer’s rights."

The user of a procedure who is not its owner is termed the "invoker." Additional privileges on referenced objects are required for invoker’s rights procedures, but not for definer’s rights procedures.

**See Also:** "PL/SQL Blocks and Roles" on page 5-24

**Definer’s Rights** A user of a definer’s rights procedure requires only the privilege to execute the procedure and no privileges on the underlying objects that the procedure accesses, because a definer’s rights procedure operates under the security domain of the user who owns the procedure, regardless of who is executing it. The procedure's owner must have all the necessary object privileges for referenced objects. Fewer privileges have to be granted to users of a definer’s rights procedure, resulting in tighter control of database access.

You can use definer’s rights procedures to control access to private database objects and add a level of database security. By writing a definer’s rights procedure and granting only EXECUTE privilege to a user, the user can be forced to access the referenced objects only through the procedure.

At runtime, the privileges of the owner of a definer’s rights stored procedure are always checked before the procedure is executed. If a necessary privilege on a referenced object has been revoked from the owner of a definer’s rights procedure, then the procedure cannot be executed by the owner or any other user.

---

**Note:** Trigger execution follows the same patterns as definer’s rights procedures. The user executes a SQL statement, which that user is privileged to execute. As a result of the SQL statement, a trigger is fired. The statements within the triggered action temporarily execute under the security domain of the user that owns the trigger.

---

**See Also:** "Triggers" in *Oracle Database Concepts*

**Invoker’s Rights** An invoker’s rights procedure executes with all of the invoker's privileges. Roles are enabled unless the invoker’s rights procedure was called directly or indirectly by a definer’s rights procedure. A user of an invoker’s rights procedure needs privileges (either directly or through a role) on objects that the procedure accesses through external references that are resolved in the invoker's schema.

The invoker needs privileges at runtime to access program references embedded in DML statements or dynamic SQL statements, because they are effectively recompiled at runtime.

For all other external references, such as direct PL/SQL function calls, the owner's privileges are checked at compile time, and no runtime check is made. Therefore, the user of an invoker’s rights procedure needs no privileges on external references outside DML or dynamic SQL statements. Alternatively, the developer of an invoker’s rights procedure only needs to grant privileges on the procedure itself, not on all objects directly referenced by the invoker’s rights procedure.

Many packages provided by Oracle, such as most of the `DBMS_*` packages, run with invoker’s rights—they do not run as the owner (SYS) but rather as the current user. However, some exceptions exist such as the `DBMS_RLS` package.

You can create a software bundle that consists of multiple program units, some with definer’s rights and others with invoker’s rights, and restrict the program entry points (**controlled step-in**). A user who has the privilege to execute an entry-point procedure can also execute internal program units indirectly, but cannot directly call the internal programs.

**See Also:**

- "Fine-Grained Access Control" on page 22-24
- *PL/SQL Packages and Types Reference* for detailed documentation of the Oracle supplied packages

### System Privileges Needed to Create or Alter a Procedure

To create a procedure, a user must have the CREATE PROCEDURE or CREATE ANY PROCEDURE **system privilege**. To alter a procedure, that is, to manually recompile a procedure, a user must own the procedure or have the ALTER ANY PROCEDURE system privilege.

The user who owns the procedure also must have privileges for schema objects referenced in the procedure body. To create a procedure, you must have been explicitly granted the necessary privileges (system or object) on all objects referenced by the procedure. You cannot have obtained the required privileges through roles. This includes the EXECUTE privilege for any procedures that are called inside the procedure being created.

Triggers also require that privileges to referenced objects be granted explicitly to the trigger owner. Anonymous PL/SQL blocks can use any privilege, whether the privilege is granted explicitly or through a role.

### Packages and Package Objects

A user with the EXECUTE object privilege for a package can execute any public procedure or function in the package and access or modify the value of any public package variable. Specific EXECUTE privileges cannot be granted for a package's constructs. Therefore, you may find it useful to consider two alternatives for establishing security when developing procedures, functions, and packages for a database application. These alternatives are described in the following examples.

**Packages and Package Objects Example 1** This example shows four procedures created in the bodies of two packages.

```
CREATE PACKAGE BODY hire_fire AS
  PROCEDURE hire(...) IS
    BEGIN
      INSERT INTO employees . . .
    END hire;
  PROCEDURE fire(...) IS
    BEGIN
      DELETE FROM employees . . .
    END fire;
END hire_fire;

CREATE PACKAGE BODY raise_bonus AS
  PROCEDURE give_raise(...) IS
    BEGIN
      UPDATE employees SET salary = . . .
    END give_raise;
  PROCEDURE give_bonus(...) IS
    BEGIN
      UPDATE employees SET bonus = . . .
    END give_bonus;
END raise_bonus;
```

Access to execute the procedures is given by granting the EXECUTE privilege for the package, using the following statements:

```
GRANT EXECUTE ON hire_fire TO big_bosses;
GRANT EXECUTE ON raise_bonus TO little_bosses;
```

Granting EXECUTE privilege granted for a package provides uniform access to all package objects.

**Packages and Package Objects Example 2** This example shows four procedure definitions within the body of a single package. Two additional standalone procedures and a package are created specifically to provide access to the procedures defined in the main package.

```
CREATE PACKAGE BODY employee_changes AS
  PROCEDURE change_salary(...) IS BEGIN ... END;
  PROCEDURE change_bonus(...) IS BEGIN ... END;
  PROCEDURE insert_employee(...) IS BEGIN ... END;
  PROCEDURE delete_employee(...) IS BEGIN ... END;
END employee_changes;

CREATE PROCEDURE hire
  BEGIN
    employee_changes.insert_employee(...)
  END hire;

CREATE PROCEDURE fire
  BEGIN
    employee_changes.delete_employee(...)
  END fire;
```

```
PACKAGE raise_bonus IS
  PROCEDURE give_raise(...) AS
    BEGIN
      employee_changes.change_salary(...)
    END give_raise;

  PROCEDURE give_bonus(...)
    BEGIN
      employee_changes.change_bonus(...)
    END give_bonus;
```

Using this method, the procedures that actually do the work (the procedures in the `employee_changes` package) are defined in a single package and can share declared global variables, cursors, on so on. By declaring top-level procedures `hire` and `fire`, and an additional package `raise_bonus`, you can grant selective `EXECUTE` privileges on procedures in the main package:

```
GRANT EXECUTE ON hire, fire TO big_bosses;
GRANT EXECUTE ON raise_bonus TO little_bosses;
```

## Type Privileges

The following subsections describe the use of privileges for types, methods, and objects:

- System Privileges for Named Types
- Object Privileges
- Method Execution Model
- Privileges Required to Create Types and Tables Using Types
- Example of Privileges for Creating Types and Tables Using Types
- Privileges on Type Access and Object Access
- Type Dependencies

### System Privileges for Named Types

Oracle defines system privileges shown in Table 5–1 for named types (object types, VARRAYs, and nested tables):

***Table 5–1 System Privileges for Named Types***

| Privilege | Allows you to... |
|---|---|
| `CREATE TYPE` | Create named types in your own schemas. |
| `CREATE ANY TYPE` | Create a named type in any schema. |
| `ALTER ANY TYPE` | Alter a named type in any schema. |
| `DROP ANY TYPE` | Drop a named type in any schema. |
| `EXECUTE ANY TYPE` | Use and reference a named type in any schema. |

The CONNECT and RESOURCE roles include the CREATE TYPE system privilege. The DBA role includes all of these privileges.

### Object Privileges

The only object privilege that applies to named types is EXECUTE. If the EXECUTE privilege exists on a named type, a user can use the named type to:

- Define a table
- Define a column in a relational table
- Declare a variable or parameter of the named type

The EXECUTE privilege permits a user to invoke the type's methods, including the type constructor. This is similar to EXECUTE privilege on a stored PL/SQL procedure.

### Method Execution Model

Method execution is the same as any other stored PL/SQL procedure.

**See Also:** "Procedure Privileges" on page 5-9

### Privileges Required to Create Types and Tables Using Types

To create a type, you must meet the following requirements:

- You must have the CREATE TYPE system privilege to create a type in your schema or the CREATE ANY TYPE system privilege to create a type in another user's schema. These privileges can be acquired explicitly or through a role.
- The owner of the type must have been explicitly granted the EXECUTE object privileges to access all other types referenced within the definition of the type,

or have been granted the EXECUTE ANY TYPE system privilege. The owner cannot have obtained the required privileges through roles.

- If the type owner intends to grant access to the type to other users, the owner must have received the EXECUTE privileges to the referenced types with the GRANT OPTION or the EXECUTE ANY TYPE system privilege with the ADMIN OPTION. If not, the type owner has insufficient privileges to grant access on the type to other users.

To create a table using types, you must meet the requirements for creating a table and these additional requirements:

- The owner of the table must have been explicitly granted the EXECUTE object privileges to access all types referenced by the table, or have been granted the EXECUTE ANY TYPE system privilege. The owner cannot have obtained the required privileges through roles.
- If the table owner intends to grant access to the table to other users, the owner must have received the EXECUTE privileges to the referenced types with the GRANT OPTION or the EXECUTE ANY TYPE system privilege with the ADMIN OPTION. If not, the table owner has insufficient privileges to grant access on the type to other users.

**See Also:** "Table Privileges" on page 5-6 for the requirements for creating a table

### Example of Privileges for Creating Types and Tables Using Types

Assume that three users exist with the CONNECT and RESOURCE roles:

- user1
- user2
- user3

User1 performs the following DDL in his schema:

```
CREATE TYPE type1 AS OBJECT (
  attr1 NUMBER);

CREATE TYPE type2 AS OBJECT (
  attr2 NUMBER);

GRANT EXECUTE ON type1 TO user2;
GRANT EXECUTE ON type2 TO user2 WITH GRANT OPTION;
```

User2 performs the following DDL in his schema:

```
CREATE TABLE tab1 OF user1.type1;
CREATE TYPE type3 AS OBJECT (
  attr3 user1.type2);
CREATE TABLE tab2 (
  col1 user1.type2);
```

The following statements succeed because user2 has EXECUTE privilege on user1's TYPE2 with the GRANT OPTION:

```
GRANT EXECUTE ON type3 TO user3;
GRANT SELECT on tab2 TO user3;
```

However, the following grant fails because user2 does not have EXECUTE privilege on user1's TYPE1 with the GRANT OPTION:

```
GRANT SELECT ON tab1 TO user3;
```

User3 can successfully perform the following statements:

```
CREATE TYPE type4 AS OBJECT (
  attr4 user2.type3);
CREATE TABLE tab3 OF type4;
```

## Privileges on Type Access and Object Access

Existing column-level and table-level privileges for DML statements apply to both column objects and row objects. Oracle defines the privileges shown in Table 5–2 for object tables:

***Table 5–2 Privileges for Object Tables***

| Privilege | Allows you to... |
|---|---|
| SELECT | Access an object and its attributes from the table |
| UPDATE | Modify the attributes of the objects that make up the table's rows |
| INSERT | Create new objects in the table |
| DELETE | Delete rows |

Similar table privileges and column privileges apply to column objects. Retrieving instances does not in itself reveal type information. However, clients must access named type information in order to interpret the type instance images. When a client requests such type information, Oracle checks for EXECUTE privilege on the type.

Consider the following schema:

```
CREATE TYPE emp_type (
    eno NUMBER, ename CHAR(31), eaddr addr_t);
CREATE TABLE emp OF emp_t;
```

and the following two queries:

```
SELECT VALUE(emp) FROM emp;
SELECT eno, ename FROM emp;
```

For either query, Oracle checks the user's SELECT privilege for the emp table. For the first query, the user needs to obtain the emp_type type information to interpret the data. When the query accesses the emp_type type, Oracle checks the user's EXECUTE privilege.

Execution of the second query, however, does not involve named types, so Oracle does not check type privileges.

Additionally, using the schema from the previous section, user3 can perform the following queries:

```
SELECT tab1.col1.attr2 FROM user2.tab1 tab1;
SELECT attr4.attr3.attr2 FROM tab3;
```

Note that in both SELECT statements, user3 does not have explicit privileges on the underlying types, but the statement succeeds because the type and table owners have the necessary privileges with the GRANT OPTION.

Oracle checks privileges on the following events, and returns an error if the client does not have the privilege for the action:

- Pinning an object in the object cache using its REF value causes Oracle to check SELECT privilege on the containing object table.
- Modifying an existing object or flushing an object from the object cache causes Oracle to check UPDATE privilege on the destination object table.
- Flushing a new object causes Oracle to check INSERT privilege on the destination object table.
- Deleting an object causes Oracle to check DELETE privilege on the destination table.
- Pinning an object of named type causes Oracle to check EXECUTE privilege on the object.

Modifying an object's attributes in a client 3GL application causes Oracle to update the entire object. Hence, the user needs `UPDATE` privilege on the object table. `UPDATE` privilege on only certain columns of the object table is not sufficient, even if the application only modifies attributes corresponding to those columns. Therefore, Oracle does not support column level privileges for object tables.

### Type Dependencies

As with stored objects such as procedures and tables, types being referenced by other objects are called dependencies. There are some special issues for types depended upon by tables. Because a table contains data that relies on the type definition for access, any change to the type causes all stored data to become inaccessible. Changes that can cause this effect are when necessary privileges required by the type are revoked or the type or dependent types are dropped. If either of these actions occur, then the table becomes invalid and cannot be accessed.

A table that is invalid because of missing privileges can automatically become valid and accessible if the required privileges are granted again. A table that is invalid because a dependent type has been dropped can never be accessed again, and the only permissible action is to drop the table.

Because of the severe effects which revoking a privilege on a type or dropping a type can cause, the SQL statements `REVOKE` and `DROP TYPE` by default implement a restrict semantics. This means that if the named type in either statement has table or type dependents, then an error is received and the statement aborts. However, if the `FORCE` clause for either statement is used, the statement always succeeds, and if there are depended-upon tables, they are invalidated.

**See Also:** *Oracle Database Reference* for details about using the `REVOKE`, `DROP TYPE`, and `FORCE` clauses

## Introduction to Roles

Managing and controlling privileges is made easier by using **roles**, which are named groups of related privileges that you grant, as a group, to users or other roles. Within a database, each role name must be unique, different from all usernames and all other role names. Unlike schema objects, roles are not contained in any schema. Therefore, a user who creates a role can be dropped with no effect on the role.

Roles are designed to ease the administration of end-user system and schema object privileges, and are often maintained in Oracle Internet Directory. However, roles are not meant to be used by application developers, because the privileges to access schema objects within stored programmatic constructs need to be granted directly.

The effective management of roles is discussed in the following subsections:

*Table 5–3 Topics and Sections in This Section*

| Authentication Considerations in These Topical Areas | Links to Relevant Subsection |
|---|---|
| Why Roles Are Advantageous | Properties of Roles |
| How Roles are Typically Used | Common Uses for Roles |
| How Users Get Roles (or Role Restrictions | Granting and Revoking Roles |
| How Roles Affect The Scope of a User's Privileges | Security Domains of Roles and Users |
| How Roles Work in PL/SQL Blocks | PL/SQL Blocks and Roles |
| How Roles Aid or Restrict DDL Usage | Data Definition Language Statements and Roles |
| What Roles are Predefined in Oracle | Predefined Roles |
| How Can Operating Systems Aid Roles | The Operating System and Roles |
| How Roles Work in a Remote Session | Roles in a Distributed Environment |
| How Secure Application Roles Are Created and Used | Secure Application Roles |

## Properties of Roles

These following properties of roles enable easier privilege management within a database:

| Property | Description |
|---|---|
| Reduced privilege administration | Rather than granting the same set of privileges explicitly to several users, you can grant the privileges for a group of related users to a role, and then only the role needs to be granted to each member of the group. |
| Dynamic privilege management | If the privileges of a group must change, only the privileges of the role need to be modified. The security domains of all users granted the group's role automatically reflect the changes made to the role. |

| Property | Description |
|---|---|
| Selective availability of privileges | You can selectively enable or disable the roles granted to a user. This allows specific control of a user's privileges in any given situation. |
| Application awareness | The data dictionary records which roles exist, so you can design applications to query the dictionary and automatically enable (or disable) selective roles when a user attempts to execute the application by way of a given username. |
| Application-specific security | You can protect role use with a password. Applications can be created specifically to enable a role when supplied the correct password. Users cannot enable the role if they do not know the password. |

Database administrators often create roles for a database application. The DBA grants a secure application role all privileges necessary to run the application. The DBA then grants the secure application role to other roles or users. An application can have several different roles, each granted a different set of privileges that allow for more or less data access while using the application.

The DBA can create a role with a password to prevent unauthorized use of the privileges granted to the role. Typically, an application is designed so that when it starts, it enables the proper role. As a result, an application user does not need to know the password for an application's role.

**See Also:**

- "Data Definition Language Statements and Roles" on page 5-24 for information about restrictions for procedures
- *Oracle Database Application Developer's Guide - Fundamentals* for instructions for enabling roles from an application

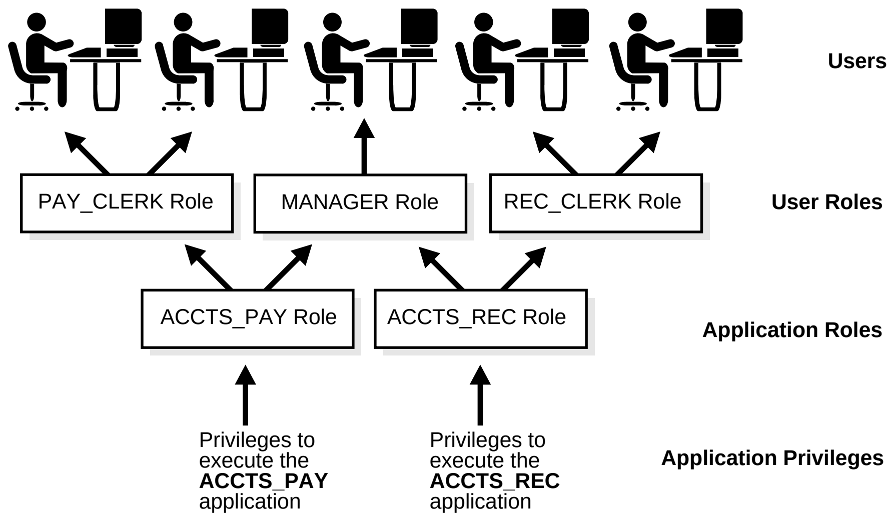
## Common Uses for Roles

In general, you create a role to serve one of two purposes:

- To manage the privileges for a database application (Application Roles)
- To manage the privileges for a user group (User Roles)

Figure 5–1 and the sections that follow describe the two uses of roles.

***Figure 5–1 Common Uses for Roles***



### Application Roles

You grant an application role all privileges necessary to run a given database application. Then, you grant the secure application role to other roles or to specific users. An application can have several different roles, with each role assigned a different set of privileges that allow for more or less data access while using the application.

### User Roles

You create a user role for a group of database users with common privilege requirements. You manage user privileges by granting secure application roles and privileges to the user role and then granting the user role to appropriate users.

## Granting and Revoking Roles

System or schema object privileges can be granted to a role, and any role can be granted to any database user or to another role (but not to itself). However, a role cannot be granted circularly, that is, a role X cannot be granted to role Y if role Y has previously been granted to role X.

To provide selective availability of privileges, Oracle allows database applications and users to enable and disable roles. Each role granted to a user is, at any given time, either enabled or disabled. A user's security domain includes the privileges of all roles currently enabled for the user and excludes the privileges of any roles currently disabled for the user.

A role granted to a role is called an indirectly granted role. It can be explicitly enabled or disabled for a user. However, whenever you enable a role that contains other roles, you implicitly enable all indirectly granted roles of the directly granted role.

You grant roles to (or revoke roles from) users or other roles by using either of the following methods:

- The Oracle Enterprise Manager 10g Database Control
- The SQL statements `GRANT` and `REVOKE`

Privileges are granted to and revoked from roles using the same options. Roles can also be granted to and revoked from users using the operating system that executes Oracle, or through network services.

> **See Also:** For more information about
>
> - The Database Control, see *Oracle 2 Day DBA*.
> - Modifying users, roles, or privileges with the Database Control, see the Enterprise Manager online help.

### Who Can Grant or Revoke Roles?

Any user with the `GRANT ANY ROLE` system privilege can grant or revoke any role except a global role to or from other users or roles of the database. You should grant this system privilege conservatively because it is very powerful.

Any user granted a role with the `ADMIN OPTION` can grant or revoke that role to or from other users or roles of the database. This option allows administrative powers for roles on a selective basis.

> **See Also:** *Oracle Database Administrator's Guide* for information about global roles

## Security Domains of Roles and Users

Each role and user has its own unique security domain. A role's security domain includes the privileges granted to the role plus those privileges granted to any roles that are granted to the role.

A user's security domain includes privileges on all schema objects in the corresponding schema, the privileges granted to the user, and the privileges of roles granted to the user that are **currently enabled**. (A role can be simultaneously enabled for one user and disabled for another.) A user's security domain also includes the privileges and roles granted to the user group `PUBLIC`.

## PL/SQL Blocks and Roles

The use of roles in a PL/SQL block depends on whether it is an anonymous block or a named block (stored procedure, function, or trigger), and whether it executes with definer's rights or invoker's rights.

### Named Blocks with Definer's Rights

All roles are disabled in any named PL/SQL block (stored procedure, function, or trigger) that executes with definer's rights. Roles are not used for privilege checking and you cannot set roles within a definer's rights procedure.

The `SESSION_ROLES` view shows all roles that are currently enabled. If a named PL/SQL block that executes with definer's rights queries `SESSION_ROLES`, the query does not return any rows.

**See Also:** *Oracle Database Reference*

### Anonymous Blocks with Invoker's Rights

Named PL/SQL blocks that execute with invoker's rights and anonymous PL/SQL blocks are executed based on privileges granted through enabled roles. Current roles are used for privilege checking within an invoker's rights PL/SQL block, and you can use dynamic SQL to set a role in the session.

**See Also:**

- *PL/SQL User's Guide and Reference* for an explanation of invoker's and definer's rights
- "Dynamic SQL in PL/SQL" in *Oracle Database Concepts*

## Data Definition Language Statements and Roles

A user requires one or more privileges to successfully execute a data definition language (DDL) statement, depending on the statement. For example, to create a table, the user must have the `CREATE TABLE` or `CREATE ANY TABLE` system privilege. To create a view of another user's table, the creator requires the `CREATE`

VIEW or CREATE ANY VIEW system privilege and either the SELECT *object* privilege for the table or the SELECT ANY TABLE system privilege.

Oracle avoids the dependencies on privileges received by way of roles by restricting the use of specific privileges in certain DDL statements. The following rules outline these privilege restrictions concerning DDL statements:

- All system privileges and schema object privileges that permit a user to perform a DDL operation are usable when received through a role. For example:
  - System Privileges: the CREATE TABLE, CREATE VIEW and CREATE PROCEDURE privileges.
  - Schema Object Privileges: the ALTER and INDEX privileges for a table.

  *Exception:* The REFERENCES object privilege for a table cannot be used to define a table's foreign key if the privilege is received through a role.
- All system privileges and object privileges that allow a user to perform a DML operation that is required to issue a DDL statement are *not* usable when received through a role. For example:
  - A user who receives the SELECT ANY TABLE system privilege or the SELECT *object* privilege for a table through a role can use neither privilege to create a view on another user's table.

The following example further clarifies the permitted and restricted uses of privileges received through roles:

Assume that a user is:

- Granted a role that has the CREATE VIEW system privilege
- Granted a role that has the SELECT *object* privilege for the employees table, but the user is indirectly granted the SELECT *object* privilege for the employees table
- Directly granted the SELECT *object* privilege for the departments table

Given these directly and indirectly granted privileges:

- The user can issue SELECT statements on both the employees and departments tables.
- Although the user has both the CREATE VIEW and SELECT privilege for the employees table through a role, the user cannot create a usable view on the employees table, because the SELECT *object* privilege for the employees table was granted through a role. Any views created will produce errors when accessed.

- The user can create a view on the `departments` table, because the user has the `CREATE VIEW` privilege through a role and the `SELECT` privilege for the `departments` table directly.

## Predefined Roles

The following roles are defined automatically for Oracle databases:

- `CONNECT`
- `RESOURCE`
- `DBA`
- `EXP_FULL_DATABASE`
- `IMP_FULL_DATABASE`

These roles are provided for backward compatibility to earlier versions of Oracle and can be modified in the same manner as any other role in an Oracle database.

**Note:** Each installation should create its own roles and assign only those privileges that are needed. For example, it is unwise to grant CONNECT if all that is needed is CREATE SESSION, since CONNECT includes several additional privileges: see Table 10–1, " Predefined Roles", in Chapter 10, "Administering User Privileges, Roles, and Profiles". Creating its own roles gives an organization detailed control of the privileges it assigns, and protects it in case Oracle were to change or remove roles that it defines.

## The Operating System and Roles

In some environments, you can administer database security using the operating system. The operating system can be used to manage the granting (and revoking) of database roles and to manage their password authentication. This capability is not available on all operating systems.

**See Also:** Your operating system specific Oracle documentation for details on managing roles through the operating system

## Roles in a Distributed Environment

When you use roles in a distributed database environment, you must ensure that all needed roles are set as the default roles for a distributed (remote) session. Those roles cannot be enabled when you connect to a remote database from within a local database session. For example, you cannot execute a remote procedure that attempts to enable a role at the remote site.

> **See Also:** *Oracle Database Heterogeneous Connectivity Administrator's Guide*

## Secure Application Roles

Oracle provides secure application roles, which are roles that can only be enabled by authorized PL/SQL packages. This mechanism restricts the enabling of such roles to the invoking application.

Security is strengthened when passwords are not embedded in application source code or stored in a table. Instead, a secure application role can be created, specifying which PL/SQL package is authorized to enable the role. Package identity is used to determine whether privileges are sufficient to enable the roles. Before enabling the role, the application can perform authentication and customized authorization, such as checking whether the user has connected through a proxy.

> **Note:** Because of the restriction that users cannot change security domain inside definer's right procedures, secure application roles can only be enabled inside invoker's right procedures.

### Creation of Secure Application Roles

Secure application roles are created by using the statement `CREATE ROLE` ... `IDENTIFIED USING`. Here is an example:

```
CREATE ROLE admin_role IDENTIFIED USING hr.admin;
```

This statement indicates the following:

- The role `admin_role` to be created is a secure application role.
- The role can only be enabled by modules defined inside the PL/SQL package `hr.admin`.

You must have the system privilege `CREATE ROLE` to execute this statement.

When such a role is assigned to a user, it becomes a default role for that user, automatically enabled at login without resorting to the package. A user with a default role does not have to be authenticated in any way to use the role; for example, the password for the role is not requested or required.

To restrict the role solely to the use specified by the IDENTIFIED USING clause, you can take either of the following actions:

- Immediately after granting such a role to a user, issue an ALTER USER statement with the clause DEFAULT ROLE ALL EXCEPT *role*, substituting the application role for *role*. Then *role* can only be used by applications executing the authorized package.
- When assigning roles, use GRANT ALL EXCEPT *role*.

Roles that are enabled inside an Invoker's Right procedure remain in effect even after the procedure exits. Therefore, you can have a dedicated procedure that deals with enabling the role for the rest of the session to use.

**See Also:**

- *Oracle Database SQL Reference*
- *PL/SQL User's Guide and Reference*
- *PL/SQL Packages and Types Reference*
- *Oracle Database Application Developer's Guide - Fundamentals*

## User Resource Limits

You can set limits on the amount of various system resources available to each user as part of that user's security domain. By doing so, you can prevent the uncontrolled consumption of valuable system resources such as CPU time.

This resource limit feature is very useful in large, multiuser systems, where system resources are very expensive. Excessive consumption of these resources by one or more users can detrimentally affect the other users of the database. In single-user or small-scale multiuser database systems, the system resource feature is not as important, because users' consumption of system resources is less likely to have detrimental impact.

You manage a user's resource limits by means of the Database Resource Manager. You can set password management preferences using profiles, either set individually or using a default profile for many users. Each Oracle database can have an unlimited number of profiles. Oracle allows the security administrator to enable or disable the enforcement of profile resource limits universally.

**See Also:**

- For resource management, see the *Database Resource Manager*.
- For passwords, see Password Management Policy on page 7-12

Setting resource limits causes a slight performance degradation when users create sessions, because Oracle loads all resource limit data for each user upon each connection to the database.

**See Also:** *Oracle Database Administrator's Guide* for information about security administrators.

Resource limits and profiles are discussed in the following sections:

- Types of System Resources and Limits
- Profiles

## Types of System Resources and Limits

Oracle can limit the use of several types of system resources, including CPU time and logical reads. In general, you can control each of these resources at the session level, the call level, or both, as discussed in the following subsections:

- Session Level
- Call Level
- CPU Time
- Logical Reads
- Limiting Other Resources

### Session Level

Each time a user connects to a database, a session is created. Each session consumes CPU time and memory on the computer that executes Oracle. You can set several resource limits at the session level.

If a user exceeds a session-level resource limit, Oracle terminates (rolls back) the current statement and returns a message indicating the session limit has been reached. At this point, all previous statements in the current transaction are intact, and the only operations the user can perform are `COMMIT`, `ROLLBACK`, or disconnect (in this case, the current transaction is committed). All other operations produce an error. Even after the transaction is committed or rolled back, the user can accomplish no more work during the current session.

### Call Level

Each time a SQL statement is executed, several steps are taken to process the statement. During this processing, several calls are made to the database as part of the different execution phases. To prevent any one call from using the system excessively, Oracle lets you set several resource limits at the call level.

If a user exceeds a call-level resource limit, Oracle halts the processing of the statement, rolls back the statement, and returns an error. However, all previous statements of the current transaction remain intact, and the user's session remains connected.

### CPU Time

When SQL statements and other types of calls are made to Oracle, an amount of CPU time is necessary to process the call. Average calls require a small amount of CPU time. However, a SQL statement involving a large amount of data or a runaway query can potentially consume a large amount of CPU time, reducing CPU time available for other processing.

To prevent uncontrolled use of CPU time, you can limit the CPU time for each call and the total amount of CPU time used for Oracle calls during a session. The limits are set and measured in CPU one-hundredth seconds (0.01 seconds) used by a call or a session.

### Logical Reads

Input/output (I/O) is one of the most expensive operations in a database system. SQL statements that are I/O intensive can monopolize memory and disk use and cause other database operations to compete for these resources.

To prevent single sources of excessive I/O, Oracle let you limit the logical data block reads for each call and for each session. Logical data block reads include data block reads from both memory and disk. The limits are set and measured in number of block reads performed by a call or during a session.

### Limiting Other Resources

Oracle also provides for limiting several other resources at the session level:

- You can limit the number of **concurrent sessions for each user**. Each user can create only up to a predefined number of concurrent sessions.

- You can limit the **idle time** for a session. If the time between Oracle calls for a session reaches the idle time limit, the current transaction is rolled back, the session is aborted, and the resources of the session are returned to the system. The next call receives an error that indicates the user is no longer connected to the instance. This limit is set as a number of elapsed minutes.

  **Note:** Shortly after a session is aborted because it has exceeded an idle time limit, the process monitor (PMON) background process cleans up after the aborted session. Until PMON completes this process, the aborted session is still counted in any session / user resource limit.

- You can limit the elapsed connect time for each session. If a session's duration exceeds the elapsed time limit, the current transaction is rolled back, the session is dropped, and the resources of the session are returned to the system. This limit is set as a number of elapsed minutes.

  **Note:** Oracle does not constantly monitor the elapsed idle time or elapsed connection time. Doing so would reduce system performance. Instead, it checks every few minutes. Therefore, a session can exceed this limit slightly (for example, by five minutes) before Oracle enforces the limit and aborts the session.

- You can limit the amount of private SGA space (used for private SQL areas) for a session. This limit is only important in systems that use the shared server configuration. Otherwise, private SQL areas are located in the PGA. This limit is set as a number of bytes of memory in an instance's SGA. Use the characters **K** or **M** to specify kilobytes or megabytes.

  **See Also:** For instructions on enabling / disabling resource limits:

  - *Viewing Information About Database Users and Profiles on page 10-9*
  - Managing User Roles on page 10-20
  - *Oracle Database Administrator's Guide*

## Profiles

In general, the word "profile" refers to a collection of attributes that apply to a user, enabling a single point of reference for any of multiple users that share the those exact attributes. User profiles in Oracle Internet Directory contain a wide range of attributes pertinent to directory usage and authentication for each user. Similarly, profiles in Oracle Label Security contain attributes useful in label security user administration and operations management. Profile attributes can include restrictions on system resources, but for that purpose the Database Resource Manager is preferred.

**See Also:**

- For resources, see discussion of the *Database Resource Manager* in the Oracle Database Administrator's Guide.
- For viewing resource information, see *Viewing Information About Database Users and Profiles* on page 10-9.
- For password policies, see Password Management Policy in Chapter 7, "Security Policies".

### Determining Values for Resource Limits

Before creating profiles and setting the resource limits associated with them, you should determine appropriate values for each resource limit. You can base these values on the type of operations a typical user performs. For example, if one class of user does not normally perform a high number of logical data block reads, then set the `LOGICAL_READS_PER_SESSION` and `LOGICAL_READS_PER_CALL` limits conservatively.

Usually, the best way to determine the appropriate resource limit values for a given user profile is to gather historical information about each type of resource usage. For example, the database or security administrator can use the `AUDIT SESSION` clause to gather information about the limits `CONNECT_TIME`, `LOGICAL_READS_PER_SESSION`, and `LOGICAL_READS_PER_CALL`.

You can gather statistics for other limits using the Monitor feature of Oracle Enterprise Manager (or SQL*Plus), specifically the Statistics monitor.

**See Also:**

- Chapter 8, "Database Auditing: Security Considerations"
- For more information about the Database Control, see *Oracle 2 Day DBA*.
- For information about the Monitor feature, see the Enterprise Manager online help.

# 6 Access Controls on Tables, Views, Synonyms, or Rows

The authentication processes described in Chapter 4 validate the identities of the entities using your networks, databases, and applications

The authorization processes described in Chapter 5 provide limits to their access and actions, limits that are linked to their identities and roles.

This chapter describes restrictions associated not with users but with objects, providing protection regardless of the entity who seeks, by whatever means, to access or alter them.

You provide object protections using object-level privileges and views, as well as by designing and using policies to restrict access to specific tables, views, synonyms, or rows. This level of control, which enables you to use application context with fine-grained access control, is called Virtual Private Database, or VPD. Such policies invoke functions that you design to specify dynamic predicates establishing the restrictions. You can also group established policies, applying a policy group to a particular application.

Having established such protections, you need to be notified when they are threatened or breached. Auditing capabilities enable you to receive notification of activities you want watched, and to investigate in advance of or in response to being notified. Given notification, you can strengthen your defenses and deal with the consequences of inappropriate actions and the entities who caused them. Oracle's auditing facilities are introduced in Chapter 8, "Database Auditing: Security Considerations" and described in detail in Chapter 11, "Configuring and Administering Auditing".

This chapter describes Oracle's access control capabilities in the following sections:

- Introduction to Views

- Fine-Grained Access Control
- Security Followup: Auditing as well as Prevention

**See Also:**

- Regarding policies, see Chapter 7, "Security Policies".
- Regarding application development considerations on policies, see Chapter 12, "Introducing Database Security for Application Developers", Chapter 13, and Chapter 14, "Implementing Application Context and Fine-Grained Access Control"
- Regarding fine-grained access control, see Chapter 13, "Using Virtual Private Database to Implement Application Security Policies", about developing applications to configure and administer such controls.
- Regarding security conditions for auditing, see Chapter 8.
- Regarding how to configure and administer auditing features and mechanisms for both standard users and DBAs, see Chapter 11.

## Introduction to Views

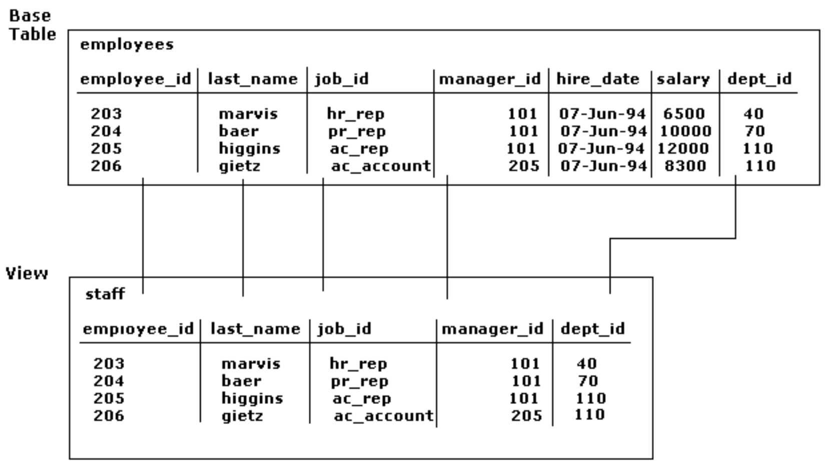
A view is a presentation of data selected from one or more tables (possibly including other views). In addition to showing the selected data, a view also shows the structure of the underlying tables, and can be thought of as the result of a stored query.

The view contains no actual data but rather derives what it shows from the tables and views on which it is based. A view can be queried, and the data it represents can be changed. Data in a view can be updated or deleted, and new data inserted. These operations directly alter the tables on which the view is based and are subject to the integrity constraints and triggers of the base tables.

For example, a base table of all employee data may have several columns and numerous rows of information. If you want users to see only specific columns, you can create a view of that table, containing only the allowable columns. You can then grant other users access to the new view, while disallowing access to the base table.

Figure 6–1 shows an example of a view called staff derived from the base table employees. Notice that the view shows only five of the columns in the base table.

***Figure 6–1 An Example of a View***



employees

| employee_id | last_name | job_id | manager_id | hire_date | salary | dept_id |
|---|---|---|---|---|---|---|
| 203 | marvis | hr_rep | 101 | 07-Jun-94 | 6500 | 40 |
| 204 | baer | pr_rep | 101 | 07-Jun-94 | 10000 | 70 |
| 205 | higgins | ac_rep | 101 | 07-Jun-94 | 12000 | 110 |
| 206 | gietz | ac_account | 205 | 07-Jun-94 | 8300 | 110 |

staff

| employee_id | last_name | job_id | manager_id | dept_id |
|---|---|---|---|---|
| 203 | marvis | hr_rep | 101 | 40 |
| 204 | baer | pr_rep | 101 | 70 |
| 205 | higgins | ac_rep | 101 | 110 |
| 206 | gietz | ac_account | 205 | 110 |

As discussed extensively in Chapter 5, a **schema object privilege** is a privilege or right to perform a particular action on a specific schema object. Different object privileges are available for different types of schema objects. Privileges related to views are discussed in that chapter, in the View Privileges section. Some schema objects, such as clusters, indexes, triggers, and database links, do not have associated object privileges. Their use is controlled with system privileges. For example, to alter a cluster, a user must own the cluster or have the ALTER ANY CLUSTER system privilege.

All these privileges, including those for tables, views, procedures, types and more, are introduced in Chapter 5's section entitled Introduction to Privileges. The tools and processes for managing these security facilities are discussed in Chapter 10, "Administering User Privileges, Roles, and Profiles".

In some circumstances, a finer level of access control is needed for tables or rows and the possible actions on them, sometimes associated with particular applications. When such controls are needed, Oracle's fine-grained access control capabilities can be used, as described in the next section.

## Fine-Grained Access Control

Fine-grained access control enables you to use functions to implement security policies and to associate those security policies with tables, views, or synonyms.

**See Also:** Using application context with fine-grained access control is called Virtual Private Database, or VPD. See these references:

- Application Context on page 6-6.
- Chapter 13, "Using Virtual Private Database to Implement Application Security Policies"
- Chapter 14, "Implementing Application Context and Fine-Grained Access Control"

The database server automatically enforces your security policies, no matter how the data is accessed, including, for example, through an application by ad hoc queries.

Fine-grained access control enables you to use all of the following capabilities:

- Limit access at the row level, using different policies for `SELECT`, `INSERT`, `UPDATE`, and `DELETE`.
- Use security policies only where you need them (for example, on salary information).
- Invoke a policy only if a particular column is referenced.
- Restrict access using a combination of row- and column-level controls, by applying a VPD policy to a view.
- Have some policies that are *always* applied, called static policies, and others that can change during execution, called dynamic policies (see Application Context).
- Use more than one policy for each table, including building on top of base policies in packaged applications.
- Distinguish policies between different applications, by using *policy groups*. Each policy group is a set of policies that belong to an application.
- Distinguish and control the use of `INDEX`, in row level security policies.
- Designate an application context, called a *driving context*, to indicate the policy group in effect. When tables, views, or synonyms are accessed, the fine-grained access control engine looks up the driving context to determine the policy group in effect and enforces all the associated policies that belong to that policy group.

The PL/SQL package DBMS_RLS let you administer your security policies. Using this package, you can add, drop, enable, disable, and refresh the policies (or policy groups) you create.

**See Also:**

- The DBMS_RLS chapter in *PL/SQL Packages and Types Reference* for information about package implementation
- Chapter 13, "Using Virtual Private Database to Implement Application Security Policies"
- Chapter 14, "Implementing Application Context and Fine-Grained Access Control"
- *Oracle Database Application Developer's Guide - Fundamentals* for information and examples on establishing security policies

The following subsections describe how fine-grained access control works:

- Dynamic Predicates
- Application Context
- Dynamic Contexts

## Dynamic Predicates

A dynamic predicate for a table, view, or synonym is generated by a PL/SQL function, which you write and associate with a security policy through a PL/SQL interface.

Dynamic predicates are acquired at statement parse time, when the base table or view is referenced in a query using SELECT or a DML statement.

The function or package that implements the security policy you create returns a predicate (a WHERE condition). This predicate controls access according to the policy you specified. Rewritten queries are fully optimized and shareable.

Here is an example of such a policy:

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

Whenever the EMPLOYEES table, under the HR schema, is referenced in a query or subquery (SELECT), the server calls your EMP_SEC function (under the HR schema). This function returns a predicate (called P1 in the following section) defined in the function, which in this example could be specific to the current user for the EMP_POLICY policy. Your policy function can generate the predicates based on the session environment variables available during the function call, that is, from the application context as described in the next section. The policy can specify any combination of security-relevant columns and any combination of these statement types: SELECT, INSERT, UPDATE, DELETE, or INDEX. You can also specify whether the result of an INSERT or UPDATE should immediately be checked against the policy.

The server then produces a transient view, with the text:

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

Here, P1 (for example, where SAL > 10000, or even a subquery) is the predicate returned from your EMP_SEC function. The server treats the EMPLOYEES table as a view and does the view expansion just like the ordinary view, except that the view text is taken from the transient view instead of the data dictionary.

The policy function creates a WHERE clause relevant to the current user by using information from the set of session environment variables called application context.

## Application Context

Application context helps you apply fine-grained access control because you can link your function-based security policies with applications.

Oracle provides a built-in application context namespace, USERENV, which provides access to predefined attributes. These attributes are session primitives--information that the database automatically captures regarding a user's session. For example, the IP address from which a user connected, the username, and a proxy username (in cases where a user connection is proxied through a middle tier), are all available as predefined attributes through the USERENV application context.

Each application has its own application-specific context, which users cannot arbitrarily change (for example, through SQL*Plus). Context attributes are accessible to the functions implementing your security policies.

For example, context attributes you could use from a human resources application could include "position," "organizational unit," and "country." Attributes available from an order-entry control system might include "customer number" and "sales region".

Application contexts thus permit flexible, parameter-based access control using context attributes relevant to an application and to policies you might want to create for controlling its use.

You can:

- Base predicates on context values
- Use context values within predicates, as bind variables
- Set user attributes
- Access user attributes

To define an application context:

1. Create a PL/SQL package with functions that validate and set the context for your application. You may want to use an event trigger on login to set the initial context for logged-in users.
2. Use `CREATE CONTEXT` to specify a unique context name and associate it with the PL/SQL package you created.
3. Then do either of the following:
   - Reference the application context within the policy function implementing fine-grained access control.
   - Create an event trigger on login to set the initial context for a user. For example, you could query a user's employee number and set this as an "employee number" context value.
4. Reference the application context. For example, to limit customers to seeing their own records only, use fine-grained access control to dynamically modify the user's query from `SELECT * FROM Orders_tab` to the following:

   ```
   SELECT * FROM Orders_tab
      WHERE Custno = SYS_CONTEXT ('order_entry', 'cust_num');
   ```

**See Also:** Regarding how applications configure, administer, and use application context, see

- Chapter 13, "Using Virtual Private Database to Implement Application Security Policies"
- Chapter 14, "Implementing Application Context and Fine-Grained Access Control"
- *PL/SQL User's Guide and Reference*
- *PL/SQL Packages and Types Reference*
- *Oracle Database Application Developer's Guide - Fundamentals*

The next subsection, Dynamic Contexts, describes run-time efficiencies you can establish by identifying how dynamic each of your policies is, using these categories: static, shared, context-sensitive, or dynamic.

## Dynamic Contexts

When you create a policy, you can establish run-time efficiencies by specifying whether the policy is static, shared, context-sensitive, or dynamic:

*Table 6–1 Policy Types and Run-Time Efficiencies*

| Policy Type | Predicate and Policy Function | Description and Operational Explanation |
|---|---|---|
| Static | Same predicate string for anyone accessing the object | Executed once and cached in SGA. Policies for statements accessing the same object do not reexecute the policy function, but use the cached predicate instead. |
| Shared-static | Same as static, except the policy can be shared across multiple objects | Ideal for data partitions in hosting environments because almost all objects share the same function and the policy is static. Also executed once and cached in SGA, but the server first looks for a cached predicate generated by the same policy function of the same policy type. |
| Context-sensitive<br><br>and<br><br>shared context-sensitive | Policy function executed when statement parsed, but value returned is not cached | The policy function is not re-evaluated at statement execution time unless the server detects context changes since the last use of the cursor. (For session pooling where multiple clients share a database session, the middle tier must reset context during client switches.)<br><br>When a context-sensitive policy is labeled "shared," the server first looks for a cached predicate generated by the same policy function of the same policy type within the same database session.<br><br>If the predicate is found in the session memory, the policy function is not re-executed and the cached value is valid until session private application context changes occur. |
| Dynamic | Policy function always re-executed on each statement parsing or execution | Server assumes the predicate may be affected by any system or session environment at any time.<br><br>Dynamic is the system default. If no policy type is specified when DBMS_RLS.ADD_POLICY is called, dynamic is assumed. |

**See also:** The section titled How to Add a Policy to a Table, View, or Synonym in Chapter 14, "Implementing Application Context and Fine-Grained Access Control".

## Security Followup: Auditing as well as Prevention

Even after designing and implementing protective measures using privileges, views, and policies, you want to know when these measures are threatened or breached. Auditing can notify you of suspicious or questionable activities. You can then investigate, strengthen your defenses, and deal with inappropriate actions, consequences, and security offenders.

Use auditing to complement your access controls from several perspectives:

- Audit the data you considered important enough to protect with database security mechanisms like access controls.
- Audit as a way of verifying that your access control mechanisms are implemented properly and working as you intended.
- Design audit policies that you expect will never actually "fire" because your other security mechanisms (authentication, authorization, access controls) should be protecting that data. Then if such an audit policy does fire, you are alerted that you have a security breach. It may mean, for example, that your security protections are not operating as you expected them to in protecting the data.

Chapter 8, "Database Auditing: Security Considerations" introduces Oracle's auditing facilities, and Chapter 11, "Configuring and Administering Auditing" describes them in detail.

7

# Security Policies

The idea of security policies includes many dimensions. Broad considerations include requiring backups to be done regularly and stored off-site. Narrow table or data considerations include ensuring that unauthorized access to sensitive data, such as employee salaries, is precluded by built-in restrictions on every type of access to the table that contains them.

This chapter discusses security policies in the following sections:

- System Security Policy
- Data Security Policy
- User Security Policy
- Password Management Policy
- Auditing Policy
- A Security Checklist

## System Security Policy

This section describes aspects of system security policy, and contains the following topics:

- Database User Management
- User Authentication
- Operating System Security

Each database has one or more administrators who are responsible for maintaining all aspects of the security policy: the security administrators. If the database system is small, the database administrator may have the responsibilities of the security administrator. However, if the database system is large, a special person or group of people may have responsibilities limited to those of a security administrator.

After deciding who will manage the security of the system, a security policy must be developed for every database. A database's security policy should include several sub-policies, as explained in the following sections.

## Database User Management

Database users are the access paths to the information in an Oracle database. Therefore, tight security should be maintained for the management of database users. Depending on the size of a database system and the amount of work required to manage database users, the security administrator may be the only user with the privileges required to create, alter, or drop database users. On the other hand, there may be a number of administrators with privileges to manage database users. Regardless, only trusted individuals should have the powerful privileges to administer database users.

**See Also:** *Oracle Database Administrator's Guide*

## User Authentication

Database users can be **authenticated** (verified as the correct person) by Oracle using database passwords, the host operating system, network services, or by Secure Sockets Layer (SSL).

**Note:** To be authenticated using network authentication services or SSL, requires that you have installed Oracle Advanced Security. Refer to the *Oracle Advanced Security Administrator's Guide* for information about these types of authentication.

User authentication and how it is specified is discussed in "User Authentication Methods" on page 9-1.

## Operating System Security

The following security issues must also be considered for the operating system environment executing Oracle and any database applications:

- Database administrators must have the operating system privileges to create and delete files.

- Typical database users should not have the operating system privileges to create or delete files related to the database.
- If the operating system identifies database roles for users, the security administrators must have the operating system privileges to modify the security domain of operating system accounts.

**See Also:** Your operating-system-specific Oracle documentation contains more information about operating system security issues

## Data Security Policy

**Data security** includes the mechanisms that control the access to and use of the database at the object level. Your data security policy determines which users have access to a specific schema object, and the specific types of actions allowed for each user on the object. For example, the policy could establish that user `scott` can issue `SELECT` and `INSERT` statements but not `DELETE` statements using the `emp` table. Your data security policy should also define the actions, if any, that are audited for each schema object.

Your data security policy is determined primarily by the level of security you want to establish for the data in your database.For example, it may be acceptable to have little data security in a database when you want to allow any user to create any schema object, or grant access privileges for their objects to any other user of the system. Alternatively, it might be necessary for data security to be very controlled when you want to make a database or security administrator the only person with the privileges to create objects and grant access privileges for objects to roles and users.

Overall data security should be based on the sensitivity of data. If information is not sensitive, then the data security policy can be more lax. However, if data is sensitive, a security policy should be developed to maintain tight control over access to objects.

Some means of implementing data security include system and object privileges, and through roles. A role is a set of privileges grouped together that can be granted to users. Privileges and roles are discussed in Chapter 10, "Administering User Privileges, Roles, and Profiles".

Views can also implement data security because their definition can restrict access to table data. They can exclude columns containing sensitive data.

Another means of implementing data security is through fine-grained access control and use of an associated application context. Fine-grained access control is a feature of Oracle Database that enables you to implement security policies with functions, and to associate those security policies with tables or views. In effect, the security policy function generates a `WHERE` condition that is appended to relevant SQL statements, thereby restricting user access to rows of data in the table or view. An application context is a secure data cache for storing information used to make access control decisions.

**See Also:**

- Introduction to Views in Chapter 6
- Introduction to Fine-Grained Access Control in Chapter 13
- Introduction to Application Context in Chapter 13

## User Security Policy

This section describes aspects of user security policy, and contains the following topics:

- General User Security
- End-User Security
- Administrator Security
- Application Developer Security
- Application Administrator Security

### General User Security

For all types of database users, consider the following general user security issues:

- Password Security
- Privilege Management

#### Password Security

If user authentication is managed by the database, security administrators should develop a password security policy to maintain database access security. For example, database users should be required to change their passwords at regular intervals, and of course, when their passwords are revealed to others. By forcing a user to modify passwords in such situations, unauthorized database access can be reduced.

Passwords are always automatically and transparently encrypted during network (client/server and server/server) connections, using a modified DES (Data Encryption Standard) algorithm, before sending them across the network.

### Privilege Management

Security administrators should consider issues related to privilege management for all types of users. For example, in a database with many usernames, it may be beneficial to use roles (which are named groups of related privileges that you grant to users or other roles) to manage the privileges available to users. Alternatively, in a database with a handful of usernames, it may be easier to grant privileges explicitly to users and avoid the use of roles.

Security administrators managing a database with many users, applications, or objects should take advantage of the benefits offered by roles. Roles greatly simplify the task of privilege management in complicated environments.

## End-User Security

Security administrators must define a policy for end-user security. If a database has many users, the security administrator can decide which groups of users can be categorized into user groups, and then create user roles for these groups. The security administrator can grant the necessary privileges or application roles to each user role, and assign the user roles to the users. To account for exceptions, the security administrator must also decide what privileges must be explicitly granted to individual users.

### Using Roles for End-User Privilege Management

Roles are the easiest way to grant and manage the common privileges needed by different groups of database users.

Consider a situation where every user in the accounting department of a company needs the privileges to run the accounts receivable and accounts payable database applications (ACCTS_REC and ACCTS_PAY). Roles are associated with both applications, and they contain the object privileges necessary to execute those applications.

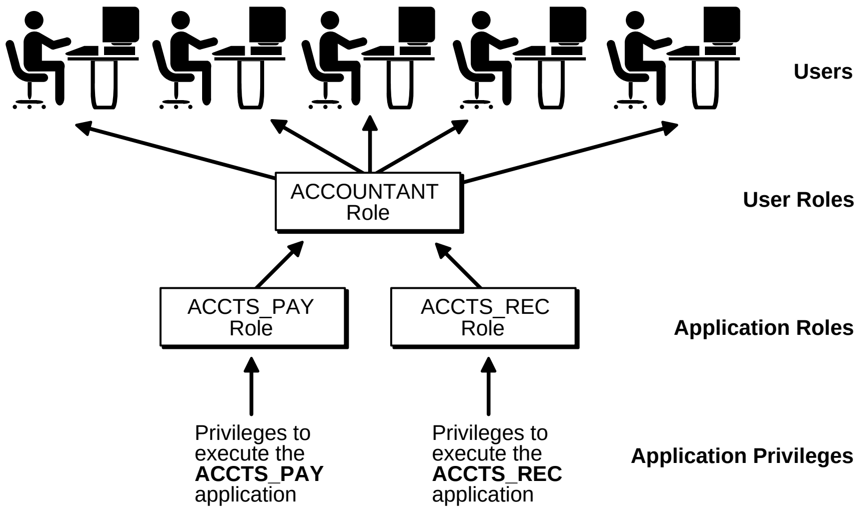
The following actions, performed by the database or security administrator, address this simple security situation:

1. Create a role named `accountant`.

2. Grant the roles for the ACCTS_REC and ACCTS_PAY database applications to the `accountant` role.
3. Grant each user of the accounting department the `accountant` role.

This security model is illustrated in Figure 7–1.

***Figure 7–1 User Role***



This plan addresses the following potential situations:

- If accountants subsequently need a role for a new database application, that application's role can be granted to the `accountant` role, and all users in the accounting department will automatically receive the privileges associated with the new database application. The application's role does not need to be granted to individual users requiring use of the application.
- Similarly, if the accounting department no longer requires the need for a specific application, the application's role can be dropped from the `accountant` role.
- If the privileges required by the ACCTS_REC and ACCTS_PAY applications change, the new privileges can be granted to, or revoked from, the application's role. The security domain of the `accountant` role, and all users granted the `accountant` role, automatically reflect the privilege modification.

Use roles in all possible situations to make end-user privilege management efficient and simple.

### Using a Directory Service for End-User Privilege Management

You can also manage users and their authorizations centrally, in a directory service, through the enterprise user and enterprise role features of Oracle Advanced Security. See the *Oracle Advanced Security Administrator's Guide* for information about this functionality.

## Administrator Security

Security administrators should have a policy addressing database administrator security. For example, when the database is large and there are several types of database administrators, the security administrator may decide to group related administrative privileges into several administrative roles. The administrative roles can then be granted to appropriate administrator users. Alternatively, when the database is small and has only a few administrators, it may be more convenient to create one administrative role and grant it to all administrators.

### Protection for Connections as SYS and SYSTEM

After database creation, and if you used the default passwords for SYS and SYSTEM, *immediately* change the passwords for the SYS and SYSTEM administrative usernames. Connecting as SYS or SYSTEM gives a user powerful privileges to modify a database. For example, connecting as SYS allows a user to alter data dictionary tables. The privileges associated with these usernames are extremely sensitive, and should only be available to select database administrators.

If you have installed options that have caused other administrative usernames to be created, such username accounts are initially created locked. To unlock these accounts, use the ALTER USER statement. The ALTER USER statement should also be used to change the associated passwords for these accounts.

The passwords for these accounts can be modified using the procedures described in "Altering Users" on page 10-7.

### Protection for Administrator Connections

Only database administrators should have the capability to connect to a database with administrative privileges. For example:

```
CONNECT username/password AS SYSDBA/SYSOPER
```

Connecting as SYSOPER gives a user the ability to perform basic operational tasks (such as STARTUP, SHUTDOWN, and recovery operations). Connecting as SYSDBA gives the user these abilities plus unrestricted privileges to do anything to a database or the objects within a database (including, CREATE, DROP, and DELETE). Connecting as SYSDBA places a user in the SYS schema, where he can alter data dictionary tables.

**Notes:**

- Connections requested AS SYSDBA or AS SYSOPER must use these phrases; without them, the connection fails. The Oracle parameter 07_DICTIONARY_ACCESSIBILITY is set to FALSE by default, to limit sensitive data dictionary access only to those authorized.
- Such connections are authorized only after verification with the password file or with the operating system privileges and permissions. If operating system authentication is used, the database does not use the supplied username/password. Operating system authentication is used if there is no password file, or if the supplied username/password is not in that file, or if no username/password is supplied.
- However, if authentication succeeds by means of the password file, the connection is logged with the username; if authentication succeeds by means of the operating system, it's a CONNECT / connection that does not record the specific user.

### Using Roles for Administrator Privilege Management

Roles are the easiest way to restrict the powerful system privileges and roles required by personnel administrating the database.

Consider a scenario where the database administrator responsibilities at a large installation are shared among several database administrators, each responsible for the following specific database management jobs:

- Object creation and maintenance
- Database tuning and performance
- Creation of new users and granting roles and privileges to database users
- Routine database operation (for example: STARTUP, SHUTDOWN, and backup and recovery operations)
- Emergency situations, such as database recovery

There are also new, inexperienced database administrators needing limited capabilities to experiment with database management

In this scenario, the security administrator should structure the security for administrative personnel as follows:

1. Define six roles to contain the distinct privileges required to accomplish each type of job (for example, `dba_objects`, `dba_tune`, `dba_security`, `dba_maintain`, `dba_recov`, `dba_new`).
2. Grant each role the appropriate privileges.
3. Grant each type of database administrator the corresponding role.

This plan diminishes the likelihood of future problems in the following ways:

- If a database administrator's job description changes to include more responsibilities, that database administrator can be granted other administrative roles corresponding to the new responsibilities.
- If a database administrator's job description changes to include fewer responsibilities, that database administrator can have the appropriate administrative roles revoked.
- The data dictionary always stores information about each role and each user, so information is available to disclose the task of each administrator.

## Application Developer Security

Security administrators must define a special security policy for the application developers using a database. A security administrator could grant the privileges to create necessary objects to application developers. Or, alternatively, the privileges to create objects could be granted only to a database administrator, who then receives requests for object creation from developers.

### Application Developers and Their Privileges

Database application developers are unique database users who require special groups of privileges to accomplish their jobs. Unlike end users, developers need system privileges, such as `CREATE TABLE`, `CREATE PROCEDURE`, and so on. However, only specific system privileges should be granted to developers to restrict their overall capabilities in the database.

### The Application Developer's Environment: Test and Production Databases

In many cases, application development is restricted to test databases and is not allowed on production databases. This restriction ensures that application developers do not compete with end users for database resources, and that they cannot detrimentally affect a production database.

After an application has been thoroughly developed and tested, it is permitted access to the production database and made available to the appropriate end users of the production database.

### Free Versus Controlled Application Development

The database administrator can define the following options when determining which privileges should be granted to application developers:

- Free development

  An application developer is allowed to create new schema objects, including tables, indexes, procedures, packages, and so on. This option allows the application developer to develop an application independent of other objects.

- Controlled Development

  An application developer is not allowed to create new schema objects. All required tables, indexes, procedures, and so on are created by a database administrator, as requested by an application developer. This option allows the database administrator to completely control a database's space usage and the access paths to information in the database.

Although some database systems use only one of these options, other systems could mix them. For example, application developers can be allowed to create new stored procedures and packages, but not allowed to create tables or indexes. A security administrator's decision regarding this issue should be based on the following:

- The control desired over a database's space usage
- The control desired over the access paths to schema objects
- The database used to develop applications—if a test database is being used for application development, a more liberal development policy would be in order

### Roles and Privileges for Application Developers

Security administrators can create roles to manage the privileges required by the typical application developer. For example, a typical role named `APPLICATION_DEVELOPER` might include the `CREATE TABLE`, `CREATE VIEW`, and `CREATE`

`PROCEDURE` system privileges. Consider the following when defining roles for application developers:

- `CREATE` system privileges are usually granted to application developers so that they can create their own objects. However, `CREATE ANY` system privileges, which allow a user to create an object in any user's schema, are not usually granted to developers. This restricts the creation of new objects only to the developer's user account.
- Object privileges are rarely granted to roles used by application developers, because granting object privileges through roles often restricts their usability in creating other objects (primarily views and stored procedures). It is more practical to allow application developers to create their own objects for development purposes.

### Space Restrictions Imposed on Application Developers

While application developers are typically given the privileges to create objects as part of the development process, security administrators must maintain limits on what and how much database space can be used by each application developer. For example, as the security administrator, you should specifically set or restrict the following limits for each application developer:

- The tablespaces in which the developer can create tables or indexes
- The quota for each tablespace accessible to the developer

Both limitations can be set by altering a developer's security domain. This is discussed in "Altering Users" on page 10-7.

## Application Administrator Security

In large database systems with many database applications, you might consider assigning application administrators. An application administrator is responsible for the following types of tasks:

- Creating roles for an application and managing the privileges of each application role
- Creating and managing the objects used by a database application
- Maintaining and updating the application code and Oracle procedures and packages, as necessary

Often, an application administrator is also the application developer who designed an application. However, an application administrator could be any individual familiar with the database application.

## Password Management Policy

Database security systems that are dependent on passwords require that passwords be kept secret at all times. Since passwords are vulnerable to theft, forgery, and misuse, Oracle Database has DBAs and security officers control password management policy through user profiles, enabling greater control over database security.

You use the `CREATE PROFILE` statement to create a user profile. The profile is assigned to a user with the `CREATE USER` or `ALTER USER` statement. Details of creating and altering database users are not discussed in this section. This section is concerned with the password parameters that can be specified using the `CREATE PROFILE` (or `ALTER PROFILE`) statement.

This section contains the following topics relating to Oracle password management:

- Account Locking
- Password Aging and Expiration
- Password History
- Password Complexity Verification

**See Also:**

- "Managing Resources with Profiles" on page 10-13
- "Managing Oracle Users" on page 10-1
- *Oracle Database SQL Reference* for syntax and specific information about SQL statements discussed in this section

### Account Locking

When a particular user exceeds a designated number of failed login attempts, the server automatically locks that user's account. You specify the permissible number of failed login attempts using the `CREATE PROFILE` statement. You can also specify the amount of time accounts remain locked.

In the following example, the maximum number of failed login attempts for the user ashwini is four, and the amount of time the account will remain locked is 30 days. The account will unlock automatically after the passage of 30 days.

```
CREATE PROFILE prof LIMIT
    FAILED_LOGIN_ATTEMPTS 4
    PASSWORD_LOCK_TIME 30;
ALTER USER ashwini PROFILE prof;
```

If you do not specify a time interval for unlocking the account, PASSWORD_LOCK_TIME assumes the value specified in a default profile. If you specify PASSWORD_LOCK_TIME as UNLIMITED, the account must be explicitly unlocked using an ALTER USER statement. For example, assuming that PASSWORD_LOCK_TIME UNLIMITED is specified for ashwini, then the following statement must be used to unlock the account:

```
ALTER USER ashwini ACCOUNT UNLOCK;
```

After a user successfully logs into an account, that user's unsuccessful login attempt count, if there is one, is reset to 0.

The security officer can also explicitly lock user accounts. When this occurs, the account cannot be unlocked automatically, and only the security officer should unlock the account. The CREATE USER or ALTER USER statements are used to explicitly lock or unlock user accounts. For example, the following statement locks user account susan:

```
ALTER USER susan ACCOUNT LOCK;
```

## Password Aging and Expiration

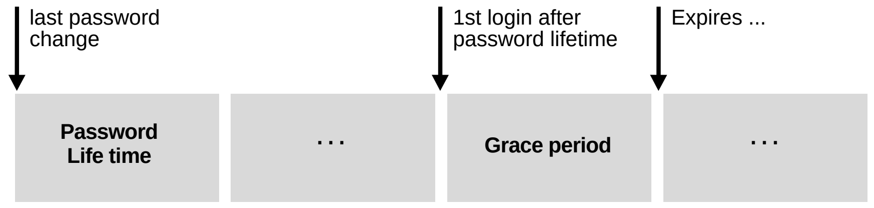
Use the CREATE PROFILE statement to specify a maximum lifetime for passwords. When the specified amount of time passes and the password expires, the user or DBA must change the password. The following statements create and assign a profile to user ashwini, and the PASSWORD_LIFE_TIME clause specifies that ashwini can use the same password for 90 days before it expires.

```
CREATE PROFILE prof LIMIT
   FAILED_LOGIN_ATTEMPTS 4
   PASSWORD_LOCK_TIME 30
   PASSWORD_LIFE_TIME 90;
ALTER USER ashwini PROFILE prof;
```

You can also specify a grace period for password expiration. Users enter the grace period upon the first attempt to log in to a database account after their password has expired. During the grace period, a warning message appears each time users try to log in to their accounts, and continues to appear until the grace period expires. Users must change the password within the grace period. If the password is not changed within the grace period, thereafter users are prompted for a new password each time an attempt is made to access their accounts. Access to an account is denied until a new password is supplied.

Figure 7–2 shows the chronology of the password lifetime and grace period.

***Figure 7–2 Chronology of Password Lifetime and Grace Period***



In the following example, the profile assigned to `ashwini` includes the specification of a grace period: `PASSWORD_GRACE_TIME = 3`. The first time `ashwini` tries to log in to the database after 90 days (this can be *any* day after the 90th day; that is, the 70th day, 100th day, or another day), she receives a warning message that her password will expire in three days. If three days pass, and she does not change her password, the password expires. Thereafter, she receives a prompt to change her password on any attempt to log in, and cannot log in until she does so.

```
CREATE PROFILE prof LIMIT
   FAILED_LOGIN_ATTEMPTS 4
   PASSWORD_LOCK_TIME 30
   PASSWORD_LIFE_TIME 90
   PASSWORD_GRACE_TIME 3;
ALTER USER ashwini PROFILE prof;
```

Oracle provides a means of explicitly expiring a password. The `CREATE USER` and `ALTER USER` statements provide this functionality. The following statement creates a user with an expired password. This setting forces the user to change the password before the user can log in to the database.

```
CREATE USER jbrown
     IDENTIFIED BY zX83yT
     ...
     PASSWORD EXPIRE;
```

## Password History

The following two parameters control the user's ability to reuse an old password:

***Table 7–1 Parameters Controlling Re-Use of an Old Password***

| Parameter Name | Description and Use |
|---|---|
| PASSWORD_REUSE_TIME | requires either<br>■ a number specifying how many days (or a fraction of a day) between the earlier use of a password and its next use, or<br>■ the word UNLIMITED. |
| PASSWORD_REUSE_MAX | requires either<br>■ an integer to specify the number of password changes required before a password can be reused, or<br>■ ·the word UNLIMITED. |

If you specify neither, the user can reuse passwords at any time, which is not a "security best practice."

If neither parameter is UNLIMITED, password reuse is allowed, but only after meeting both conditions. The user must have changed the password the specified number of times, and the specified number of days must have passed since the old password was last used.

For example, suppose user A's profile had PASSWORD_REUSE_MAX set to 10 and PASSWORD_REUSE_TIME set to 30. Then user A could not reuse a password until she had reset her password ten times, and 30 days had passed since she last used that password.

If either parameter is specified as UNLIMITED, the user can never reuse a password.

If both parameters are set to UNLIMITED, Oracle ignores both, and the user can reuse any password at any time.

**Note:** If you specify DEFAULT for either parameter, then Oracle uses the value defined in the DEFAULT profile, which by default sets all parameters to UNLIMITED. Oracle thus uses UNLIMITED for any parameter specified as DEFAULT, unless you change the setting for that parameter in the DEFAULT profile.

## Password Complexity Verification

Oracle's sample password complexity verification routine can be specified using a PL/SQL script (`UTLPWDMG.SQL`), which sets the default profile parameters.

The password complexity verification routine ensures that the password meets the following requirements:

- Is at least four characters long
- Differs from the username
- Has at least one alpha, one numeric, and one punctuation mark character
- Is not simple or obvious, such as `welcome`, `account`, `database`, or `user`
- Differs from the previous password by at least 3 characters

---

**Note:** The ALTER USER command now has a REPLACE clause whereby users can change their own unexpired passwords by supplying the old password to authenticate themselves.

If the password has expired, the user cannot log in to SQL to issue the ALTER USER command. Instead, the OCIPasswordChange() function must be used, which also requires the old password.

A DBA with ALTER ANY USER privilege can alter any user's password (force a new password) without supplying the old one.

---

### Password Verification Routine Formatting Guidelines

You can enhance the existing password verification complexity routine or create other password verification routines using PL/SQL or third-party tools.

The PL/SQL call must adhere to the following format:

```
routine_name
(
userid_parameter IN VARCHAR(30),
password_parameter IN VARCHAR (30),
old_password_parameter IN VARCHAR (30)
)
RETURN BOOLEAN
```

After a new routine is created, it must be assigned as the password verification routine using the user's profile or the system default profile.

```
CREATE/ALTER PROFILE profile_name LIMIT
PASSWORD_VERIFY_FUNCTION routine_name
```

The password verify routine must be owned by SYS.

## Sample Password Verification Routine

You can use this sample password verification routine as a model when developing your own complexity checks for a new password.

The default password complexity function performs the following minimum complexity checks:

- The password satisfies minimum length requirements.
- The password is not the username. You can modify this function based on your requirements.

This function must be created in SYS schema, and you must `connect SYS/`*`password`* `AS SYSDBA` before running the script.

```
CREATE OR REPLACE FUNCTION verify_function
(username varchar2,
   password varchar2,
   old_password varchar2)
   RETURN boolean IS
   n boolean;
   m integer;
   differ integer;
   isdigit boolean;
   ischar  boolean;
   ispunct boolean;
   digitarray varchar2(20);
   punctarray varchar2(25);
   chararray varchar2(52);

BEGIN
   digitarray:= '0123456789';
   chararray:= 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
   punctarray:='!"#$%&()''*+,-/:;<=>?_';

--Check if the password is same as the username
IF password = username THEN
   raise_application_error(-20001, 'Password same as user');
END IF;
```

```
--Check for the minimum length of the password
IF length(password) < 4 THEN
   raise_application_error(-20002, 'Password length less than 4');
END IF;

--Check if the password is too simple. A dictionary of words may be
--maintained and a check may be made so as not to allow the words
--that are too simple for the password.
IF NLS_LOWER(password) IN ('welcome', 'database', 'account', 'user',
   'password', 'oracle', 'computer', 'abcd')
   THEN raise_application_error(-20002, 'Password too simple');
END IF;

--Check if the password contains at least one letter,
--one digit and one punctuation mark.
--1. Check for the digit
--You may delete 1. and replace with 2. or 3.
isdigit:=FALSE;
m := length(password);
FOR i IN 1..10 LOOP
  FOR j IN 1..m LOOP
    IF substr(password,j,1) = substr(digitarray,i,1) THEN
      isdigit:=TRUE;
        GOTO findchar;
    END IF;
   END LOOP;
END LOOP;
IF isdigit = FALSE THEN
  raise_application_error(-20003, 'Password should contain at least one \
  digit, one character and one punctuation');
END IF;
--2. Check for the character

<<findchar>>
ischar:=FALSE;
FOR i IN 1..length(chararray) LOOP
  FOR j IN 1..m LOOP
    IF substr(password,j,1) = substr(chararray,i,1) THEN
       ischar:=TRUE;
         GOTO findpunct;
       END IF;
    END LOOP;
END LOOP;
IF ischar = FALSE THEN
  raise_application_error(-20003, 'Password should contain at least one digit,\
    one character and one punctuation');
END IF;
--3. Check for the punctuation

<<findpunct>>
ispunct:=FALSE;
FOR i IN 1..length(punctarray) LOOP
  FOR j IN 1..m LOOP
    IF substr(password,j,1) = substr(punctarray,i,1) THEN
       ispunct:=TRUE;
         GOTO endsearch;
       END IF;
   END LOOP;
END LOOP;
IF ispunct = FALSE THEN raise_application_error(-20003, 'Password should \
 contain at least one digit, one character and one punctuation');
END IF;

<<endsearch>>
--Check if the password differs from the previous password by at least 3 letters
IF old_password = '' THEN
  raise_application_error(-20004, 'Old password is null');
END IF;
--Everything is fine; return TRUE ;
differ := length(old_password) - length(password);
IF abs(differ) < 3 THEN
  IF length(password) < length(old_password) THEN
    m := length(password);
  ELSE
    m:= length(old_password);
  END IF;
  differ := abs(differ);
  FOR i IN 1..m LOOP
    IF substr(password,i,1) != substr(old_password,i,1) THEN
             differ := differ + 1;
    END IF;
  END LOOP;
  IF differ < 3 THEN
    raise_application_error(-20004, 'Password should differ by at \
      least 3 characters');
    END IF;
  END IF;
--Everything is fine; return TRUE ;
  RETURN(TRUE);
END;
```

## Auditing Policy

Security administrators should define a policy for the auditing procedures of each database. You may, for example, decide to have database auditing disabled unless questionable activities are suspected. When auditing is required, the security administrator must decide what level of detail to audit the database; usually, general system auditing is followed by more specific types of auditing after the origins of suspicious activity are determined. In addition to standard database auditing, Oracle supports fine-grained auditing using policies that can monitor multiple specific objects, columns, and statements, including INDEX.

Auditing is discussed in Chapter 8, "Database Auditing: Security Considerations" and Chapter 11, "Configuring and Administering Auditing".

## A Security Checklist

Information security and privacy and protection of corporate assets and data are of pivotal importance in any business. Oracle Database comprehensively addresses the need for information security by offering cutting-edge security features such as deep data protection, auditing, scalable security, secure hosting and data exchange.

The Oracle Database server leads the industry in security. However, in order to fully maximize the security features offered by Oracle Database in any business environment, it is imperative that the database itself be well-protected. Furthermore, proper use of its security features and adherence to basic security practices will help protect against database-related threats and attacks. Such an approach provides a much more secure operating environment for the Oracle Database database.

This security checklist provides guidance on configuring Oracle Database in a secure manner by adhering to and recommending industry-standard "best security practices" for operational database deployments.

In simple summary, before looking at the more detailed checklist: consider all paths the data travels and assess the threats that impinge on each path and node. Then take steps to lessen or eliminate both those threats and the consequences of a successful breach of security. Monitoring and auditing to detect either increased threat levels or successful penetration increases the likelihood of preventing or minimizing security losses.

Details on specific database-related tasks and actions can be found throughout the Oracle documentation set.

1. **INSTALL ONLY WHAT IS REQUIRED.**

**Options and Products**

The Oracle Database CD pack contains a host of options and products in addition to the database server. Install additional products and options only as necessary. Use Custom Installation to avoid installing unnecessary products or, following a typical installation, deinstall unneeded options and products. There is no need to maintain the additional products and options if they are not being used. They can always be properly and easily reinstalled as required.

**Sample Schemas**

Oracle Corporation provides Sample Schemas to provide a common platform for examples. If your database will be used in a production environment, do not install the Sample Schema. If you have installed the Sample Schema on a test database, then before going production, remove or re-lock the Sample Schema accounts.

2. **LOCK AND EXPIRE DEFAULT USER ACCOUNTS.**

   Oracle Database installs with a number of default (preset) database server user accounts. Upon successful installation of the database server, the Database Configuration Assistant automatically locks and expires most default database user accounts.

   If a manual (not utilizing Database Configuration Assistant) installation of Oracle Database is performed, no default database users are locked upon successful installation of the database server. If left open in their default states, these user accounts can be exploited to gain unauthorized access to data or disrupt database operations.

   Therefore, after performing any kind of initial installation that does not utilize Database Configuration Assistant, you should *lock* and *expire* all default database user accounts. Oracle Database provides SQL to perform such operations.

   Installing additional products and components later also results in creating more default database server accounts. Database Configuration Assistant automatically locks and expires all additionally created database server user accounts. Unlock only those accounts that are need to be accessed on a regular basis and assign a strong, meaningful password to each of these unlocked accounts. Oracle provides SQL and password management to perform such operations.

   Table 7–2 shows the database users after a typical Oracle Database installation utilizing Database Configuration Assistant.

*Table 7–2 Default Accounts and Their Status (Standard Installation)*

| USERNAME | ACCOUNT_STATUS |
|---|---|
| ANONYMOUS | EXPIRED & LOCKED |
| CTXSYS | EXPIRED & LOCKED |
| DBSNMP | EXPIRED & LOCKED |
| DIP | EXPIRED & LOCKED |
| DMSYS | EXPIRED & LOCKED |
| EXFSYS | EXPIRED & LOCKED |
| HR | EXPIRED & LOCKED |
| MDDATA | EXPIRED & LOCKED |
| MDSYS | EXPIRED & LOCKED |
| MGMT_VIEW | EXPIRED & LOCKED |
| ODM | EXPIRED & LOCKED |
| ODM_MTR | EXPIRED & LOCKED |
| OE | EXPIRED & LOCKED |
| OLAPSYS | EXPIRED & LOCKED |
| ORDPLUGINS | EXPIRED & LOCKED |
| ORDSYS | EXPIRED & LOCKED |
| OUTLN | EXPIRED & LOCKED |
| PM | EXPIRED & LOCKED |
| QS | EXPIRED & LOCKED |
| QS_ADM | EXPIRED & LOCKED |
| QS_CB | EXPIRED & LOCKED |
| QS_CBADM | EXPIRED & LOCKED |
| QS_CS | EXPIRED & LOCKED |
| QS_ES | EXPIRED & LOCKED |
| QS_OS | EXPIRED & LOCKED |
| QS_WS | EXPIRED & LOCKED |
| RMAN | EXPIRED & LOCKED |

*Table 7–2 (Cont.) Default Accounts and Their Status (Standard Installation)*

| USERNAME | ACCOUNT_STATUS |
|---|---|
| SCOTT | EXPIRED & LOCKED |
| SH | EXPIRED & LOCKED |
| SI_INFORMTN_SCHEMA | EXPIRED & LOCKED |
| SYS | OPEN |
| SYSMAN | EXPIRED & LOCKED |
| SYSTEM | OPEN |
| WK_TEST | EXPIRED & LOCKED |
| WKPROXY | EXPIRED & LOCKED |
| WKSYS | EXPIRED & LOCKED |
| WMSYS | EXPIRED & LOCKED |
| XDB | EXPIRED & LOCKED |

If any default database server user account other the ones left open is required for any reason, a database administrator (DBA) need simply unlock and activate that account with a new, meaningful password.

**Enterprise Manager Accounts**

The preceding list of accounts depends on whether you choose to install Enterprise Manager. If so, SYSMAN and DBSNMP are open as well, unless you configure Enterprise Manager for Central Administration: then the SYSMAN account (if present) will be locked as well.

If you do not install Enterprise Manager, then only SYS and SYSTEM are open. Database Configuration Assistant locks and expires all other accounts (including SYSMAN and DBSNMP).

3. **CHANGE DEFAULT USER PASSWORDS.**

   The most trivial method by which Oracle Database can be compromised is a default database server user account which still has a default password associated with it *even after installation*.

   a. **Change default passwords of administrative users.**

      Oracle Database 10g installation enables you to use the same or different passwords for the SYS, SYSTEM, SYSMAN and DBSNMP administrative accounts. Use different passwords for each: in any Oracle environment (production or test), assign strong, meaningful, and distinct passwords to these administrative accounts. If Database Configuration Assistant is used, it requires you to enter passwords for the SYS and SYSTEM accounts, disallowing the use of the defaults `CHANGE_ON_INSTALL` and `MANAGER`.

Similarly, for production environments, do not use default passwords for any administrative accounts, including SYSMAN and DBSNMP.

At the end of database creation, Database Configuration Assistant displays a page requiring you to enter and confirm new passwords for the SYS and SYSTEM user accounts.

b. **Change default passwords of all users.**

In Oracle Database, `SCOTT` no longer installs with default password `TIGER`, but instead is locked and expired, as is DBSNMP. Each of the other accounts install with a default password that is exactly the same as that user account (for example, user `MDSYS` installs with password `MDSYS`).

If any of the default user accounts that were locked and expired upon installation need to be activated, assign a new meaningful password to each such user account.

Even though Oracle does not explicitly mandate changing the default password for user `SCOTT`, Oracle nevertheless recommends that this user account also be locked in a production environment.

c. **Enforce password management.**

Oracle recommends that basic password management rules (such as password length, history, complexity, and so forth) as provided by the database be applied to all user passwords and that all users be required to change their passwords periodically.

Oracle also recommends, if possible, utilizing Oracle Advanced Security (an option to the Enterprise Edition of Oracle Database) with network authentication services (such as Kerberos), token cards, smart cards or X.509 certificates. These services enable strong authentication of users to provide better protection against unauthorized access to Oracle Database.

4. **ENABLE DATA DICTIONARY PROTECTION.**

Oracle recommends that customers implement data dictionary protection to prevent users having the `ANY` system privileges from using such privileges on the data dictionary.

To enable dictionary protection, set the following configuration parameter to FALSE, in the init<sid>.ora control file:

```
O7_DICTIONARY_ACCESSIBILITY = FALSE
```

By doing so, only those authorized users making DBA-privileged (for example `CONNECT / AS SYSDBA`) connections can use the `ANY` system privilege on the data dictionary. If this parameter is not set to the value recommended earlier, any user with a `DROP ANY TABLE` (for example) system privilege will be able to maliciously drop parts of the data dictionary.

However, if a user requires view access to the data dictionary, it is permissible to grant that user the `SELECT ANY DICTIONARY` system privilege.

**Notes:**

- Regarding `O7_DICTIONARY_ACCESSIBILITY`, note that in Oracle Database, the default is `FALSE`; whereas in Oracle8i, this parameter is set to `TRUE` by default and must specifically be changed to `FALSE` to enable this security feature.
- Regarding the SELECT ANY DICTIONARY privilege: this privilege is not included in the GRANT ALL PRIVILEGES statement, but it can be granted through a role.

5. **PRACTICE PRINCIPLE OF LEAST PRIVILEGE.**

   a. **Grant necessary privileges only.**

   Do not provide database users more privileges than are necessary. In other words, *principle of least privilege* is that a user be given only those privileges that are actually required to efficiently and succinctly perform his or her job.

   To implement least privilege, restrict: 1) the number of `SYSTEM` and `OBJECT` privileges granted to database users, and 2) the number of people who are allowed to make `SYS`-privileged connections to the database as much as possible. For example, there is generally no need to grant `CREATE ANY TABLE` to any non DBA-privileged user.

   b. **Revoke unnecessary privileges from `PUBLIC`.**

Revoke all unnecessary privileges and roles from the database server user group PUBLIC. PUBLIC acts as a default role granted to every user in an Oracle database. Any database user can exercise privileges that are granted to PUBLIC. Such privileges include EXECUTE on various PL/SQL packages that may permit a minimally privileged user to access and execute packages that he may not directly be permitted to access. The more powerful packages that may potentially be misused are listed in the following table:

| Package | Description |
|---|---|
| UTL_SMTP[1] | This package permits arbitrary mail messages to be sent from one arbitrary user to another arbitrary user. Granting this package to PUBLIC may permit unauthorized exchange of mail messages. |
| UTL_TCP[1] | This package permits outgoing network connections to be established by the database server to any receiving (or waiting) network service. Thus, arbitrary data may be sent between the database server and any waiting network service. |
| UTL_HTTP[1] | This package allows the database server to request and retrieve data using HTTP. Granting this package to PUBLIC may permit using HTML forms to send data to a malicious Web site. |
| UTL_FILE[1] | If configured improperly, this package allows text level access to any file on the host operating system. Even when properly configured, this package may allow unauthorized access to sensitive operating system files, such as trace files, because it does not distinguish between its calling applications. The result can be that one application accessing UTL_FILE may write arbitrary data into the same location that is written to by another application. |
| DBMS_RANDOM | This package can be used to encrypt stored data. Generally, most users should not have the privilege to encrypt data since encrypted data may be non-recoverable if the keys are not securely generated, stored, and managed. |

[1] These packages should be revoked from PUBLIC and made executable for an application only when absolutely necessary.

These packages are extremely useful to some applications that need them. They require proper configuration and usage for safe and secure operation, and may not be suitable for most applications.

c. **Grant users roles only if they need all of the role's privileges.**

Roles (groups of privileges) are useful for quickly and easily granting permissions to users. If your application users do not need all the privileges encompassed by an existing role, then create your own roles containing only the appropriate privileges for your requirements. Similary, ensure that roles contain only the privileges that reflect job responsibility.

For example, grant users the CREATE SESSION privilege to authorize them to log in to the database, rather than granting them the CONNECT role, which has many additional privileges. Unless users require all the extra privileges contained in the CONNECT role (or any other role), assign them individually only the minimum set of individual privileges truly needed. Alternatively, create your own roles and assign only needed privileges.

For example, it is imperative to strictly limit the privileges of SCOTT. Drop the CREATE DBLINK privilege for SCOTT. Then drop the entire role for the user, since privileges acquired by means of a role cannot be dropped individually. Recreate your own role with only the privileges needed, and grant that new role to that user. Similarly, for even better security, drop the CREATE DBLINK privilege from all users who do not require it.

d. **Restrict permissions on run-time facilities.**

Do not assign "all permissions" to any database server run-time facility such as the Oracle Java Virtual Machine (OJVM). Grant specific permissions to the explicit document root file paths for such facilities that may execute files and packages outside the database server.

Here is an example of a vulnerable run-time call:

```
call dbms_java.grant_permission('SCOTT',
'SYS:java.io.FilePermission','<<ALL FILES>>','read');
```

Here is an example of a better (more secure) run-time call:

```
call dbms_java.grant_permission('SCOTT',
'SYS:java.io.FilePermission','<<actual directory path>>','read');
```

6. **ENFORCE ACCESS CONTROLS EFFECTIVELY.**

**Authenticate clients properly.**

By default, Oracle allows operating-system-authenticated logins only over secure connections, which precludes using Oracle Net and a shared server configuration. This default restriction prevents a remote user from impersonating another operating system user over a network connection.

Setting the initialization parameter `REMOTE_OS_AUTHENT` to `TRUE` forces the RDBMS to accept the client operating system user name received over a nonsecure connection and use it for account access. Since clients, in general, such as PCs, are not trusted to perform operating system authentication properly, it is very poor security practice to turn on this feature.

The default setting, `REMOTE_OS_AUTHENT = FALSE`, creates a more secure configuration that enforces proper, server-based authentication of clients connecting to an Oracle database.

You should not alter the default setting of the `REMOTE_OS_AUTHENT` initialization parameter, which is `FALSE`.

Setting this parameter to FALSE does not mean that users cannot connect remotely. It simply means that the database will not trust that the client has already authenticated, and will therefore apply its standard authentication processes.

7. **RESTRICT OPERATING SYSTEM ACCESS.**

   **Limit the number of operating system users.**

   **Limit the privileges of the operating system accounts** (administrative, root-privileged or DBA) on the Oracle Database host (physical machine) to the least privileges needed for the user's tasks.

   Oracle also recommends:

   - Restricting the ability to modify the default file and directory permissions for the Oracle Database home (installation) directory or its contents. Even privileged operating system users and the Oracle owner should not modify these permissions, unless instructed otherwise by Oracle Corporation.
   - Restricting symbolic links. Ensure that when providing a path or file to the database, neither the file nor any part of the path is modifiable by an untrusted user. The file and all components of the path should be owned by the DBA or some trusted account, such as root.

     This recommendation applies to all types of files: data files, log files, trace files, external tables, bfiles, and so on.

8. **RESTRICT NETWORK ACCESS.**

   a. **Use a firewall.**

      Keep the database server behind a firewall. Oracle Database's network infrastructure, Oracle Net (formerly known as Net8 and SQL*Net), offers support for a variety of firewalls from various vendors. Supported proxy-enabled firewalls include Network Associates' Gauntlet and Axent's Raptor. Supported packet-filtered firewalls include Cisco's PIX Firewall and supported stateful inspection firewalls (more sophisticated packet-filtered firewalls) include CheckPoint's Firewall-1.

b. **Never poke a hole through a firewall.**

If Oracle Database is behind a firewall, do not, under any circumstances, poke a hole through the firewall; for example, do not leave open Oracle Listener's 1521 port to make a connection to the Internet or vice versa.

Doing so will introduce a number of significant security vulnerabilities including more port openings through the firewall, multi-threaded operating system server issues and revelation of crucial information on database(s) behind the firewall. Furthermore, an Oracle Listener running without an established password may be probed for critical details about the database(s) on which it is listening such as trace and logging information, banner information and database descriptors and service names.

Such a plethora of information and the availability of an ill-configured firewall will provide an attacker ample opportunity to launch malicious attacks on the target database(s).

c. **Protect the Oracle Listener.**

Because the listener acts as the database's gateway to the network, it is important to limit the consequences of malicious interference:

* Restrict the privileges of the listener, so that it cannot read or write files in the database or the Oracle server address space.

  This restriction prevents external procedure agents spawned by the listener (or procedures executed by such an agent) from inheriting the ability to do such reads or writes. The owner of this separate listener process should not be the owner that installed Oracle or executes the Oracle instance (such as ORACLE, the default owner).

  Sample configuration:

```
EXTPROC_LISTENER=
  (DESCRIPTION=
    (ADDRESS=
      (PROTOCOL=ipc)(KEY=extproc)))
SID_LIST_EXTPROC_LISTENER=
  (SID_LIST=
    (SID_DESC=
      (SID_NAME=plsextproc)
      (ORACLE_HOME=/u1/app/oracle/9.0)
      (PROGRAM=extproc)))
```

* Secure administration by these three steps:

  i. Prevent on-line administration by requiring the administrator to have write privileges on the `LISTENER.ORA` file and the listener's password:

  Add or alter this line in the LISTENER.ORA file

  ```
  ADMIN_RESTRICTIONS_LISTENER=ON
  ```

  Then RELOAD the configuration.

  ii. Use SSL when administering the listener, by making the TCPS protocol the first entry in the address list:

  ```
  LISTENER=
    (DESCRIPTION=
      (ADDRESS_LIST=
        (ADDRESS=
          (PROTOCOL=tcps)
          (HOST = ed-pdsun1.us.oracle.com)
          (PORT = 8281)))
  ```

  (To administer the listener remotely, you need to define the listener in the client computer's LISTENER.ORA file. For example, to access listener USER281 remotely., using the following configuration:)

  ```
  user281 =
    (DESCRIPTION =
      (ADDRESS =
        (PROTOCOL = tcps)
        (HOST = ed-pdsun1.us.oracle.com)
        (PORT = 8281))
      )
    )
  ```

  iii. Always establish a meaningful, well-formed password for the Oracle Listener to prevent remote configuration of the Oracle Listener. Password protect the listener:

  ```
  LSNRCTL> CHANGE_PASSWORD
  Old password: lsnrc80
  New password: lsnrc90
  Reenter new password: lsnrc90
  ```

```
LSNRCTL> SET PASSWORD
Password:
The command completed successfully
LSNRCTL> SAVE_CONFIG
The command completed successfully
```

* Actually remove the external procedure configuration from the listener.ora file if you do not intend to use such procedures.
* Monitor listener activity.

d. **Be sure of who is accessing your systems.**

Authenticating client computers over the Internet is problematic. Do user authentication instead, which avoids client system issues that include falsified IP addresses, hacked operating systems or applications, and falsified or stolen client system identities. The following steps improve client computer security:

* Configure the connection to use SSL. Using SSL (Secure Sockets Layer) communication makes eavesdropping unfruitful and enables the use of certificates for user and server authentication.
* Set up certificate authentication for clients and servers such that:

  i. The organization is identified by unit and certificate issuer and the user is identified by distinguished name and certificate issuer.

  ii. Applications test for expired certificates.

  iii. Certificate revocation lists are audited.

e. **Check network IP addresses.**

Utilize the Oracle Net "valid node checking" security feature to allow or deny access to Oracle server processes from network clients with specified IP addresses. To use this feature, set the following `protocol.ora` (Oracle Net configuration file) parameters:

```
tcp.validnode_checking = YES

tcp.excluded_nodes = {list of IP addresses}

tcp.invited_nodes = {list of IP addresses}
```

The first parameter turns on the feature whereas the latter two parameters respectively deny or allow specific client IP addresses from making connections to the Oracle Listener (and thereby preventing potential Denial of Service attacks).

f. **Encrypt network traffic.**

   If possible, utilize Oracle Advanced Security to encrypt network traffic between clients, databases, and application servers. (Note that Oracle Advanced Security is available only with the Enterprise Edition of the Oracle database. It installs in Typical Installation mode and can be configured, after licensing, with the Oracle Net Manager tool or by manually setting six sqlnet.ora parameters to enable network encryption. )

g. **Harden the operating system.**

   Harden the host operating system by disabling all unnecessary operating system services. Both UNIX and Windows platforms provide a variety of operating system services, most of which are not necessary for most deployments. Such services include FTP, TFTP, TELNET, and so forth. Be sure to close both the UDP and TCP ports for each service that is being disabled. Disabling one type of port and not the other does not make the operating system more secure.

9. **APPLY ALL SECURITY PATCHES AND WORKAROUNDS.**

   Always apply all relevant and current security patches for both the operating system on which Oracle Database resides and Oracle Database itself, and for all installed Oracle Database options and components thereof.

   Periodically check the security site on Oracle Technology Network for details on security alerts released by Oracle Corporation.

   `http://otn.oracle.com/deploy/security/alerts.htm`

   Also check Oracle Worldwide Support Service's site, Metalink, for details on available and upcoming security-related patches.

   `http://metalink.oracle.com`

In summary, consider all paths the data travels and assess the threats that impinge on each path and node. Then take steps to lessen or eliminate both those threats and the consequences of a successful breach of security. Also monitor and audit to detect either increased threat levels or successful penetration.

10. **CONTACT ORACLE SECURITY PRODUCTS.**

If you believe that you have found a security vulnerability in Oracle Database, submit an iTAR to Oracle Worldwide Support Services using Metalink, or e-mail a complete description of the problem, including product version and platform, together with any exploit scripts and examples to the following address:

`secalert_us@oracle.com`

# 8 Database Auditing: Security Considerations

**Auditing** is the monitoring and recording of selected user database actions. It can be based on individual actions, such as the type of SQL statement executed, or on combinations of factors that can include user name, application, time, and so on. Security policies can trigger auditing when specified elements in an Oracle database are accessed or altered, including the contents within a specified object.

An overview of database auditing appears in Chapter 6.

Chapter 11 provides detailed information and guidelines on configuring auditing parameters and administering auditing actions and results.

The present chapter describes the different types and focuses of auditing and the resulting audit trails and records.

Auditing is normally used to:

- Enable future accountability for current actions taken in a particular schema, table, or row, or affecting specific content
- Deter users (or others) from inappropriate actions, based on that accountability
- Investigate suspicious activity. For example, if some user is deleting data from tables, the security administrator might decide to audit all connections to the database and all successful and unsuccessful deletions of rows from all tables in the database.
- Notify an auditor that an unauthorized user is manipulating or deleting data and that the user has more privileges than expected, which can lead to reassessing user authorizations.
- Monitor and gather data about specific database activities. For example, the database administrator can gather statistics about which tables are being updated, how many logical I/Os are performed, or how many concurrent users connect at peak times.

- Detect problems with an authorization or access control implementation. For example, you can create audit policies that you expect will never generate an audit record because the data is protected in other ways. However, if these policies do generate audit records, you will know the other security controls are not properly implemented.

This chapter describes the types of auditing available in Oracle systems, in the following sections:

- Auditing Types and Records
- Statement Auditing
- Privilege Auditing
- Schema Object Auditing
- Fine-Grained Auditing
- Focusing Statement, Privilege, and Schema Object Auditing
- Auditing in a Multitier Environment

**See Also:**

- Chapter 11, "Configuring and Administering Auditing"
- *Oracle Database Administrator's Guide*

## Auditing Types and Records

Oracle allows audit options to be focused or broad, enabling you to audit:

- Successful statement executions, unsuccessful statement executions, or both
- Statement executions once in each user session or once every time the statement is executed
- Activities of all users or of a specific user

Table 8–1 describes the different Oracle auditing mechanisms. Each entry in the first column is a link to a more extensive discussion of that particular method.

*Table 8–1 Auditing Types and Descriptions*

| Type of Auditing (link to discussion) | Meaning/Description |
|---|---|
| Statement Auditing | Audits SQL statements by type of statement, not by the specific schema objects on which they operate. Typically broad, statement auditing audits the use of several types of related actions for each option. For example, `AUDIT TABLE` tracks several DDL statements regardless of the table on which they are issued. You can also set statement auditing to audit selected users or every user in the database. |
| Privilege Auditing | Audits the use of powerful system privileges that enable corresponding actions, such as `AUDIT CREATE TABLE`. Privilege auditing is more focused than statement auditing, which audits only a particular type of action. You can set privilege auditing to audit a selected user or every user in the database. |
| Schema Object Auditing | Audits specific statements on a particular schema object, such as `AUDIT SELECT ON employees`. Schema object auditing is very focused, auditing only a single specified type of statement (such as SELECT) on a specified schema object. Schema object auditing always applies to all users of the database. |
| Fine-Grained Auditing | Audits, at the most granular level, data access and actions based on content, using any boolean measure, such as *value* > 1,000,000. Enables auditing based on access to or changes in a column. |

The following subsections explain the records and timing of the different audit trails:

- Audit Records and the Audit Trails
- When Are Audit Records Created?

## Audit Records and the Audit Trails

Audit records include such information as the operation that was audited, the user performing the operation, and the date and time of the operation. Audit records can be stored in either a data dictionary table, called the **database audit trail**, or in operating system files, called an operating system audit trail.

> **See Also:** The complete contents of these audit trails is described in Chapter 11, "Configuring and Administering Auditing", in the section entitled What Information is Contained in the Audit Trail?

The two general types of auditing are standard auditing, which is based on privileges, schemas, objects, and statements, and fine-grained auditing. Standard audit records can be written either to DBA_AUDIT_TRAIL (the sys.aud$ table) or to the operating system. Fine-grained audit records are written to DBA_FGA_AUDIT_TRAIL (the sys.fga_log$ table) and the `DBA_COMMON_AUDIT_TRAIL` view, which combines standard and fine-grained audit log records.

The following subsections describe these trails and records:

- Database Audit Trail (DBA_AUDIT_TRAIL)
- Operating System Audit Trail
- Operating System Audit Records
- Records Always in the Operating System Audit Trail

## Database Audit Trail (DBA_AUDIT_TRAIL)

The database audit trail is a single table named `SYS.AUD$` in the `SYS` schema of each Oracle database's data dictionary. Several predefined views are provided to help you use the information in this table, such as `DBA_AUDIT_TRAIL`.

Audit trail records can contain different types of information, depending on the events audited and the auditing options set. The partial list in the following section shows columns that always appear in the audit trail; if the data they represent is available, that data populates the corresponding column. (For certain columns, this list shows the column name displayed in the audit record, here inside parentheses.)

***Table 8–2 Columns Shown in the Database Audit Trail (DBA_AUDIT_TRAIL)***

| Column Description/Name | Also Appears in the Operating System Audit Trail? |
|---|---|
| Operating system login user name (`CLIENT USER`) | **Yes.** |
| Database user name (`DATABASE USER`) | No. |
| Session identifier | **Yes.** |
| Terminal identifier | **Yes.** |
| Name of the schema object accessed | **Yes.** |
| Operation performed or attempted (`ACTION`) | **Yes.** |
| Completion code of the operation | **Yes.** |
| Date and time stamp in UTC (Coordinated Universal Time) format | No. |

*Table 8–2 (Cont.) Columns Shown in the Database Audit Trail (DBA_AUDIT_TRAIL)*

| Column Description/Name | Also Appears in the Operating System Audit Trail? |
|---|---|
| System privileges used (PRIVILEGE) | **Yes.** |
| Proxy Session's auditid | No. |
| Global User unique id | No. |
| Distinguished name | **Yes.** |
| Instance number | No. |
| Process number | No. |
| TransactionId | No. |
| SCN (system change number) for the SQL statement | No. |
| SQL text that triggered the auditing (SQLTEXT) | No. |
| Bind values used for the SQL statement, if any (SQLBIND) | No. |

**Notes:**

- The "Process number" column is always NULL in Oracle Database 10*g*.
- SQLBIND and SQLTEXT are not populated unless you specified AUDIT_TRAIL=DB_EXTENDED in the database initialization file, init.ora, since CLOBs are comparatively expensive to populate.

If the database destination for audit records becomes full or unavailable and therefore unable to accept new records, an audited action cannot complete. Instead, it causes an error message and is not done. In some cases, an operating system log allows such an action to complete.

### Operating System Audit Trail

Oracle allows audit trail records to be directed to an operating system audit trail if the operating system makes such an audit trail available to Oracle. If not, audit records are written to a file outside the database. The target directory varies by platform: on the Solaris platform, it is $ORACLE_HOME/rdbms/audit, but for other platforms you must check the platform documentation to learn the correct target directory. In Windows, the information is accessed through Event Viewer.

> **See Also:** Your operating system specific Oracle documentation, to see if this feature has been implemented on your operating system

An operating system audit trail or file system can become full and therefore unable to accept new records, including audit records directed to the operating system. In this circumstance, Oracle still allows certain actions that are *always* audited to continue, even though the audit record cannot be stored because the operating system destination is full. Using a database audit trail prevents audited actions from completing if their audit records cannot be stored.

System administrators configuring operating system auditing should ensure that the operating system audit trail or the file system does not fill completely. Most operating systems provide administrators with sufficient information and warning to enable them to ensure this does not occur.

Note, however, that configuring auditing to use the database audit trail removes this potential loss of audit information. The Oracle server prevents audited events from occurring if the audit trail is unable to accept the database audit record for the statement.

### Operating System Audit Records

The operating system audit trail is encoded, but it is decoded in data dictionary files and error messages.

- **Action code** describes the operation performed or attempted. The `AUDIT_ACTIONS` data dictionary table contains a list of these codes and their descriptions.
- **Privileges used** describes any system privileges used to perform the operation. The `SYSTEM_PRIVILEGE_MAP` table lists all of these codes and their descriptions.
- **Completion code** describes the result of the attempted operation. Successful operations return a value of zero, and unsuccessful operations return the Oracle error code describing why the operation was unsuccessful.

**See Also:**

- Table 8–2, " Columns Shown in the Database Audit Trail (DBA_AUDIT_TRAIL)", which also indicates the columns that appear in the operating system audit trail.
- *Oracle Database Administrator's Guide* for instructions for creating and using predefined views
- *Oracle Database Error Messages* for a list of completion codes

### Records Always in the Operating System Audit Trail

Some database-related actions are always recorded into the operating system audit trail regardless of whether database auditing is enabled. The fact that these records are always created is sometimes referred to as mandatory auditing:

- At instance startup, an audit record is generated that details the operating system user starting the instance, the user's terminal identifier, the date and time stamp. This information is recorded into the operating system audit trail because the database audit trail is not available until after startup has successfully completed.
- At instance shutdown, an audit record is generated that details the operating system user shutting down the instance, the user's terminal identifier, the date and time stamp.
- During connections made with administrator privileges, an audit record is generated that details the operating system user connecting to Oracle with administrator privileges. This record provides accountability regarding users connected with administrator privileges.

On operating systems that do not make an audit trail accessible to Oracle, these audit trail records are placed in an Oracle audit trail file in the same directory as background process trace files, and in a similar format.

**See Also:** Your operating system specific Oracle documentation for more information about the operating system audit trail

## When Are Audit Records Created?

Standard auditing for the entire database is either enabled or disabled by the security administrator. If it is disabled, no audit records are created.

---

**Note:** Fine-grained auditing uses audit policies applied to individual objects. Therefore, standard audit settings that are on or off for the entire database do not affect fine-grained auditing.

---

If database auditing is enabled by the security administrator, then individual audit options become effective. These audit options can be set by any authorized database user for database objects he owns.

When auditing is enabled in the database and an action set to be audited occurs, an audit record is generated during the execute phase of statement execution.

SQL statements inside PL/SQL program units are individually audited, as necessary, when the program unit is executed.

The generation and insertion of an audit trail record is independent of a user's transaction being committed. That is, even if a user's transaction is rolled back, the audit trail record remains committed.

Statement and privilege audit options in effect at the time a database user connects to the database remain in effect for the duration of the session. Setting or changing statement or privilege audit options in a session does not cause effects in that session. The modified statement or privilege audit options take effect only when the current session is ended and a new session is created.

In contrast, changes to schema object audit options become effective for current sessions immediately.

---

**Note:** Operations by the `SYS` user and by users connected through `SYSDBA` or `SYSOPER` can be fully audited with the `AUDIT_SYS_OPERATIONS` initialization parameter. Every successful SQL statement from `SYS` is audited. This specialized form of auditing audits all actions performed by every user with the SYSDBA privilege and writes only to an operating system location. It is not dependent on the standard auditing parameter AUDIT_TRAIL=. Sending these records to a location separate from the usual database audit trail in the `SYS` schema provides for greater auditing security.

---

**See Also:**

- *Oracle Database Administrator's Guide* for instructions on enabling and disabling auditing
- "SQL, PL/SQL, and Java" in *Oracle Database Concepts* for information about the different phases of SQL statement processing and shared SQL

## Statement Auditing

Statement auditing is the selective auditing of related groups of statements regarding a particular type of database structure or schema object, but not a specifically named structure or schema object. These statements fall into two categories:

- DDL statements. As an example, `AUDIT TABLE` audits all `CREATE` and `DROP TABLE` statements.
- DML statements. As an example, `AUDIT SELECT TABLE` audits all `SELECT ... FROM TABLE/VIEW` statements, regardless of the table or view.

Statement auditing can be broad or focused, auditing the activities of all database users or of only a select list.

## Privilege Auditing

Privilege auditing audits statements that use a system privilege, such as `SELECT ANY TABLE.` For example, when AUDIT SELECT ANY TABLE is in force, all statements issued by users with the SELECT ANY TABLE privilege are audited.

You can audit the use of any system privilege. Like statement auditing, privilege auditing can audit the activities of all database users or of only a specified list.

If similar statement and privilege audit options are both set, only a single audit record is generated. For example, if the statement clause `TABLE` and the system privilege `CREATE TABLE` are both audited, only a single audit record is generated each time a table is created.

Thus privilege auditing does not occur if the action is already permitted by the existing owner and schema object privileges. Privilege auditing triggered only if they are insufficient, that is, only if what makes the action possible is a system privilege.

Privilege auditing is more focused than statement auditing because each privilege auditing option audits only specific types of statements, not a related list of statements. For example, the statement auditing clause TABLE audits CREATE TABLE, ALTER TABLE, and DROP TABLE statements. However, the privilege auditing option CREATE TABLE audits only CREATE TABLE statements, because only the CREATE TABLE statement requires the CREATE TABLE privilege.

## Schema Object Auditing

Schema object auditing can audit all SELECT and DML statements permitted by schema object privileges, such as SELECT or DELETE statements on a given table. The GRANT and REVOKE statements that control those privileges are also audited.

You can audit statements that reference tables, views, sequences, **standalone** stored procedures or functions, and packages, but not individual procedures within packages. Further discussion appears in the next section, entitled Schema Object Audit Options for Views, Procedures, and Other Elements.

Statements that reference clusters, database links, indexes, or synonyms are not audited directly. However, you can audit access to these schema objects indirectly, by auditing the operations that affect the base table.

Schema object audit options are always set for all users of the database. These options cannot be set for a specific list of users. You can set default schema object audit options for all auditable schema objects.

**See Also:** *Oracle Database SQL Reference* for information about auditable schema objects

### Schema Object Audit Options for Views, Procedures, and Other Elements

The definitions for views and procedures (including stored functions, packages, and triggers) reference underlying schema objects. Because of this dependency, some unique characteristics apply to auditing views and procedures, such as the likelihood of generating multiple audit records.

Views and procedures are subject to the enabled audit options on the base schema objects, including the default audit options. These options apply to the resulting SQL statements as well.

Consider the following series of SQL statements:

```
AUDIT SELECT ON employees;
```

```
CREATE VIEW employees_departments AS
  SELECT employee_id, last_name, department_id
    FROM employees, departments
    WHERE employees.department_id = departments.department_id;

AUDIT SELECT ON employees_departments;

SELECT * FROM employees_departments;
```

As a result of the query on `employees_departments`, two audit records are generated: one for the query on the `employees_departments` view and one for the query on the base table `employees` (indirectly through the `employees_departments` view). The query on the base table `departments` does not generate an audit record because the `SELECT` audit option for this table is not enabled. All audit records pertain to the user that queried the `employees_departments` view.

The audit options for a view or procedure are determined when the view or procedure is first used and placed in the shared pool. These audit options remain set until the view or procedure is flushed from, and subsequently replaced in, the shared pool. Auditing a schema object invalidates that schema object in the cache and causes it to be reloaded. Any changes to the audit options of base schema objects are not observed by views and procedures in the shared pool.

In the given example, if the "`AUDIT SELECT ON employees;`" statement is omitted, then using the `employees_departments` view will not generate an audit record for the `employees` table.

Table 8–3, " Auditing Actions Newly Enabled by Oracle Database 10g" lists auditing actions that were not available before Oracle Database.

***Table 8–3  Auditing Actions Newly Enabled by Oracle Database 10g***

| Object or Element | Newly Auditable Actions |
|---|---|
| Materialized Views | AUDIT, DELETE, INSERT, SELECT, UPDATE, AND FLASHBACK |
| Tables & views | REFERENCES, UNDER, ON COMMIT REFRESH, QUERY REWRITE, DEBUG, and FLASHBACK |
| Library | AUDIT, EXECUTE, and DEBUG |
| Java source | AUDIT, EXECUTE, and DEBUG |
| Operator | AUDIT and EXECUTE |
| Index type | AUDIT and EXECUTE |

*Table 8–3 (Cont.) Auditing Actions Newly Enabled by Oracle Database 10g*

| Object or Element | Newly Auditable Actions |
|---|---|
| Directory | WRITE |
| Queue | AUDIT, ENQUEUE, and DEQUEUE |

## Focusing Statement, Privilege, and Schema Object Auditing

Oracle lets you focus statement, privilege, and schema object auditing in three areas, as discussed in the following subsections:

- Auditing Statement Executions: Successful, Unsuccessful, or Both
- Number of Audit Records from Multiple Executions of a Statement
- Audit By User, for specific users or for all users in the database (statement and privilege auditing only)

### Auditing Statement Executions: Successful, Unsuccessful, or Both

For statement, privilege, and schema object auditing, Oracle allows the selective auditing of successful executions of statements, unsuccessful attempts to execute statements, or both. Therefore, you can monitor actions even if the audited statements do not complete successfully. Monitoring unsuccessful SQL can expose users who are snooping or acting maliciously, though of course most unsuccessful SQL is neither.

Auditing an unsuccessful statement execution provides a report only if a valid SQL statement is issued but fails because it lacks proper authorization or references a nonexistent schema object. Statements that failed to execute because they simply were not valid cannot be audited.

For example, an enabled privilege auditing option set to audit unsuccessful statement executions audits statements that use the target system privilege but have failed for other reasons. One example is when a `CREATE TABLE` auditing condition is set, but some `CREATE TABLE` statements fail due to lack of quota for the specified tablespace.

When your audit statement includes the `WHENEVER SUCCESSFUL` clause, you will be auditing only successful executions of the audited statement.

When your audit statement includes the `WHENEVER NOT SUCCESSFUL` clause, you will be auditing only unsuccessful executions of the audited statement.

When your audit statement includes neither of the preceding two clauses, you will be auditing both successful and unsuccessful executions of the audited statement.

## Number of Audit Records from Multiple Executions of a Statement

If an audited statement is issued multiple times in a single user session, your audit trail can have one or more related records. The controlling clause `BY ACCESS` causes each execution of an auditable operation within a cursor to generate a separate audit record. If you use the `BY SESSION` clause instead, your audit trail will contain a single audit record for each session, for each user and schema object. Only one audit record results, no matter how often the statement occurs in that session.

However, several audit options can be set only `BY ACCESS`:

- All statement audit options that audit DDL statements
- All privilege audit options that audit DDL statements

For all other audit options, `BY SESSION` is used by default.

This section provides detailed examples of using each clause, in the following subsections:

- BY SESSION
- BY ACCESS

**See Also:** *Oracle Database SQL Reference*

### BY SESSION

For any type of audit (schema object, statement, or privilege), `BY SESSION` inserts only one audit record in the audit trail, for each user and schema object, during a session that includes an audited action.

A **session** is the time between when a user connects to and disconnects from an Oracle database.

**BY SESSION Example 1** Assume the following:

- The `SELECT TABLE` statement auditing option is set `BY SESSION`.
- `JWARD` connects to the database and issues five `SELECT` statements against the table named `departments` and then disconnects from the database.

- SWILLIAMS connects to the database and issues three SELECT statements against the table employees and then disconnects from the database.

In this case, the audit trail contains two audit records for the eight SELECT statements— one for each session that issued a SELECT statement.

**BY SESSION Example 2** Alternatively, assume the following:

- The SELECT TABLE statement auditing option is set BY SESSION.
- JWARD connects to the database and issues five SELECT statements against the table named departments, and three SELECT statements against the table employees, and then disconnects from the database.

In this case, the audit trail contains two records—one for each schema object against which the user issued a SELECT statement in a session.

**Note:** If you use the BY SESSION clause when directing audit records to the operating system audit trail, Oracle generates and stores an audit record each time an access is made. Therefore, in this auditing configuration, BY SESSION is equivalent to BY ACCESS.

### BY ACCESS

Setting audit BY ACCESS inserts one audit record into the audit trail for each execution of an auditable operation within a cursor. Events that cause cursors to be reused include the following:

- An application, such as Oracle Forms, holding a cursor open for reuse
- Subsequent execution of a cursor using new bind variables
- Statements executed within PL/SQL loops where the PL/SQL engine optimizes the statements to reuse a single cursor

Note that auditing is **not** affected by whether a cursor is shared. Each user creates her or his own audit trail records on first execution of the cursor.

For example, assume that:

- The SELECT TABLE statement auditing option is set BY ACCESS.
- JWARD connects to the database and issues five SELECT statements against the table named departments and then disconnects from the database.

- SWILLIAMS connects to the database and issues three SELECT statements against the table departments and then disconnects from the database.

The single audit trail contains eight records for the eight SELECT statements.

### Audit By User

Statement and privilege audit options can audit statements issued by any user or statements issued by a specific list of users. By focusing on specific users, you can minimize the number of audit records generated.

**Audit By User Example** To audit statements by the users SCOTT and BLAKE that query or update a table or view, issue the following statements:

```
AUDIT SELECT TABLE, UPDATE TABLE
     BY scott, blake;
```

**See Also:** *Oracle Database SQL Reference* for more information about auditing by user

## Auditing in a Multitier Environment

In a multitier environment, Oracle can preserve the identity of a client through all tiers. Thus, you can audit actions taken on behalf of the client by a mid-tier application. To do so, use the BY **proxy** clause in your AUDIT statement.

This clause allows you a few options. You can:

- Audit SQL statements issued by the specific proxy on its own behalf
- Audit statements executed on behalf of a specified user or users
- Audit all statements executed on behalf of any user

The middle tier can also set the user's client identity in a database session, enabling audit of end-user actions through the mid-tier application. The end-user's client identity then shows up in the audit trail.

**See Also:**

- *Oracle Database Application Developer's Guide - Fundamentals*
- *Oracle Call Interface Programmer's Guide*
- *PL/SQL User's Guide and Reference*

## Fine-Grained Auditing

Fine-Grained Auditing (FGA) enables you to monitor data access based on content. A built-in audit mechanism in the database prevents users from bypassing the audit.

While Oracle triggers can potentially monitor DML actions such as `INSERT`, `UPDATE`, and `DELETE`, monitoring on `SELECT` can be costly. In some cases, a trigger may audit too much; in others, its effectiveness or completeness may be uncertain. Triggers also do not enable users to define their own alert action in response to a triggered audit, beyond simply inserting an audit record into the audit trail.

Fine-Grained Auditing provides an extensible interface for creating policies to audit SELECT and DML statements on tables and views. The `DBMS_FGA` package administers these value-based audit policies. Using `DBMS_FGA`, the security administrator creates an audit policy on the target object. If any rows returned from a query match the audit condition, then an audit event entry is inserted into the fine-grained audit trail. This entry includes all the information reported in the regular audit trail: see the Audit Records and the Audit Trails section on page 8-3. Only one row of audit information is inserted into the audit trail for every FGA policy that evaluates to TRUE. The extensibility framework in FGA also enables administrators optionally to define an appropriate **audit event handler** to process the event, for example by sending an alert page to the administrator.

The administrator uses the DBMS_FGA.ADD_POLICY interface to define each FGA policy for a table or view, identifying any combination of SELECT, UPDATE, DELETE, or INSERT statements.

FGA policies associated with a table or view may also specify *relevant columns*, so that any specified statement type affecting a relevant column is audited. If no *relevant column* is specified, auditing applies to all columns. That is, auditing occurs whenever any specified statement type affects any column, independent of whether any rows are returned or not.

The relevant-column capability enables you to hone in on particularly important types of data to audit. Examples include privacy-relevant columns, such as those containing social security numbers, salaries, patient diagnoses, and so on. You could combine the fine-grained audit records to surface queries that had addressed both name and social security number, a potential violation of privacy security laws.

One added benefit is that the audit records being created are more clearly relevant, because they relate to specific data of interest or concern. Another benefit is that fewer total audit records need be generated, because each is now more specific and useful than what could be tracked in earlier releases.

**See Also:**

- *Chapter 11, "Configuring and Administering Auditing"*
- *Oracle Database Application Developer's Guide - Fundamentals*
- The DBMS_FGA chapter in *PL/SQL Packages and Types Reference*

# Part III

# Security Implementation, Configuration, and Administration

Part III presents the details of configuring and administering Oracle Database security features.

This part contains the following chapter:

# 9 Administering Authentication

Authentication is the process of verifying the identity of a user, device, or other entity in a computer system, often as a prerequisite to granting access to resources in a system.

## User Authentication Methods

Oracle provides several means for users to be authenticated before they are allowed to create a database session, as discussed in the following sections:

| Topics: You can define users who are ... | Links to Topics |
|---|---|
| identified and authenticated by the database, which is called **database authentication**. | Database Authentication |
| authenticated by the operating system or network service, which is called **external authentication**. | External Authentication |
| authenticated **globally** by SSL (Secure Sockets Layer), called **global users**, whose database access is through **global roles**, authorized by an enterprise directory. | Global Authentication and Authorization |
| allowed to connect through a middle-tier server that authenticates the user, assumes that identity, and can enable specific roles for the user. This combination of actions and abilities is called **proxy authentication** and authorization. | Proxy Authentication and Authorization |

### Database Authentication

If you choose database authentication for a user, administration of the user account including authentication of that user is performed entirely by Oracle. To have Oracle authenticate a user, specify a password for the user when you create or alter the user. Users can change their password at any time. Passwords are stored in an encrypted format. While usernames can be multibyte, each password must be made up of single-byte characters, even if your database uses a multibyte character set.

---

**Note:** Oracle Corporation recommends that you encode user names and passwords in ASCII or EBCDIC characters only, depending on your platform. Doing so will maintain compatibility for supporting future changes to your database character set.

Basing user names or passwords on characters that expand in size when migrated to a new target character set can cause login difficulties. Authentication can fail after such a migration because the encrypted user names and passwords in the data dictionary are not updated during a migration to a new database character set.

For example, assuming the current database character set is WE8MSWIN1252 and the target database character set is UTF8, the user name `scött` (o with an umlaut) will change from 5 bytes to 6 bytes in UTF8. the user `scött` will no longer be able to log in.

If user names and passwords are not based on ASCII or EBCDIC characters, then if a migration to a new character set occurs, the affected user names and passwords will need to be reset.

---

To enhance security when using database authentication, Oracle recommends the use of password management, including account locking, password aging and expiration, password history, and password complexity verification.

**See Also:** "Password Management Policy" on page 23-12

## Creating a User Who is Authenticated by the Database

The following statement creates a user who is identified and authenticated by Oracle. User `scott` must specify the password `tiger` whenever connecting to Oracle.

```
CREATE USER scott IDENTIFIED BY tiger;
```

**See Also:** *Oracle Database SQL Reference* for more information about valid passwords, and how to specify the `IDENTIFIED BY` clause in the `CREATE USER` and `ALTER USER` statements

### Advantages of Database Authentication

Following are advantages of database authentication:

- User accounts and all authentication are controlled by the database. There is no reliance on anything outside of the database.
- Oracle provides strong password management features to enhance security when using database authentication.
- It is easier to administer when there are small user communities.

## External Authentication

When you choose external authentication for a user, the user account is maintained by Oracle, but password administration and user authentication is performed by an external service. This external service can be the operating system or a network service, such as Oracle Net.

With external authentication, your database relies on the underlying operating system or network authentication service to restrict access to database accounts. A database password is not used for this type of login. If your operating system or network service permits, you can have it authenticate users. If you do so, set the initialization parameter `OS_AUTHENT_PREFIX`, and use this prefix in Oracle user names. The `OS_AUTHENT_PREFIX` parameter defines a prefix that Oracle adds to the beginning of every user's operating system account name. Oracle compares the prefixed user name with the Oracle user names in the database when a user attempts to connect.

For example, assume that `OS_AUTHENT_PREFIX` is set as follows:

```
OS_AUTHENT_PREFIX=OPS$
```

**Note:** The text of the `OS_AUTHENT_PREFIX` initialization parameter is case sensitive on some operating systems. See your operating system specific Oracle documentation for more information about this initialization parameter.

If a user with an operating system account named `tsmith` is to connect to an Oracle database and be authenticated by the operating system, Oracle checks that there is a corresponding database user `OPS$tsmith` and, if so, allows the user to connect. All references to a user authenticated by the operating system must include the prefix, as seen in `OPS$tsmith`.

The default value of this parameter is OPS$ for backward compatibility with previous versions of Oracle. However, you might prefer to set the prefix value to some other string or a null string (an empty set of double quotes: ""). Using a null string eliminates the addition of any prefix to operating system account names, so that Oracle user names exactly match operating system user names.

After you set OS_AUTHENT_PREFIX, it should remain the same for the life of a database. If you change the prefix, any database user name that includes the old prefix cannot be used to establish a connection, unless you alter the user name to have it use password authentication.

### Creating a User Who is Authenticated Externally

The following statement creates a user who is identified by Oracle and authenticated by the operating system or a network service. This example assumes that OS_AUTHENT_PREFIX = "".

```
CREATE USER scott IDENTIFIED EXTERNALLY;
```

Using CREATE USER ... IDENTIFIED EXTERNALLY, you create database accounts that must be authenticated by the operating system or network service. Oracle will then rely on this external login authentication when it provides that specific operating system user with access to a specific database user's resources.

> **See Also:** *Oracle Advanced Security Administrator's Guide* for more information about external authentication

### Operating System Authentication

By default, Oracle allows operating-system-authenticated logins only over secure connections, which precludes using Oracle Net and a shared server configuration. This default restriction prevents a remote user from impersonating another operating system user over a network connection.

Setting REMOTE_OS_AUTHENT to TRUE in the database's initialization parameter file forces the RDBMS to accept the client operating system user name received over a nonsecure connection and use it for account access. Since clients, in general, such as PCs, are not trusted to perform operating system authentication properly, it is very poor security practice to turn on this feature.

The default setting, REMOTE_OS_AUTHENT = FALSE, creates a more secure configuration that enforces proper, server-based authentication of clients connecting to an Oracle database.

Any change to this parameter takes effect the next time you start the instance and mount the database. Generally, user authentication through the host operating system offers faster and more convenient connection to Oracle without specifying a separate database user name or password. Also, user entries correspond in the database and operating system audit trails.

### Network Authentication

Network authentication is performed using Oracle Advanced Security, which can be configured to use a third party service such as Kerberos. If you are using Oracle Advanced Security as your only external authentication service, the setting of the parameter `REMOTE_OS_AUTHENT` is irrelevant, since Oracle Advanced Security only allows secure connections.

### Advantages of External Authentication

Following are advantages of external authentication:

- More choices of authentication mechanism are available, such as smart cards, fingerprints, Kerberos, or the operating system.
- Many network authentication services, such as Kerberos and DCE, support single sign-on, enabling users to have fewer passwords to remember.
- If you are already using some external mechanism for authentication, such as one of those listed earlier, there may be less administrative overhead to use that mechanism with the database as well.

## Global Authentication and Authorization

Oracle Advanced Security enables you to centralize management of user-related information, including authorizations, in an LDAP-based directory service. Users can be identified in the database as global users, meaning that they are authenticated by SSL and that the management of these users is done outside of the database by the centralized directory service. Global roles are defined in a database and are known only to that database, but authorizations for such roles is done by the directory service.

**Note:** You can also have users authenticated by SSL, whose authorizations are not managed in a directory; that is, they have local database roles only. See the *Oracle Advanced Security Administrator's Guide* for details.

This centralized management enables the creation of **enterprise users** and **enterprise roles**. Enterprise users are defined and managed in the directory. They have unique identities across the enterprise, and can be assigned enterprise roles that determine their access privileges across multiple databases. An enterprise role consists of one or more global roles, and might be thought of as a container for global roles.

### Creating a User Who is Authorized by a Directory Service

You have a couple of options as to how you specify users who are authorized by a directory service.

**Creating a Global User** The following statement illustrates the creation of a global user, who is authenticated by SSL and authorized by the enterprise directory service:

```
CREATE USER scott
   IDENTIFIED GLOBALLY AS 'CN=scott,OU=division1,O=oracle,C=US';
```

The string provided in the AS clause provides an identifier (**distinguished name,** or **DN**) meaningful to the enterprise directory.

In this case, `scott` is truly a global user. But, the disadvantage here is that user `scott` must then be created in every database that he must access, plus the directory.

**Creating a Schema-Independent User** Creating schema-independent users allows multiple enterprise users to access a shared schema in the database. A schema-independent user is:

- Authenticated by SSL or passwords
- *Not* created in the database with a `CREATE USER` statement of any type
- A user whose privileges are managed in a directory
- A user who connects to a shared schema

The process of creating a schema-independent user is as follows:

1. Create a shared schema in the database as follows.

   ```
   CREATE USER appschema INDENTIFIED GLOBALLY AS '';
   ```

2. In the directory, you now create multiple enterprise users, and a mapping object.

The mapping object tells the database how you want to map users' DNs to the shared schema. You can either do a full DN mapping (one directory entry for each unique DN), or you can map, for example, every user containing the following DN components to the `appschema`:

```
OU=division,O=Oracle,C=US
```

See the *Oracle Internet Directory Administrator's Guide* for an explanation of these mappings.

Most users do not need their own schemas, and implementing schema-independent users divorces users from databases. You create multiple users who share the same schema in a database, and as enterprise users, they can access shared schemas in other databases as well.

### Advantages of Global Authentication and Global Authorization

Some of the advantages of global user authentication and authorization are the following:

- Provides strong authentication using SSL or Windows NT native authentication
- Enables centralized management of users and privileges across the enterprise
- Is easy to administer—for every user you do not have to create a schema in every database in the enterprise
- Facilitates single sign-on—users only need to sign on once to access multiple databases and services. Further, users using passwords can have a single password to access databases accepting password authenticated enterprise users.
- Because it provides password based access, previously defined password authenticated database users can be migrated to the directory (using the User Migration Utility) to be centrally administered. This makes global authentication and authorization available for prior Oracle release clients that are still supported.
- `CURRENT_USER` database links connect as a global user. A local user can connect as a global user in the context of a stored procedure—without storing the global user's password in a link definition.

**See Also:** The following books contain additional information about global authentication and authorization, and enterprise users and roles:

- *Oracle Advanced Security Administrator's Guide*
- *Oracle Internet Directory Administrator's Guide*

## Proxy Authentication and Authorization

It is possible to design a middle-tier server to proxy clients in a secure fashion.

Oracle provides three forms of proxy authentication:

- The middle-tier server authenticates itself with the database server and a client, in this case an application user or another application, authenticates itself with the middle-tier server. Client identities can be maintained all the way through to the database.
- The client, in this case a database user, is not authenticated by the middle-tier server. The clients identity and database password are passed through the middle-tier server to the database server for authentication.
- The client, in this case a global user, is authenticated by the middle-tier server, and passes one of the following through the middle tier for retrieving the client's user name.
  - Distinguished name (DN)
  - Certificate

In all cases, the middle-tier server must be authorized to act on behalf of the client by the administrator.

To authorize a middle-tier server to proxy a client use the `GRANT CONNECT THROUGH` clause of the `ALTER USER` statement. You can also specify roles that the middle tier is permitted to activate when connecting as the client.

Operations done on behalf of a client by a middle-tier server can be audited.

The `PROXY_USERS` data dictionary view can be queried to see which users are currently authorized to connect through a middle tier.

Use the `REVOKE CONNECT THROUGH` clause of `ALTER USER` to disallow a proxy connection.

**See Also:**

- *Oracle Call Interface Programmer's Guide* and *Oracle Database Application Developer's Guide - Fundamentals* for details about designing a middle-tier server to proxy users
- *Oracle Database SQL Reference* for a description and syntax of the proxy clause for `ALTER USER`
- "Auditing in a Multi-Tier Environment" on page 26-13 for details of auditing operations done on behalf of a user by a middle tier

### Authorizing a Middle Tier to Proxy and Authenticate a User

The following statement authorizes the middle-tier server `appserve` to connect as user `bill`. It uses the `WITH ROLE` clause to specify that `appserve` activate all roles associated with `bill`, except `payroll`.

```
ALTER USER bill
    GRANT CONNECT THROUGH appserve
    WITH ROLE ALL EXCEPT payroll;
```

To revoke the middle-tier server's (`appserve`) authorization to connect as user `bill`, the following statement is used:

```
ALTER USER bill REVOKE CONNECT THROUGH appserve;
```

### Authorizing a Middle Tier to Proxy a User Authenticated by Other Means

Use the `AUTHENTICATED USING` clause of the `ALTER USER ... GRANT CONNECT THROUGH` statement to authorize a user to be proxied, but not authenticated, by a middle tier. Currently, `PASSWORD` is the only means supported.

The following statement illustrates this form of authentication:

```
ALTER USER mary
    GRANT CONNECT THROUGH midtier
    AUTHENTICATED USING PASSWORD;
```

In the preceding statement, middle-tier server `midtier` is authorized to connect as `mary`, and `midtier` must also pass `mary`'s password to the database server for authorization.

# 10

# Administering User Privileges, Roles, and Profiles

Many tasks, with many interwoven considerations, are involved in administering user privileges, roles, and profiles. These necessary operations and principles are discussed in the following sections:

| Topic Category | Links to Topics |
|---|---|
| Managing Privileges, Roles, and Profiles | ■ Managing Oracle Users<br>■ Viewing Information About Database Users and Profiles<br>■ Managing Resources with Profiles<br>■ Understanding User Privileges and Roles<br>■ Managing User Roles |
| Granting, Revoking, and Viewing Privileges and Roles | ■ Granting User Privileges and Roles<br>■ Revoking User Privileges and Roles<br>■ Granting to and Revoking from the User Group PUBLIC<br>■ When Do Grants and Revokes Take Effect?<br>■ Granting Roles Using the Operating System or Network<br>■ Viewing Privilege and Role Information |

## Managing Oracle Users

Each Oracle database has a list of valid database users. To access a database, a user must run a database application and connect to the database instance using a valid user name defined in the database. This section explains how to manage users for a database, and contains the following topics:

- Creating Users
- Altering Users
- Dropping Users

> **See Also:** *Oracle Database SQL Reference* for more information about SQL statements used for managing users

## Creating Users

You create a database user with the `CREATE USER` statement.To create a user, you must have the `CREATE USER` system privilege. Because it is a powerful privilege, a DBA or security administrator is normally the only user who has the `CREATE USER` system privilege.

The following example creates a user and specifies that user's password, default tablespace, temporary tablespace where temporary segments are created, tablespace quotas, and profile.

```
CREATE USER jward
    IDENTIFIED BY AZ7BC2
    DEFAULT TABLESPACE data_ts
    QUOTA 100M ON test_ts
    QUOTA 500K ON data_ts
    TEMPORARY TABLESPACE temp_ts
    PROFILE clerk;
GRANT create session TO jward;
```

A newly created user cannot connect to the database until granted the `CREATE SESSION` system privilege.

> **Note:** As administrator, you should create your own roles and assign only those privileges that are needed. For example, it is unwise to grant CONNECT if all that is needed is CREATE SESSION, since CONNECT includes several additional privileges: see Table 10–1 on page 10-18. Creating its own roles gives an organization detailed control of the privileges it assigns, and protects it in case Oracle were to change or remove roles that it defines.

This section refers to the preceding example as it discusses the following aspects of creating a user:

- Specifying a Name
- Setting a User's Authentication
- Assigning a Default Tablespace
- Assigning Tablespace Quotas
- Assigning a Temporary Tablespace
- Specifying a Profile
- Setting Default Roles

**See Also:** "Granting System Privileges and Roles" on page 25-11

### Specifying a Name

Within each database a user name must be unique with respect to other user names and roles. A user and role cannot have the same name. Furthermore, each user has an associated schema. Within a schema, each schema object must have a unique name.

### Setting a User's Authentication

In the previous `CREATE USER` statement, the new user is to be authenticated using the database. In this case, the connecting user must supply the correct password to the database to connect successfully.

Selecting and specifying the method of user authentication is discussed in "User Authentication Methods" on page 9-1.

### Assigning a Default Tablespace

Each user should have a default tablespace. When a user creates a schema object and specifies no tablespace to contain it, Oracle stores the object in the user's default tablespace.

The default setting for every user's default tablespace is the `SYSTEM` tablespace. If a user does not create objects, and has no privileges to do so, this default setting is fine. However, if a user is likely to create any type of object, you should specifically assign the user a default tablespace. Using a tablespace other than `SYSTEM` reduces contention between data dictionary objects and user objects for the same datafiles. In general, it is not advisable for user data to be stored in the `SYSTEM` tablespace.

You can create a permanent default tablespace other than `SYSTEM` at the time of database creation, to be used as the database default for permanent objects. By separating the user data from the system data, you reduce the likelihood of problems with the SYSTEM tablespace, which can in some circumstances cause the entire database to become non-functional. This default permanent tablespace is not used by system users, that is, SYS, SYSTEM, and OUTLN, whose default permanent tablespace remains SYSTEM. A tablespace designated as the default permanent tablespace cannot be dropped; to accomplish this goal, another tablespace must first be designated as the default permanent tablespace. It is possible to ALTER the default permanent tablespace to another tablespace, affecting all users/objects created after the ALTER DDL commits.

You can also set a user's default tablespace during user creation, and change it later with the ALTER USER statement. Changing the user's default tablespace affects only objects created after the setting is changed.

When you specify the user's default tablespace, also specify a quota on that tablespace.

In the previous CREATE USER statement, jward's default tablespace is data_ts, and his quota on that tablespace is 500K.

### Assigning Tablespace Quotas

You can assign each user a tablespace quota for any tablespace (except a temporary tablespace). Assigning a quota does two things:

- Users with privileges to create certain types of objects can create those objects in the specified tablespace.
- Oracle limits the amount of space that can be allocated for storage of a user's objects within the specified tablespace to the amount of the quota.

By default, a user has no quota on any tablespace in the database. If the user has the privilege to create a schema object, you must assign a quota to allow the user to create objects. Minimally, assign users a quota for the default tablespace, and additional quotas for other tablespaces in which they can create objects.

You can assign a user either individual quotas for a specific amount of disk space in each tablespace or an unlimited amount of disk space in all tablespaces. Specific quotas prevent a user's objects from consuming too much space in the database.

You can assign a user's tablespace quotas when you create the user, or add or change quotas later. If a new quota is less than the old one, then the following conditions hold true:

- If a user has already exceeded a new tablespace quota, the user's objects in the tablespace cannot be allocated more space until the combined space of these objects falls below the new quota.
- If a user has not exceeded a new tablespace quota, or if the space used by the user's objects in the tablespace falls under a new tablespace quota, the user's objects can be allocated space up to the new quota.

**Revoking Users' Ability to Create Objects in a Tablespace** You can revoke a user's ability to create objects in a tablespace by changing the user's current quota to zero. After a quota of zero is assigned, the user's objects in the tablespace remain, but new objects cannot be created and existing objects cannot be allocated any new space.

**UNLIMITED TABLESPACE System Privilege** To permit a user to use an unlimited amount of any tablespace in the database, grant the user the `UNLIMITED TABLESPACE` system privilege. This overrides all explicit tablespace quotas for the user. If you later revoke the privilege, explicit quotas again take effect. You can grant this privilege only to users, not to roles.

Before granting the `UNLIMITED TABLESPACE` system privilege, consider the consequences of doing so.

**Advantage:**

- You can grant a user unlimited access to all tablespaces of a database with one statement.

**Disadvantages:**

- The privilege overrides all explicit tablespace quotas for the user.
- You cannot selectively revoke tablespace access from a user with the `UNLIMITED TABLESPACE` privilege. You can grant access selectively only after revoking the privilege.

### Assigning a Temporary Tablespace

Each user also should be assigned a temporary tablespace. When a user executes a SQL statement that requires a temporary segment, Oracle stores the segment in the user's temporary tablespace. These temporary segments are created by the system when doing sorts or joins and are owned by SYS, which has resource privileges in all tablespaces.

In the previous `CREATE USER` statement, `jward`'s temporary tablespace is `temp_ts`, a tablespace created explicitly to contain only temporary segments. Such a tablespace is created using the `CREATE TEMPORARY TABLESPACE` statement.

If a user's temporary tablespace is not explicitly set, the user is assigned the default temporary tablespace that was specified at database creation, or by an `ALTER DATABASE` statement at a later time. If there is no default temporary tablespace explicitly assigned, the default is the `SYSTEM` tablespace or another permanent default established by the system administrator. It is not advisable for user data to be stored in the `SYSTEM` tablespace. Also, assigning a tablespace to be used specifically as a temporary tablespace eliminates file contention among temporary segments and other types of segments.

---

**Note:** If your `SYSTEM` tablespace is locally managed, then users must be assigned a specific default (locally managed) temporary tablespace. They may not be allowed to default to using the `SYSTEM` tablespace because temporary objects cannot be placed in permanent locally managed tablespaces.

---

You can set a user's temporary tablespace at user creation, and change it later using the `ALTER USER` statement. Do not set a quota for temporary tablespaces. You can also establish tablespace groups instead of assigning individual temporary tablespaces.

**See Also:** These sections in *Oracle Database Administrator's Guide*:, Chapter 8, Managing Tablespaces:

- "Temporary Tablespaces"
- "Multiple Temporary Tablespaces: Using Tablespace Groups"

### Specifying a Profile

You also specify a profile when you create a user. A profile is a set of limits on database resources and password access to the database. If no profile is specified, the user is assigned a default profile.

**See Also:** "Managing Resources with Profiles" on page 10-13

### Setting Default Roles

You cannot set a user's default roles in the `CREATE USER` statement. When you first create a user, the user's default role setting is `ALL`, which causes all roles subsequently granted to the user to be default roles. Use the ALTER USER statement to change the user's default roles.

## Altering Users

Users can change their own passwords. However, to change any other option of a user's security domain, you must have the ALTER USER system privilege. Security administrators are normally the only users that have this system privilege, as it allows a modification of *any* user's security domain. This privilege includes the ability to set tablespace quotas for a user on any tablespace in the database, even if the user performing the modification does not have a quota for a specified tablespace.

You can alter a user's security settings with the ALTER USER statement. Changing a user's security settings affects the user's future sessions, not current sessions.

The following statement alters the security settings for user avyrros:

```
ALTER USER avyrros
    IDENTIFIED EXTERNALLY
    DEFAULT TABLESPACE data_ts
    TEMPORARY TABLESPACE temp_ts
    QUOTA 100M ON data_ts
    QUOTA 0 ON test_ts
    PROFILE clerk;
```

The ALTER USER statement here changes avyrros's security settings as follows:

- Authentication is changed to use avyrros's operating system account.
- avyrros's default and temporary tablespaces are explicitly set.
- avyrros is given a 100M quota for the data_ts tablespace.
- avyrros's quota on the test_ts is revoked.
- avyrros is assigned the clerk profile.

### Changing a User's Authentication Mechanism

Most non-DBA users can still change their own passwords with the ALTER USER statement, as follows:

```
ALTER USER andy
   IDENTIFIED BY swordfish;
```

No special privileges (other than those to connect to the database) are required for a user to change passwords. Users should be encouraged to change their passwords frequently.

Users must have the `ALTER USER` privilege to switch between methods of authentication. Usually, only an administrator has this privilege.

**See Also:** "User Authentication Methods" on page 9-1 for information about the authentication methods that are available for Oracle users

### Changing a User's Default Roles

A default role is one that is automatically enabled for a user when the user creates a session. You can assign a user zero or more default roles.

**See Also:** "Managing User Roles" on page 10-20

## Dropping Users

When a user is dropped, the user and associated schema are removed from the data dictionary and all schema objects contained in the user's schema, if any, are immediately dropped.

**Notes:**

- If a user's schema and associated objects must remain but the user must be denied access to the database, revoke the `CREATE SESSION` privilege from the user.
- Do not attempt to drop the SYS or SYSTEM user. Doing so will corrupt your database.

A user that is currently connected to a database cannot be dropped. To drop a connected user, you must first terminate the user's sessions using the SQL statement `ALTER SYSTEM` with the `KILL SESSION` clause.

You can drop a user from a database using the `DROP USER` statement. To drop a user and all the user's schema objects (if any), you must have the `DROP USER` system privilege. Because the `DROP USER` system privilege is so powerful, a security administrator is typically the only type of user that has this privilege.

If the user's schema contains any schema objects, use the `CASCADE` option to drop the user and all associated objects and foreign keys that depend on the tables of the user successfully. If you do not specify CASCADE and the user's schema contains objects, an error message is returned and the user is not dropped. Before dropping a user whose schema contains objects, thoroughly investigate which objects the user's schema contains and the implications of dropping them. Pay attention to any unknown cascading effects. For example, if you intend to drop a user who owns a table, check whether any views or procedures depend on that particular table.

The following statement drops user jones and all associated objects and foreign keys that depend on the tables owned by jones.

```
DROP USER jones CASCADE;
```

> **See Also:** "Terminating Sessions" in *Oracle Database Administrator's Guide* for more information about terminating sessions

## Viewing Information About Database Users and Profiles

The wide variety of options for viewing such information is discussed in the following subsections:

- User and Profile Information in Data Dictionary Views
- Listing All Users and Associated Information
- Listing All Tablespace Quotas
- Listing All Profiles and Assigned Limits
- Viewing Memory Use for Each User Session

### User and Profile Information in Data Dictionary Views

The following data dictionary views contain information about database users and profiles:

| View | Description |
|---|---|
| DBA_USERS | DBA view describes all users of the database. |
| ALL_USERS | ALL view lists users visible to the current user, but does not describe them. |
| USER_USERS | USER view describes only the current user. |

| View | Description |
|---|---|
| DBA_TS_QUOTAS<br>USER_TS_QUOTAS | Describes tablespace quotas for users. |
| USER_PASSWORD_LIMITS | Describes the password profile parameters that are assigned to the user. |
| USER_RESOURCE_LIMITS | Displays the resource limits for the current user. |
| DBA_PROFILES | Displays all profiles and their limits. |
| RESOURCE_COST | Lists the cost for each resource. |
| V$SESSION | Lists session information for each current session. Includes user name. |
| V$SESSTAT | Lists user session statistics. |
| V$STATNAME | Displays decoded statistic names for the statistics shown in the V$SESSTAT view. |
| PROXY_USERS | Describes users who can assume the identity of other users. |

The following sections present some example of using these views, and assume a database in which the following statements have been executed:

```
CREATE PROFILE clerk LIMIT
    SESSIONS_PER_USER 1
    IDLE_TIME 30
    CONNECT_TIME 600;

CREATE USER jfee
    IDENTIFIED BY wildcat
    DEFAULT TABLESPACE users
    TEMPORARY TABLESPACE temp_ts
    QUOTA 500K ON users
    PROFILE clerk;

CREATE USER dcranney
    IDENTIFIED BY bedrock
    DEFAULT TABLESPACE users
    TEMPORARY TABLESPACE temp_ts
    QUOTA unlimited ON users;

CREATE USER userscott
     IDENTIFIED BY scott1;
```

**See Also:** *Oracle Database SQL Reference* for complete descriptions of the preceding data dictionary and dynamic performance views

## Listing All Users and Associated Information

The following query lists users and their associated information as defined in the database:

```
SELECT USERNAME, PROFILE, ACCOUNT_STATUS FROM DBA_USERS;

USERNAME        PROFILE         ACCOUNT_STATUS
--------------- --------------- ---------------
SYS             DEFAULT         OPEN
SYSTEM          DEFAULT         OPEN
USERSCOTT       DEFAULT         OPEN
JFEE            CLERK           OPEN
DCRANNEY        DEFAULT         OPEN
```

All passwords are encrypted to preserve security. If a user queries the `PASSWORD` column, that user is not able to determine another user's password.

## Listing All Tablespace Quotas

The following query lists all tablespace quotas specifically assigned to each user:

```
SELECT * FROM DBA_TS_QUOTAS;

TABLESPACE    USERNAME    BYTES     MAX_BYTES    BLOCKS    MAX_BLOCKS
----------    ---------  --------   ----------   -------   ----------
USERS         JFEE              0       512000         0          250
USERS         DCRANNEY          0           -1         0           -1
```

When specific quotas are assigned, the exact number is indicated in the `MAX_BYTES` column. Note that this number is always a multiple of the database block size, so if you specify a tablespace quota that is not a multiple of the database block size, it is rounded up accordingly. Unlimited quotas are indicated by "-1".

## Listing All Profiles and Assigned Limits

The following query lists all profiles in the database and associated settings for each limit in each profile:

```
SELECT * FROM DBA_PROFILES
   ORDER BY PROFILE;
```

```
PROFILE             RESOURCE_NAME              RESOURCE   LIMIT
-----------------   ---------------            ---------- --------------
CLERK               COMPOSITE_LIMIT            KERNEL     DEFAULT
CLERK               FAILED_LOGIN_ATTEMPTS      PASSWORD   DEFAULT
CLERK               PASSWORD_LIFE_TIME         PASSWORD   DEFAULT
CLERK               PASSWORD_REUSE_TIME        PASSWORD   DEFAULT
CLERK               PASSWORD_REUSE_MAX         PASSWORD   DEFAULT
CLERK               PASSWORD_VERIFY_FUNCTION   PASSWORD   DEFAULT
CLERK               PASSWORD_LOCK_TIME         PASSWORD   DEFAULT
CLERK               PASSWORD_GRACE_TIME        PASSWORD   DEFAULT
CLERK               PRIVATE_SGA                KERNEL     DEFAULT
CLERK               CONNECT_TIME               KERNEL     600
CLERK               IDLE_TIME                  KERNEL     30
CLERK               LOGICAL_READS_PER_CALL     KERNEL     DEFAULT
CLERK               LOGICAL_READS_PER_SESSION  KERNEL     DEFAULT
CLERK               CPU_PER_CALL               KERNEL     DEFAULT
CLERK               CPU_PER_SESSION            KERNEL     DEFAULT
CLERK               SESSIONS_PER_USER          KERNEL     1
DEFAULT             COMPOSITE_LIMIT            KERNEL     UNLIMITED
DEFAULT             PRIVATE_SGA                KERNEL     UNLIMITED
DEFAULT             SESSIONS_PER_USER          KERNEL     UNLIMITED
DEFAULT             CPU_PER_CALL               KERNEL     UNLIMITED
DEFAULT             LOGICAL_READS_PER_CALL     KERNEL     UNLIMITED
DEFAULT             CONNECT_TIME               KERNEL     UNLIMITED
DEFAULT             IDLE_TIME                  KERNEL     UNLIMITED
DEFAULT             LOGICAL_READS_PER_SESSION  KERNEL     UNLIMITED
DEFAULT             CPU_PER_SESSION            KERNEL     UNLIMITED
DEFAULT             FAILED_LOGIN_ATTEMPTS      PASSWORD   UNLIMITED
DEFAULT             PASSWORD_LIFE_TIME         PASSWORD   UNLIMITED
DEFAULT             PASSWORD_REUSE_MAX         PASSWORD   UNLIMITED
DEFAULT             PASSWORD_LOCK_TIME         PASSWORD   UNLIMITED
DEFAULT             PASSWORD_GRACE_TIME        PASSWORD   UNLIMITED
DEFAULT             PASSWORD_VERIFY_FUNCTION   PASSWORD   UNLIMITED
DEFAULT             PASSWORD_REUSE_TIME        PASSWORD   UNLIMITED
32 rows selected.
```

## Viewing Memory Use for Each User Session

The following query lists all current sessions, showing the Oracle user and current UGA (user global area) memory use for each session:

```
SELECT USERNAME, VALUE || 'bytes' "Current UGA memory"
   FROM V$SESSION sess, V$SESSTAT stat, V$STATNAME name
```

```
WHERE sess.SID = stat.SID
   AND stat.STATISTIC# = name.STATISTIC#
   AND name.NAME = 'session uga memory';

USERNAME                       Current UGA memory
------------------------------ ---------------------------------------------
                               18636bytes
                               17464bytes
                               19180bytes
                               18364bytes
                               39384bytes
                               35292bytes
                               17696bytes
                               15868bytes
USERSCOTT                      42244bytes
SYS                            98196bytes
SYSTEM                         30648bytes

11 rows selected.
```

To see the maximum UGA memory ever allocated to each session since the instance started, replace `'session uga memory'` in the preceding query with `'session uga memory max'`.

## Managing Resources with Profiles

A profile is a named set of resource limits that restrict a user's database usage and instance resources, as well as password practices. For profiles to take effect, resource limits must be turned on for the database as a whole. You can assign a profile to each user, and a default profile to all others. Each user can have only one profile; creating a new one supersedes any earlier one.

However, in Oracle 10g, resource allocations and restrictions are primarily handled through the Database Resource Manager.

**See Also:**

- For resource allocation, see the *Database Resource Manager*, as described in the Oracle Database Administrator's Guide.
- For password policies, see Password Management Policy on page 7-12.

A profile can be created, assigned to users, altered, and dropped at any time (using CREATE USER or ALTER USER) by any authorized database user. Profiles can be assigned only to users and not to roles or other profiles. Such assignments do not affect current sessions. Profile resource limits are enforced only when you enable resource limitation for the associated database. Enabling such limitation can occur either before starting up the database (the `RESOURCE_LIMIT` initialization parameter) or while it is open (using an `ALTER SYSTEM` statement).

**See Also:**

- *Oracle Database SQL Reference* for more information about the SQL statements used for managing profiles, such as CREATE PROFILE, and for information on how to calculate composite limits.
- "Creating Users" on page 10-2
- "Altering Users" on page 10-7
- *Database Resource Manager* in the Oracle Database Administrator's Guide.

## Dropping Profiles

To drop a profile, you must have the `DROP PROFILE` system privilege. You can drop a profile (other than the default profile) using the SQL statement `DROP PROFILE`. To successfully drop a profile currently assigned to a user, use the `CASCADE` option.

The following statement drops the profile `clerk`, even though it is assigned to a user:

```
DROP PROFILE clerk CASCADE;
```

Any user currently assigned to a profile that is dropped is automatically assigned to the `DEFAULT` profile. The `DEFAULT` profile cannot be dropped. When a profile is dropped, the drop does not affect currently active sessions. Only sessions created after a profile is dropped abide by any modified profile assignments.

## Understanding User Privileges and Roles

A user **privilege** is a right to execute a particular type of SQL statement, or a right to access another user's object, execute a PL/SQL package, and so on. The types of privileges are defined by Oracle.

**Roles** are created by users (usually administrators) to group together privileges or other roles. They are a means of facilitating the granting of multiple privileges or roles to users.

This section describes Oracle user privileges, and contains the following topics:

- System Privileges
- Object Privileges
- User Roles

> **See Also:** *Oracle Database Concepts* for additional information about privileges and roles

### System Privileges

There are over 100 distinct system privileges. Each system privilege allows a user to perform a particular database operation or class of database operations.

> **Caution:** System privileges can be very powerful, and should be granted only when necessary to roles and trusted users of the database.

> **See Also:** *Oracle Database SQL Reference.* for the complete list of system privileges and their descriptions

#### Restricting System Privileges

Because system privileges are so powerful, Oracle recommends that you configure your database to prevent regular (non-DBA) users exercising `ANY` system privileges (such as `UPDATE ANY TABLE`) on the data dictionary. In order to secure the data dictionary, ensure that the `O7_DICTIONARY_ACCESSIBILITY` initialization parameter is set to `FALSE`, the default value. This feature is called the dictionary protection mechanism.

**Note:** The O7_DICTIONARY_ACCESSIBILITY initialization parameter controls restrictions on system privileges when you upgrade from Oracle database version 7 to Oracle8*i* and higher releases. If the parameter is set to TRUE, access to objects in the SYS schema is allowed (Oracle database version 7 behavior). If this parameter is set to FALSE, system privileges that allow access to objects in "any schema" do not allow access to objects in SYS schema. The default for O7_DICTIONARY_ACCESSIBILITY is FALSE.

When this parameter is not set to FALSE, the ANY privilege applies to the data dictionary, and a malicious user with ANY privilege could access or alter data dictionary tables.

See the *Oracle Database Reference* for more information on the O7_DICTIONARY_ACCESSIBILITY initialization parameter to understand its usage.

If you enable dictionary protection (O7_DICTIONARY_ACCESSIBILITY is FALSE), access to objects in the SYS schema (dictionary objects) is restricted to users with the SYS schema. These users are SYS and those who connect as SYSDBA. System privileges providing access to objects in other schemas do *not* give other users access to objects in the SYS schema. For example, the SELECT ANY TABLE privilege allows users to access views and tables in other schemas, but does not enable them to select dictionary objects (base tables of dynamic performance views, views, packages, and synonyms). These users can, however, be granted explicit object privileges to access objects in the SYS schema.

### Accessing Objects in the SYS Schema

Users with explicit object privileges or those who connect with administrative privileges (SYSDBA) can access objects in the SYS schema. Another means of allowing access to objects in the SYS schema is by granting users any of the following roles:

- SELECT_CATALOG_ROLE

  This role can be granted to users to allow SELECT privileges on data dictionary views.

- EXECUTE_CATALOG_ROLE

This role can be granted to users to allow EXECUTE privileges for packages and procedures in the data dictionary.

- DELETE_CATALOG_ROLE

  This role can be granted to users to allow them to delete records from the system audit table (AUD$).

Additionally, the following system privilege can be granted to users who require access to tables created in the SYS schema:

- SELECT ANY DICTIONARY

  This system privilege allows query access to any object in the SYS schema, including tables created in that schema. It must be granted individually to each user requiring the privilege. It is not included in GRANT ALL PRIVILEGES, nor can it be granted through a role.

**Caution:** You should grant these roles and the SELECT ANY DICTIONARY system privilege with extreme care, since the integrity of your system can be compromised by their misuse.

## Object Privileges

Each type of object has different privileges associated with it.

You can specify ALL [PRIVILEGES] to grant or revoke all available object privileges for an object. ALL is not a privilege; rather, it is a shortcut, or a way of granting or revoking all object privileges with one word in GRANT and REVOKE statements. Note that if all object privileges are granted using the ALL shortcut, individual privileges can still be revoked.

Likewise, all individually granted privileges can be revoked by specifying ALL. However, if you REVOKE ALL, and revoking causes integrity constraints to be deleted (because they depend on a REFERENCES privilege that you are revoking), you must include the CASCADE CONSTRAINTS option in the REVOKE statement.

**See Also:** *Oracle Database SQL Reference.* for the complete list of object privileges

## User Roles

A **role** groups several privileges and roles, so that they can be granted to and revoked from users simultaneously. A role must be enabled for a user before it can be used by the user.

Oracle provides some predefined roles to help in database administration. These roles, listed in Table 10–1, are automatically defined for Oracle databases when you run the standard scripts that are part of database creation. You can grant privileges and roles to, and revoke privileges and roles from, these predefined roles in the same way as you do with any role you define.

**Note:** Each installation should create its own roles and assign only those privileges that are needed. For example, it is unwise to grant CONNECT if all that is needed is CREATE SESSION, since CONNECT includes several additional privileges: see Table 10–1. Creating its own roles gives an organization detailed control of the privileges it assigns, and protects it in case Oracle were to change or remove roles that it defines.

***Table 10–1 Predefined Roles***

| Role Name | Created By (Script) | Description |
|---|---|---|
| `CONNECT` | `SQL.BSQ` | Includes the following system privileges: `ALTER SESSION`, `CREATE CLUSTER`, `CREATE DATABASE LINK`, `CREATE SEQUENCE`, `CREATE SESSION`, `CREATE SYNONYM`, `CREATE TABLE`, `CREATE VIEW` |
| `RESOURCE` | `SQL.BSQ` | Includes the following system privileges: `CREATE CLUSTER`, `CREATE INDEXTYPE`, `CREATE OPERATOR`, `CREATE PROCEDURE`, `CREATE SEQUENCE`, `CREATE TABLE`, `CREATE TRIGGER`, `CREATE TYPE` |
| `DBA` | `SQL.BSQ` | All system privileges `WITH ADMIN OPTION` |

**Note:** The previous three roles are provided to maintain compatibility with previous versions of Oracle and may not be created automatically in future versions of Oracle. Oracle Corporation recommends that you design your own roles for database security, rather than relying on these roles.

*Table 10–1 (Cont.) Predefined Roles*

| Role Name | Created By (Script) | Description |
|---|---|---|
| EXP_FULL_DATABASE | CATEXP.SQL | Provides the privileges required to perform full and incremental database exports. Includes: SELECT ANY TABLE, BACKUP ANY TABLE, EXECUTE ANY PROCEDURE, EXECUTE ANY TYPE, ADMINISTER RESOURCE MANAGER, and INSERT, DELETE, and UPDATE on the tables SYS.INCVID, SYS.INCFIL, and SYS.INCEXP. Also the following roles: EXECUTE_CATALOG_ROLE and SELECT_CATALOG_ROLE. |
| IMP_FULL_DATABASE | CATEXP.SQL | Provides the privileges required to perform full database imports. Includes an extensive list of system privileges (use view DBA_SYS_PRIVS to view privileges) and the following roles: EXECUTE_CATALOG_ROLE and SELECT_CATALOG_ROLE. |
| DELETE_CATALOG_ROLE | SQL.BSQ | Provides DELETE privilege on the system audit table (AUD$) |
| EXECUTE_CATALOG_ROLE | SQL.BSQ | Provides EXECUTE privilege on objects in the data dictionary. Also, HS_ADMIN_ROLE. |
| SELECT_CATALOG_ROLE | SQL.BSQ | Provides SELECT privilege on objects in the data dictionary. Also, HS_ADMIN_ROLE. |
| RECOVERY_CATALOG_OWNER | CATALOG.SQL | Provides privileges for owner of the recovery catalog. Includes: CREATE SESSION, ALTER SESSION, CREATE SYNONYM, CREATE VIEW, CREATE DATABASE LINK, CREATE TABLE, CREATE CLUSTER, CREATE SEQUENCE, CREATE TRIGGER, and CREATE PROCEDURE |
| HS_ADMIN_ROLE | CATHS.SQL | Used to protect access to the HS (Heterogeneous Services) data dictionary tables (grants SELECT) and packages (grants EXECUTE). It is granted to SELECT_CATALOG_ROLE and EXECUTE_CATALOG_ROLE such that users with generic data dictionary access also can access the HS data dictionary. |
| AQ_USER_ROLE | CATQUEUE.SQL | Obsoleted, but kept mainly for release 8.0 compatibility. Provides execute privilege on DBMS_AQ and DBMS_AQIN. |

***Table 10–1 (Cont.) Predefined Roles***

| Role Name | Created By (Script) | Description |
|---|---|---|
| AQ_ADMINISTRATOR_ROLE | CATQUEUE.SQL | Provides privileges to administer Advance Queuing. Includes ENQUEUE ANY QUEUE, DEQUEUE ANY QUEUE, and MANAGE ANY QUEUE, SELECT privileges on AQ tables and EXECUTE privileges on AQ packages. |

If you install other options or products, other predefined roles may be created.

## Managing User Roles

This section describes aspects of managing roles, and contains the following topics:

- Creating a Role
- Specifying the Type of Role Authorization
- Dropping Roles

### Creating a Role

You can create a role using the CREATE ROLE statement, but you must have the CREATE ROLE system privilege to do so. Typically, only security administrators have this system privilege.

**Note:** Immediately after creation, a role has no privileges associated with it. To associate privileges with a new role, you must grant privileges or other roles to the new role.

You must give each role you create a unique name among existing usernames and role names of the database. Roles are not contained in the schema of any user. In a database that uses a multibyte character set, Oracle recommends that each role name contain at least one single-byte character. If a role name contains only multibyte characters, the encrypted role name / password combination is considerably less secure.

The following statement creates the clerk role, which is authorized by the database using the password bicentennial:

```
CREATE ROLE clerk IDENTIFIED BY bicentennial;
```

The `IDENTIFIED BY` clause specifies how the user must be authorized before the role can be enabled for use by a specific user to which it has been granted. If this clause is not specified, or `NOT IDENTIFIED` is specified, then no authorization is required when the role is enabled. Roles can be specified to be authorized by:

- The database using a password
- An application using a specified package
- Externally by the operating system, network, or other external source
- Globally by an enterprise directory service

These authorizations are discussed in following sections.

Later, you can set or change the authorization method for a role using the `ALTER ROLE` statement. The following statement alters the `clerk` role to specify that the user must have been authorized by an external source before enabling the role:

```
ALTER ROLE clerk IDENTIFIED EXTERNALLY;
```

To alter the authorization method for a role, you must have the `ALTER ANY ROLE` system privilege or have been granted the role with the `ADMIN OPTION`.

**See Also:**

- *Oracle Database SQL Reference* for syntax, restrictions, and authorization information about the SQL statements used to manage roles and privileges
- *Oracle Advanced Security Administrator's Guide*

## Specifying the Type of Role Authorization

The methods of authorizing roles are presented in this section. A role must be enabled for you to use it.

**See Also:** "When Do Grants and Revokes Take Effect?" on page 10-35 for a discussion about enabling roles

### Role Authorization by the Database

The use of a role authorized by the database can be protected by an associated password. If you are granted a role protected by a password, you can enable or disable the role by supplying the proper password for the role in a `SET ROLE` statement. However, if the role is made a default role and enabled at connect time, the user is not required to enter a password.

The following statement creates a role `manager`. When it is enabled, the password `morework` must be supplied.

```
CREATE ROLE manager IDENTIFIED BY morework;
```

---

**Note:** In a database that uses a multibyte character set, passwords for roles must include only singlebyte characters. Multibyte characters are not accepted in passwords. See the *Oracle Database SQL Reference* for information about specifying valid passwords.

---

### Role Authorization by an Application

The `INDENTIFIED USING` *`package_name`* clause lets you create an application role, which is a role that can be enabled only by applications using an authorized package. Application developers do not need to secure a role by embedding passwords inside applications. Instead, they can create an application role and specify which PL/SQL package is authorized to enable the role.

The following example indicates that the role `admin_role` is an application role and the role can only be enabled by any module defined inside the PL/SQL package `hr.admin`.

```
CREATE ROLE admin_role IDENTIFIED USING hr.admin;
```

When enabling the user's default roles at login as specified in the user's profile, no checking is performed for application roles.

### Role Authorization by an External Source

The following statement creates a role named `accts_rec` and requires that the user be authorized by an external source before it can be enabled:

```
CREATE ROLE accts_rec IDENTIFIED EXTERNALLY;
```

**Role Authorization by the Operating System** Role authentication through the operating system is useful only when the operating system is able to dynamically link operating system privileges with applications. When a user starts an application, the operating system grants an operating system privilege to the user. The granted operating system privilege corresponds to the role associated with the application. At this point, the application can enable the application role. When the application is terminated, the previously granted operating system privilege is revoked from the user's operating system account.

If a role is authorized by the operating system, you must configure information for each user at the operating system level. This operation is operating system dependent.

If roles are granted by the operating system, you do not need to have the operating system authorize them also; this is redundant.

> **See Also:** "Granting Roles Using the Operating System or Network" on page 10-36 for more information about roles granted by the operating system

**Role Authorization and Network Clients** If users connect to the database over Oracle Net, by default their roles cannot be authenticated by the operating system. This includes connections through a shared server configuration, as this connection requires Oracle Net. This restriction is the default because a remote user could impersonate another operating system user over a network connection.

If you are not concerned with this security risk and want to use operating system role authentication for network clients, set the initialization parameter `REMOTE_OS_ROLES` in the database's initialization parameter file to `TRUE`. The change will take effect the next time you start the instance and mount the database. The parameter is `FALSE` by default.

### Role Authorization by an Enterprise Directory Service

A role can be defined as a global role, whereby a (global) user can only be authorized to use the role by an enterprise directory service. You define the global role locally in the database by granting privileges and roles to it, but you cannot grant the global role itself to any user or other role in the database. When a global user attempts to connect to the database, the enterprise directory is queried to obtain any global roles associated with the user.

The following statement creates a global role:

```
CREATE ROLE supervisor IDENTIFIED GLOBALLY;
```

Global roles are one component of enterprise user security. A global role only applies to one database, but it can be granted to an enterprise role defined in the enterprise directory. An enterprise role is a directory structure which contains global roles on multiple databases, and which can be granted to enterprise users.

A general discussion of global authentication and authorization of users, and its role in enterprise user management, was presented earlier in "Global Authentication and Authorization" on page 9-5.

**See Also:** *Oracle Advanced Security Administrator's Guide* and *Oracle Internet Directory Administrator's Guide* for information about enterprise user management and how to implement it

### Dropping Roles

In some cases, it may be appropriate to drop a role from the database. The security domains of all users and roles granted a dropped role are immediately changed to reflect the absence of the dropped role's privileges. All indirectly granted roles of the dropped role are also removed from affected security domains. Dropping a role automatically removes the role from all users' default role lists.

Because the creation of objects is not dependent on the privileges received through a role, tables and other objects are not dropped when a role is dropped.

You can drop a role using the SQL statement `DROP ROLE`. To drop a role, you must have the `DROP ANY ROLE` system privilege or have been granted the role with the `ADMIN OPTION`.

The following statement drops the role `CLERK`:

```
DROP ROLE clerk;
```

## Granting User Privileges and Roles

This section describes the granting of privileges and roles, and contains the following topics:

- Granting System Privileges and Roles
- Granting Object Privileges
- Granting Privileges on Columns

It is also possible to grant roles to a user connected through a middle tier or proxy. This is discussed in "Proxy Authentication and Authorization" on page 9-8.

### Granting System Privileges and Roles

You can grant system privileges and roles to other users and roles using the `GRANT` statement. The following privileges are required:

- To grant a system privilege, you must have been granted the system privilege with the `ADMIN OPTION` or have been granted the `GRANT ANY PRIVILEGE` system privilege.

- To grant a role, you must have been granted the role with the ADMIN OPTION or have been granted the GRANT ANY ROLE system privilege.

  **Note:** You cannot grant a role that is IDENTIFIED GLOBALLY to anything. The granting (and revoking) of global roles is controlled entirely by the enterprise directory service.

The following statement grants the system privilege CREATE SESSION and the accts_pay role to the user jward:

```
GRANT CREATE SESSION, accts_pay TO jward;
```

**Notes:**

- Object privileges *cannot* be granted along with system privileges and roles in the same GRANT statement.
- Each installation should create its own roles and assign only those privileges that are needed. For example, it is unwise to grant CONNECT if all that is needed is CREATE SESSION, since CONNECT includes several additional privileges: see Table 10–1 on page 10-18. Creating its own roles gives an organization detailed control of the privileges it assigns, and protects it in case Oracle were to change or remove roles that it defines.

## Granting the ADMIN OPTION

A user or role that is granted a privilege or role specifying the WITH ADMIN OPTION clause has several expanded capabilities:

- The grantee can grant or revoke the system privilege or role to or from *any* user or other role in the database. Users cannot revoke a role from themselves.
- The grantee can further grant the system privilege or role with the ADMIN OPTION.
- The grantee of a role can alter or drop the role.

In the following statement, the security administrator grants the new_dba role to michael:

```
GRANT new_dba TO michael WITH ADMIN OPTION;
```

The user michael cannot only use all of the privileges implicit in the new_dba role, but can grant, revoke, or drop the new_dba role as deemed necessary. Because of these powerful capabilities, exercise caution when granting system privileges or roles with the ADMIN OPTION. Such privileges are usually reserved for a security administrator and rarely granted to other administrators or users of the system.

When a user creates a role, the role is automatically granted to the creator with the ADMIN OPTION

### Creating a New User with the GRANT Statement

Oracle enables you to create a new user with the GRANT statement. If you specify a password using the IDENTIFIED BY clause, and the username/password does not exist in the database, a new user with that username and password is created. The following example creates ssmith as a new user while granting ssmith the CONNECT system privilege:

```
GRANT CONNECT TO ssmith IDENTIFIED BY p1q2r3;
```

**See Also:** "Creating Users" on page 10-2

## Granting Object Privileges

You also use the GRANT statement to grant object privileges to roles and users. To grant an object privilege, you must fulfill one of the following conditions:

- You own the object specified.
- You possess the GRANT ANY OBJECT PRIVILEGE system privilege that enables you to grant and revoke privileges on behalf of the object owner.
- The WITH GRANT OPTION clause was specified when you were granted the object privilege by its owner.

**Note:** System privileges and roles cannot be granted along with object privileges in the same GRANT statement.

The following statement grants the SELECT, INSERT, and DELETE object privileges for all columns of the emp table to the users jfee and tsmith:

```
GRANT SELECT, INSERT, DELETE ON emp TO jfee, tsmith;
```

To grant all object privileges on the salary view to the user jfee, use the ALL keyword, as shown in the following example:

```
GRANT ALL ON salary TO jfee;
```

### Specifying the GRANT OPTION

Specify WITH GRANT OPTION to enable the grantee to grant the object privileges to other users and roles. The user whose schema contains an object is automatically granted all associated object privileges with the `GRANT OPTION`. This special privilege allows the grantee several expanded privileges:

- The grantee can grant the object privilege to any users in the database, with or without the `GRANT OPTION`, or to any role in the database.
- If both of the following are true, the grantee can create views on the table and grant the corresponding privileges on the views to any user or role in the database.
  - The grantee receives object privileges for the table with the `GRANT OPTION`.
  - The grantee has the `CREATE VIEW` or `CREATE ANY VIEW` system privilege.

The `GRANT OPTION` is not valid when granting an object privilege to a role. Oracle prevents the propagation of object privileges through roles so that grantees of a role cannot propagate object privileges received by means of roles.

### Granting Object Privileges on Behalf of the Object Owner

The `GRANT ANY OBJECT PRIVILEGE` system privilege allows users to grant and revoke any object privilege on behalf of the object owner. This provides a convenient means for database and application administrators to grant access to objects in any schema without requiring that they connect to the schema. This eliminates the need to maintain login credentials for schema owners so that they can grant access to objects, and it reduces the number of connections required during configuration.

This system privilege is part of the Oracle supplied `DBA` role and is thus granted (with the `ADMIN OPTION`) to any user connecting `AS SYSDBA` (user `SYS`). As with other system privileges, the `GRANT ANY OBJECT PRIVILEGE` system privilege can only be granted by a user who possesses the `ADMIN OPTION`.

When you exercise the `GRANT ANY OBJECT PRIVILEGE` system privilege to grant an object privilege to a user, if you already possess the object privilege with the `GRANT OPTION`, then the grant is performed in the usual way. In this case, you become the grantor of the grant. If you do not possess the object privilege, then the object owner is shown as the grantor, even though you, with the GRANT ANY OBJECT PRIVILEGE system privilege, actually performed the grant.

---

**Note:** The audit record generated by the GRANT statement will always show the real user who performed the grant.

---

For example, consider the following. User adams possesses the GRANT ANY OBJECT PRIVILEGE system privilege. He does not possess any other grant privileges. He issues the following statement:

```
GRANT SELECT ON hr.employees TO blake WITH GRANT OPTION;
```

If you examine the DBA_TAB_PRIVS view, you will see that hr is shown as being the grantor of the privilege:

```
SQL> SELECT GRANTEE, OWNER, GRANTOR, PRIVILEGE, GRANTABLE
  2>     FROM DBA_TAB_PRIVS
  3>     WHERE TABLE_NAME = 'EMPLOYEES' and OWNER = 'HR';

GRANTEE  OWNER GRANTOR PRIVILEGE    GRANTABLE
-------- ----- ------- -----------  ----------
BLAKE    HR    HR       SELECT      YES
```

Now assume that blake also has the GRANT ANY OBJECT PRIVILEGE system. He, issues the following statement:

```
GRANT SELECT ON hr.employees TO clark;
```

In this case, when you again query the DBA_TAB_PRIVS view, you see that blake is shown as being the grantor of the privilege:

```
GRANTEE  OWNER GRANTOR  PRIVILEGE    GRANTABLE
-------- ----- -------- --------     ----------
BLAKE    HR    HR       SELECT       YES
CLARK    HR    BLAKE    SELECT       NO
```

This occurs because blake already possesses the SELECT privilege on hr.employees with the GRANT OPTION.

**See Also:** "Revoking Object Privileges on Behalf of the Object Owner" on page 10-31

### Granting Privileges on Columns

You can grant INSERT, UPDATE, or REFERENCES privileges on individual columns in a table.

---

**Caution:** Before granting a column-specific INSERT privilege, determine if the table contains any columns on which NOT NULL constraints are defined. Granting selective insert capability without including the NOT NULL columns prevents the user from inserting any rows into the table. To avoid this situation, make sure that each NOT NULL column is either insertable or has a non-NULL default value. Otherwise, the grantee will not be able to insert rows into the table and will receive an error.

---

The following statement grants INSERT privilege on the acct_no column of the accounts table to scott:

```
GRANT INSERT (acct_no) ON accounts TO scott;
```

In another example, object privilege for the ename and job columns of the emp table are granter to the users jfee and tsmith:

```
GRANT INSERT(ename, job) ON emp TO jfee, tsmith;
```

### Row-Level Access Control

You can also provide access control at the row level, that is, within objects, using Virtual Private Database (VPD) or Oracle Label Security.

**See Also:**

- Chapter 13, "Using Virtual Private Database to Implement Application Security Policies", and
- Adding Policies for Column-Level VPD, in Chapter 14, "Implementing Application Context and Fine-Grained Access Control"

## Revoking User Privileges and Roles

This section describes aspects of revoking user privileges and roles, and contains the following topics:

- Revoking System Privileges and Roles

- Revoking Object Privileges
- Cascading Effects of Revoking Privileges

## Revoking System Privileges and Roles

You can revoke system privileges and roles using the SQL statement REVOKE.

Any user with the ADMIN OPTION for a system privilege or role can revoke the privilege or role from any other database user or role. The revoker does not have to be the user that originally granted the privilege or role. Users with GRANT ANY ROLE can revoke *any* role.

The following statement revokes the CREATE TABLE system privilege and the accts_rec role from tsmith:

```
REVOKE CREATE TABLE, accts_rec FROM tsmith;
```

**Note:** The ADMIN OPTION for a system privilege or role cannot be selectively revoked. The privilege or role must be revoked and then the privilege or role re-granted without the ADMIN OPTION.

## Revoking Object Privileges

The REVOKE statement is used to revoke object privileges. To revoke an object privilege, you must fulfill one of the following conditions:

- You previously granted the object privilege to the user or role.
- You possess the GRANT ANY OBJECT PRIVILEGE system privilege that enables you to grant and revoke privileges on behalf of the object owner.

You can only revoke the privileges that you, the grantor, directly authorized, not the grants made by other users to whom you granted the GRANT OPTION. However, there is a cascading effect. The object privilege grants propagated using the GRANT OPTION are revoked if a grantor's object privilege is revoked.

Assuming you are the original grantor, the following statement revokes the SELECT and INSERT privileges on the emp table from the users jfee and tsmith:

```
REVOKE SELECT, insert ON emp FROM jfee, tsmith;
```

The following statement revokes all object privileges for the dept table that you originally granted to the human_resource role

```
REVOKE ALL ON dept FROM human_resources;
```

---

**Note:** The GRANT OPTION for an object privilege cannot be selectively revoked. The object privilege must be revoked and then re-granted without the GRANT OPTION. Users cannot revoke object privileges from themselves.

---

### Revoking Object Privileges on Behalf of the Object Owner

The GRANT ANY OBJECT PRIVILEGE system privilege enables you to revoke any specified object privilege where the object owner is the grantor. This occurs when the object privilege is granted by the object owner, or on behalf of the owner by any user holding the GRANT ANY OBJECT PRIVILEGE system privilege.

In a situation where the object privilege has been granted by both the owner of the object and the user executing the REVOKE statement (who has both the specific object privilege and the GRANT ANY OBJECT PRIVILEGE system privilege), Oracle only revokes the object privilege granted by the user issuing the REVOKE. This can be illustrated by continuing the example started in "Granting Object Privileges on Behalf of the Object Owner" on page 10-27.

At this point, blake has granted the SELECT privilege on hr.employees to clark. Even though blake possesses the GRANT ANY OBJECT PRIVILEGE system privilege, he also holds the specific object privilege, thus this grant is attributed to him. Assume that hr also grants the SELECT privilege on hr.employees to clark. A query of the DBA_TAB_PRIVS view shows that the following grants are in effect for the hr.employees table:

```
GRANTEE  OWNER GRANTOR PRIVILEGE    GRANTABLE
-------- ----- ------- -----------  ----------
BLAKE    HR    HR        SELECT       YES
CLARK    HR    BLAKE     SELECT       NO
CLARK    HR    HR        SELECT       NO
```

User blake now issues the following REVOKE statement:

```
REVOKE  SELECT ON hr.employees FROM clark;
```

Only the object privilege for clark granted by blake is removed. The grant by the object owner, hr, remains.

```
GRANTEE  OWNER GRANTOR PRIVILEGE    GRANTABLE
-------- ----- ------- -----------  ----------
BLAKE    HR    HR        SELECT     YES
CLARK    HR    HR        SELECT     NO
```

If blake issues the REVOKE statement again, this time the effect will be to remove the object privilege granted by hr.

> **See Also:** "Granting Object Privileges on Behalf of the Object Owner" on page 10-27

### Revoking Column-Selective Object Privileges

Although users can grant column-selective INSERT, UPDATE, and REFERENCES privileges for tables and views, they cannot selectively revoke column specific privileges with a similar REVOKE statement. Instead, the grantor must first revoke the object privilege for all columns of a table or view, and then selectively re-grant the column-specific privileges that should remain.

For example, assume that role human_resources has been granted the UPDATE privilege on the deptno and dname columns of the table dept. To revoke the UPDATE privilege on just the deptno column, issue the following two statements:

```
REVOKE UPDATE ON dept FROM human_resources;
GRANT UPDATE (dname) ON dept TO human_resources;
```

The REVOKE statement revokes UPDATE privilege on all columns of the dept table from the role human_resources. The GRANT statement re-grants UPDATE privilege on the dname column to the role human_resources.

### Revoking the REFERENCES Object Privilege

If the grantee of the REFERENCES object privilege has used the privilege to create a foreign key constraint (that currently exists), the grantor can revoke the privilege only by specifying the CASCADE CONSTRAINTS option in the REVOKE statement:

```
REVOKE REFERENCES ON dept FROM jward CASCADE CONSTRAINTS;
```

Any foreign key constraints currently defined that use the revoked REFERENCES privilege are dropped when the CASCADE CONSTRAINTS clause is specified.

## Cascading Effects of Revoking Privileges

Depending on the type of privilege, there may be cascading effects when a privilege is revoked.

### System Privileges

There are no cascading effects when revoking a system privilege related to DDL operations, regardless of whether the privilege was granted with or without the ADMIN OPTION. For example, assume the following:

1. The security administrator grants the CREATE TABLE system privilege to jfee with the ADMIN OPTION.
2. User jfee creates a table.
3. User jfee grants the CREATE TABLE system privilege to tsmith.
4. User tsmith creates a table.
5. The security administrator revokes the CREATE TABLE system privilege from jfee.
6. User jfee's table continues to exist. tsmith still has the table and the CREATE TABLE system privilege.

Cascading effects can be observed when revoking a system privilege related to a DML operation. If SELECT ANY TABLE is revoked from a user, then all procedures contained in the users schema relying on this privilege will fail until the privilege is reauthorized.

### Object Privileges

Revoking an object privilege can have cascading effects that should be investigated before issuing a REVOKE statement.

- Object definitions that depend on a DML object privilege can be affected if the DML object privilege is revoked. For example, assume the procedure body of the test procedure includes a SQL statement that queries data from the emp table. If the SELECT privilege on the emp table is revoked from the owner of the test procedure, the procedure can no longer be executed successfully.
- When a REFERENCES privilege for a table is revoked from a user, any foreign key integrity constraints defined by the user that require the dropped REFERENCES privilege are automatically dropped. For example, assume that the user jward is granted the REFERENCES privilege for the deptno column of the dept table and creates a foreign key on the deptno column in the emp table that references the deptno column. If the references privilege on the deptno column of the dept table is revoked, the foreign key constraint on the deptno column of the emp table is dropped in the same operation.

- The object privilege grants propagated using the GRANT OPTION are revoked if a grantor's object privilege is revoked. For example, assume that user1 is granted the SELECT object privilege with the GRANT OPTION, and grants the SELECT privilege on emp to user2. Subsequently, the SELECT privilege is revoked from user1. This REVOKE is cascaded to user2 as well. Any objects that depended on user1's and user2's revoked SELECT privilege can also be affected, as described in previous bullet items.

Object definitions that require the ALTER and INDEX DDL object privileges are not affected if the ALTER or INDEX object privilege is revoked. For example, if the INDEX privilege is revoked from a user that created an index on someone else's table, the index continues to exist after the privilege is revoked.

## Granting to and Revoking from the User Group PUBLIC

Privileges and roles can also be granted to and revoked from the user group PUBLIC. Because PUBLIC is accessible to every database user, all privileges and roles granted to PUBLIC are accessible to every database user.

Security administrators and database users should grant a privilege or role to PUBLIC only if every database user requires the privilege or role. This recommendation reinforces the general rule that at any given time, each database user should only have the privileges required to accomplish the group's current tasks successfully.

Revoking a privilege from PUBLIC can cause significant cascading effects. If any privilege related to a DML operation is revoked from PUBLIC (for example, SELECT ANY TABLE, UPDATE ON emp), all procedures in the database, including functions and packages, must be *reauthorized* before they can be used again. Therefore, exercise caution when granting and revoking DML-related privileges to PUBLIC.

**See Also:**

- "Managing Object Dependencies" in *Oracle Database Administrator's Guide* for more information about object dependencies
- A Security Checklist, in Chapter 7, "Security Policies"

## When Do Grants and Revokes Take Effect?

Depending on what is granted or revoked, a grant or revoke takes effect at different times:

- All grants/revokes of system and object privileges to anything (users, roles, and PUBLIC) are immediately observed.
- All grants/revokes of roles to anything (users, other roles, PUBLIC) are only observed when a current user session issues a SET ROLE statement to re-enable the role after the grant/revoke, or when a new user session is created after the grant/revoke.

You can see which roles are currently enabled by examining the SESSION_ROLES data dictionary view.

### The SET ROLE Statement

During the session, the user or an application can use the SET ROLE statement any number of times to change the roles currently enabled for the session. You must already have been granted the roles that you name in the SET ROLE statement.The number of roles that can be concurrently enabled is limited by the initialization parameter MAX_ENABLED_ROLES.

This example enables the role clerk, which you have already been granted, and specifies the password.

```
SET ROLE clerk IDENTIFIED BY bicentennial;
```

You can disable all roles with the following statement:

```
SET ROLE NONE;
```

### Specifying Default Roles

When a user logs on, Oracle enables all privileges granted explicitly to the user and all privileges in the user's default roles.

A user's list of default roles can be set and altered using the ALTER USER statement. The ALTER USER statement enables you to specify roles that are to be enabled when a user connects to the database, without requiring the user to specify the roles' passwords. The user must have already been directly granted the roles with a GRANT statement. You cannot specify as a default role any role managed by an external service including a directory service (external roles or global roles).

The following example establishes default roles for user jane:

```
ALTER USER jane DEFAULT ROLE payclerk, pettycash;
```

You cannot set a user's default roles in the CREATE USER statement. When you first create a user, the user's default role setting is ALL, which causes all roles subsequently granted to the user to be default roles. Use the ALTER USER statement to limit the user's default roles.

---

**Caution:** When you create a role (other than a user role), it is granted to you implicitly and added as a default role. You receive an error at login if you have more than MAX_ENABLED_ROLES. You can avoid this error by altering the user's default roles to be less than MAX_ENABLED_ROLES. Thus, you should change the DEFAULT ROLE settings of SYS and SYSTEM before creating user roles.

---

### Restricting the Number of Roles that a User Can Enable

A user can enable as many roles as specified by the initialization parameter MAX_ENABLED_ROLES. All indirectly granted roles enabled as a result of enabling a primary role are included in this count. The database administrator can alter this limitation by modifying the value for this parameter. Higher values permit each user session to have more concurrently enabled roles, but these values also cause more memory to be used for each user session. This occurs because the PGA size requires four bytes for each role in each session. Determine the highest number of roles that will be concurrently enabled by any one user and use this value for the MAX_ENABLED_ROLES parameter.

## Granting Roles Using the Operating System or Network

This section describes aspects of granting roles through your operating system or network, and contains the following topics:

- Using Operating System Role Identification
- Using Operating System Role Management
- Granting and Revoking Roles When OS_ROLES=TRUE
- Enabling and Disabling Roles When OS_ROLES=TRUE
- Using Network Connections with Operating System Role Management

Instead of a security administrator explicitly granting and revoking database roles to and from users using GRANT and REVOKE statements, the operating system that operates Oracle can grant roles to users at connect time. Roles can be administered using the operating system and passed to Oracle when a user creates a session. As part of this mechanism, each user's default roles and the roles granted to a user with the `ADMIN OPTION` can be identified. Even if the operating system is used to authorize users for roles, all roles must be created in the database and privileges assigned to the role with `GRANT` statements.

Roles can also be granted through a network service.

The advantage of using the operating system to identify a user's database roles is that privilege management for an Oracle database can be externalized. The security facilities offered by the operating system control a user's privileges. This option may offer advantages of centralizing security for a number of system activities, such as the following situation:

- MVS Oracle administrators want RACF groups to identify a database user's roles.
- UNIX Oracle administrators want UNIX groups to identify a database user's roles.
- VMS Oracle administrators want to use rights identifiers to identify a database user's roles.

The main disadvantage of using the operating system to identify a user's database roles is that privilege management can only be performed at the role level. Individual privileges cannot be granted using the operating system, but can still be granted inside the database using `GRANT` statements.

A secondary disadvantage of using this feature is that by default users cannot connect to the database through the shared server, or any other network connection, if the operating system is managing roles. However, you can change this default; see "Using Network Connections with Operating System Role Management" on page 10-40.

**Note:** The features described in this section are available only on some operating systems. See your operating system specific Oracle documentation to determine if you can use these features.

## Using Operating System Role Identification

To operate a database so that it uses the operating system to identify each user's database roles when a session is created, set the initialization parameter `OS_ROLES` to `TRUE` (and restart the instance, if it is currently running). When a user attempts to create a session with the database, Oracle initializes the user's security domain using the database roles identified by the operating system.

To identify database roles for a user, each Oracle user's operating system account must have operating system identifiers (these may be called groups, rights identifiers, or other similar names) that indicate which database roles are to be available for the user. Role specification can also indicate which roles are the default roles of a user and which roles are available with the `ADMIN OPTION`. No matter which operating system is used, the role specification at the operating system level follows the format:

ora_*ID*_*ROLE*[_[d][a]]

where:

- `ID` has a definition that varies on different operating systems. For example, on VMS, `ID` is the instance identifier of the database; on MVS, it is the machine type; on UNIX, it is the system `ID`.

> **Note:** `ID` is case sensitive to match your `ORACLE_SID`. `ROLE` is not case sensitive.

- `ROLE` is the name of the database role.
- `d` is an optional character that indicates this role is to be a default role of the database user.
- `a` is an optional character that indicates this role is to be granted to the user with the `ADMIN OPTION`. This allows the user to grant the role to other roles only. Roles cannot be granted to users if the operating system is used to manage roles.

> **Note:** If either the `d` or `a` characters are specified, they must be preceded by an underscore.

For example, an operating system account might have the following roles identified in its profile:

```
ora_PAYROLL_ROLE1
ora_PAYROLL_ROLE2_a
ora_PAYROLL_ROLE3_d
ora_PAYROLL_ROLE4_da
```

When the corresponding user connects to the payroll instance of Oracle, role3 and role4 are defaults, while role2 and role4 are available with the ADMIN OPTION.

## Using Operating System Role Management

When you use operating system managed roles, it is important to note that database roles are being granted to an operating system user. Any database user to which the operating system user is able to connect will have the authorized database roles enabled. For this reason, you should consider defining all Oracle users as IDENTIFIED EXTERNALLY if you are using OS_ROLES = TRUE, so that the database accounts are tied to the operating system account that was granted privileges.

## Granting and Revoking Roles When OS_ROLES=TRUE

If OS_ROLES is set to TRUE, the operating system completely manages the grants and revokes of roles *to users*. Any previous grants of roles to users using GRANT statements do not apply; however, they are still listed in the data dictionary. Only the role grants made at the operating system level to users apply. Users can still grant privileges to roles and users.

**Note:** If the operating system grants a role to a user with the ADMIN OPTION, the user can grant the role only to other roles.

## Enabling and Disabling Roles When OS_ROLES=TRUE

If OS_ROLES is set to TRUE, any role granted by the operating system can be dynamically enabled using the SET ROLE statement. This still applies, even if the role was defined to require a password or operating system authorization. However, any role not identified in a user's operating system account cannot be specified in a SET ROLE statement, even if a role has been granted using a GRANT statement when OS_ROLES = FALSE. (If you specify such a role, Oracle ignores it.)

When OS_ROLES = TRUE, a user can enable as many roles as specified by the initialization parameter MAX_ENABLED_ROLES.

### Using Network Connections with Operating System Role Management

If you choose to have the operating system to manage roles, by default users cannot connect to the database through the shared server. This restriction is the default because a remote user could impersonate another operating system user over a non-secure connection.

If you are not concerned with this security risk and want to use operating system role management with the shared server, or any other network connection, set the initialization parameter REMOTE_OS_ROLES in the database's initialization parameter file to TRUE. The change will take effect the next time you start the instance and mount the database. The default setting of this parameter is FALSE.

## Viewing Privilege and Role Information

To access information about grants of privileges and roles, you can query the following data dictionary views:

| View | Description |
|---|---|
| DBA_COL_PRIVS<br>ALL_COL_PRIVS<br>USER_COL_PRIVS | DBA view describes all column object grants in the database. ALL view describes all column object grants for which the current user or PUBLIC is the object owner, grantor, or grantee. USER view describes column object grants for which the current user is the object owner, grantor, or grantee. |
| ALL_COL_PRIVS_MADE<br>USER_COL_PRIVS_MADE | ALL view lists column object grants for which the current user is object owner or grantor. USER view describes column object grants for which the current user is the grantor. |
| ALL_COL_PRIVS_RECD<br>USER_COL_PRIVS_RECD | ALL view describes column object grants for which the current user or PUBLIC is the grantee. USER view describes column object grants for which the current user is the grantee. |
| DBA_TAB_PRIVS<br>ALL_TAB_PRIVS<br>USER_TAB_PRIVS | DBA view lists all grants on all objects in the database. ALL view lists the grants on objects where the user or PUBLIC is the grantee. USER view lists grants on all objects where the current user is the grantee. |
| ALL_TAB_PRIVS_MADE<br>USER_TAB_PRIVS_MADE | ALL view lists the all object grants made by the current user or made on the objects owned by the current user. USER view lists grants on all objects owned by the current user. |
| ALL_TAB_PRIVS_RECD<br>USER_TAB_PRIVS_RECD | ALL view lists object grants for which the user or PUBLIC is the grantee. USER view lists object grants for which the current user is the grantee. |
| DBA_ROLES | This view lists all roles that exist in the database. |

| View | Description |
|---|---|
| DBA_ROLE_PRIVS<br>USER_ROLE_PRIVS | DBA view lists roles granted to users and roles. USER view lists roles granted to the current user. |
| DBA_SYS_PRIVS<br>USER_SYS_PRIVS | DBA view lists system privileges granted to users and roles. USER view lists system privileges granted to the current user. |
| ROLE_ROLE_PRIVS | This view describes roles granted to other roles. Information is provided only about roles to which the user has access. |
| ROLE_SYS_PRIVS | This view contains information about system privileges granted to roles. Information is provided only about roles to which the user has access. |
| ROLE_TAB_PRIVS | This view contains information about object privileges granted to roles. Information is provided only about roles to which the user has access. |
| SESSION_PRIVS | This view lists the privileges that are currently enabled for the user. |
| SESSION_ROLES | This view lists the roles that are currently enabled to the user. |

Some examples of using these views follow. For these examples, assume the following statements have been issued:

```
CREATE ROLE security_admin IDENTIFIED BY honcho;

GRANT CREATE PROFILE, ALTER PROFILE, DROP PROFILE,
    CREATE ROLE, DROP ANY ROLE, GRANT ANY ROLE, AUDIT ANY,
    AUDIT SYSTEM, CREATE USER, BECOME USER, ALTER USER, DROP USER
    TO security_admin WITH ADMIN OPTION;

GRANT SELECT, DELETE ON SYS.AUD$ TO security_admin;

GRANT security_admin, CREATE SESSION TO swilliams;

GRANT security_admin TO system_administrator;

GRANT CREATE SESSION TO jward;

GRANT SELECT, DELETE ON emp TO jward;

GRANT INSERT (ename, job) ON emp TO swilliams, jward;
```

**See Also:** *Oracle Database Reference* for a detailed description of these data dictionary views

## Listing All System Privilege Grants

The following query returns all system privilege grants made to roles and users:

```
SELECT * FROM DBA_SYS_PRIVS;

GRANTEE            PRIVILEGE                         ADM
--------------     --------------------------------- ---
SECURITY_ADMIN     ALTER PROFILE                     YES
SECURITY_ADMIN     ALTER USER                        YES
SECURITY_ADMIN     AUDIT ANY                         YES
SECURITY_ADMIN     AUDIT SYSTEM                      YES
SECURITY_ADMIN     BECOME USER                       YES
SECURITY_ADMIN     CREATE PROFILE                    YES
SECURITY_ADMIN     CREATE ROLE                       YES
SECURITY_ADMIN     CREATE USER                       YES
SECURITY_ADMIN     DROP ANY ROLE                     YES
SECURITY_ADMIN     DROP PROFILE                      YES
SECURITY_ADMIN     DROP USER                         YES
SECURITY_ADMIN     GRANT ANY ROLE                    YES
SWILLIAMS          CREATE SESSION                    NO
JWARD              CREATE SESSION                    NO
```

## Listing All Role Grants

The following query returns all the roles granted to users and other roles:

```
SELECT * FROM DBA_ROLE_PRIVS;

GRANTEE            GRANTED_ROLE                         ADM
------------------ ------------------------------------ ---
SWILLIAMS          SECURITY_ADMIN                       NO
```

## Listing Object Privileges Granted to a User

The following query returns all object privileges (not including column-specific privileges) granted to the specified user:

```
SELECT TABLE_NAME, PRIVILEGE, GRANTABLE FROM DBA_TAB_PRIVS
    WHERE GRANTEE = 'JWARD';

TABLE_NAME   PRIVILEGE    GRANTABLE
-----------  ------------ ----------
EMP          SELECT       NO
EMP          DELETE       NO
```

To list all the column-specific privileges that have been granted, use the following query:

```
SELECT GRANTEE, TABLE_NAME, COLUMN_NAME, PRIVILEGE
    FROM DBA_COL_PRIVS;

GRANTEE      TABLE_NAME     COLUMN_NAME      PRIVILEGE
-----------  ------------   -------------    --------------
SWILLIAMS    EMP            ENAME            INSERT
SWILLIAMS    EMP            JOB              INSERT
JWARD        EMP            NAME             INSERT
JWARD        EMP            JOB              INSERT
```

## Listing the Current Privilege Domain of Your Session

The following query lists all roles currently enabled for the issuer:

```
SELECT * FROM SESSION_ROLES;
```

If `swilliams` has enabled the `security_admin` role and issues this query, Oracle returns the following information:

```
ROLE
------------------------------
SECURITY_ADMIN
```

The following query lists all system privileges currently available in the issuer's security domain, both from explicit privilege grants and from enabled roles:

```
SELECT * FROM SESSION_PRIVS;
```

If `swilliams` has the `security_admin` role enabled and issues this query, Oracle returns the following results:

```
PRIVILEGE
----------------------------------------
AUDIT SYSTEM
CREATE SESSION
CREATE USER
BECOME USER
ALTER USER
DROP USER
CREATE ROLE
DROP ANY ROLE
GRANT ANY ROLE
AUDIT ANY
```

```
CREATE PROFILE
ALTER PROFILE
DROP PROFILE
```

If the `security_admin` role is disabled for `swilliams`, the first query would have returned no rows, while the second query would only return a row for the `CREATE SESSION` privilege grant.

## Listing Roles of the Database

The `DBA_ROLES` data dictionary view can be used to list all roles of a database and the authentication used for each role. For example, the following query lists all the roles in the database:

```
SELECT * FROM DBA_ROLES;

ROLE                  PASSWORD
----------------      --------
CONNECT               NO
RESOURCE              NO
DBA                   NO
SECURITY_ADMIN        YES
```

## Listing Information About the Privilege Domains of Roles

The `ROLE_ROLE_PRIVS`, `ROLE_SYS_PRIVS`, and `ROLE_TAB_PRIVS` data dictionary views contain information on the privilege domains of roles.

For example, the following query lists all the roles granted to the `system_admin` role:

```
SELECT GRANTED_ROLE, ADMIN_OPTION
   FROM ROLE_ROLE_PRIVS
   WHERE ROLE = 'SYSTEM_ADMIN';

GRANTED_ROLE              ADM
----------------          ----
SECURITY_ADMIN            NO
```

The following query lists all the system privileges granted to the `security_admin` role:

```
SELECT * FROM ROLE_SYS_PRIVS WHERE ROLE = 'SECURITY_ADMIN';

ROLE                    PRIVILEGE                     ADM
```

```
---------------------- ----------------------------  ---
SECURITY_ADMIN          ALTER PROFILE                YES
SECURITY_ADMIN          ALTER USER                   YES
SECURITY_ADMIN          AUDIT ANY                    YES
SECURITY_ADMIN          AUDIT SYSTEM                 YES
SECURITY_ADMIN          BECOME USER                  YES
SECURITY_ADMIN          CREATE PROFILE               YES
SECURITY_ADMIN          CREATE ROLE                  YES
SECURITY_ADMIN          CREATE USER                  YES
SECURITY_ADMIN          DROP ANY ROLE                YES
SECURITY_ADMIN          DROP PROFILE                 YES
SECURITY_ADMIN          DROP USER                    YES
SECURITY_ADMIN          GRANT ANY ROLE               YES
```

The following query lists all the object privileges granted to the `security_admin` role:

```
SELECT TABLE_NAME, PRIVILEGE FROM ROLE_TAB_PRIVS
    WHERE ROLE = 'SECURITY_ADMIN';

TABLE_NAME                         PRIVILEGE
---------------------------    ----------------
AUD$                               DELETE
AUD$                               SELECT
```

# 11 Configuring and Administering Auditing

Auditing is always about accountability, and frequently is done to protect and preserve privacy for the information stored in databases. Concern about privacy policies and practices has been rising steadily with the ubiquitous use of databases in businesses and on the Internet. Oracle Database provides a depth of auditing that readily enables system administrators to implement enhanced protections, early detection of suspicious activities, and finely-tuned security responses.

The types of auditing available in Oracle systems were described in Chapter 8, "Database Auditing: Security Considerations".

The present chapter explains how to choose the types of auditing you need, how to manage that auditing, and how to use the information gained, in the following sections:

- Actions Audited by Default
- Guidelines for Auditing
- What Information is Contained in the Audit Trail?
- Managing the Standard Audit Trail
- Viewing Database Audit Trail Information
- Fine-Grained Auditing

## Actions Audited by Default

Regardless of whether database auditing is enabled, Oracle *always* audits certain database-related operations and writes them to the operating system audit file. This fact is called mandatory auditing, and it includes the following operations:

- Connections to the instance with administrator privileges

An audit record is generated that lists the operating system user connecting to Oracle as `SYSOPER` or `SYSDBA`. This provides for accountability of users with administrative privileges. Full auditing for these users can be enabled as explained in "Auditing Administrative Users" on page 11-4.

- Database startup

  An audit record is generated that lists the operating system user starting the instance, the user's terminal identifier, the date and time stamp. This data is stored in the operating system audit trail because the database audit trail is not available until after startup has successfully completed.

- Database shutdown

  An audit record is generated that lists the operating system user shutting down the instance, the user's terminal identifier, and the date and time stamp.

## Guidelines for Auditing

Oracle Database 10*g* gives you the option of sending audit records to the database audit trail or your operating system's audit trail, when the operating system is capable of receiving them. The audit trail for database administrators, for example, is typically written to a secure location in the operating system. Writing audit trails to the operating system provides a way for a separate auditor who is root on the operating system to hold all DBAs (who don't have root access) accountable for their actions. These options, added to the broad selection of audit options and customizable triggers or stored procedures, give you the flexibility to implement an auditing scheme that suits your specific business needs.

This section describes guidelines for auditing and contains the following topics:

- Keep Audited Information Manageable
- Auditing Normal Database Activity
- Auditing Suspicious Database Activity
- Auditing Administrative Users
- Using Triggers
- Decide Whether to Use the Database or Operating System Audit Trail

## Keep Audited Information Manageable

Although auditing is relatively inexpensive, limit the number of audited events as much as possible. Doing so minimizes the performance impact on the execution of audited statements and the size of the audit trail, making it easier to analyze and understand.

Use the following general guidelines when devising an auditing strategy:

- Evaluate your purpose for auditing.

  After you have a clear understanding of the reasons for auditing, you can devise an appropriate auditing strategy and avoid unnecessary auditing.

  For example, suppose you are auditing to investigate suspicious database activity. This information by itself is not specific enough. What types of suspicious database activity do you suspect or have you noticed? A more focused auditing purpose might be to audit unauthorized deletions from arbitrary tables in the database. This purpose narrows the type of action being audited and the type of object being affected by the suspicious activity.

- Audit knowledgeably.

  Audit the minimum number of statements, users, or objects required to get the targeted information. This prevents unnecessary audit information from cluttering the meaningful information and consuming valuable space in the `SYSTEM` tablespace. Balance your need to gather sufficient security information with your ability to store and process it.

  For example, if you are auditing to gather information about database activity, determine exactly what types of activities you are tracking, audit only the activities of interest, and audit only for the amount of time necessary to gather the information you desire. As another example, do not audit *objects* if you are only interested in each session's logical I/O information.

## Auditing Normal Database Activity

When your purpose for auditing is to gather historical information about particular database activities, use the following guidelines:

- Audit only pertinent actions.

  To avoid cluttering meaningful information with useless audit records and reduce the amount of audit trail administration, only audit the targeted database activities.

- Archive audit records and purge the audit trail.

After you have collected the required information, archive the audit records of interest and purge the audit trail of this information.

- Privacy considerations

  Privacy regulations often lead to additional business privacy policies. Most privacy laws require businesses to monitor access to personally identifiable information (**PII**), and such monitoring is implemented by auditing. A business-level privacy policy should address all relevant aspects of data access and user accountability, including technical, legal, and company-policy concerns.

## Auditing Suspicious Database Activity

When you audit to monitor suspicious database activity, use the following guidelines:

- Audit generally, then specifically.

  When starting to audit for suspicious database activity, it is common that not much information is available to target specific users or schema objects. Therefore, audit options must be set more generally at first. Once preliminary audit information is recorded and analyzed, the general audit options should be turned off and more specific audit options enabled. This process should continue until enough evidence is gathered to make concrete conclusions about the origin of the suspicious database activity.

- Protect the audit trail.

  When auditing for suspicious database activity, protect the audit trail so that audit information cannot be added, changed, or deleted without being audited.

**See Also:** "Protecting the Standard Audit Trail" on page 11-21

## Auditing Administrative Users

Sessions for users who connect as SYS can be fully audited, including all users connecting as SYSDBA or SYSOPER. Use the AUDIT_SYS_OPERATIONS initialization parameter to specify whether such users are to be audited. For example, the following setting specifies that SYS is to be audited:

```
AUDIT_SYS_OPERATIONS = TRUE
```

The default value, FALSE, disables SYS auditing.

All audit records for SYS are written to the operating system file that contains the audit trail, and not to SYS.AUD$ (also viewable as DBA_AUDIT_TRAIL).

- In Windows, for example, audit records are written as events to the Event Viewer log file.
- For Solaris, if the AUDIT_FILE_DEST parameter is not specified, the default location is $ORACLE_HOME/rdbms/audit.
- For other operating systems, see their audit trail documentation.

All SYS-issued SQL statements are audited indiscriminately and regardless of the setting of the AUDIT_TRAIL initialization parameter.

Consider the following SYS session:

```
CONNECT / AS SYSDBA;
ALTER SYSTEM FLUSH SHARED_POOL;
UPDATE salary SET base=1000 WHERE name='myname';
```

When SYS auditing is enabled, both the ALTER SYSTEM and UPDATE statements are displayed in the operating system audit file as follows:

```
Thu Jan 24 12:58:00 2002
ACTION: 'CONNECT'
DATABASE USER: '/'
OSPRIV: SYSDBA
CLIENT USER: jeff
CLIENT TERMINAL: pts/2
STATUS: 0

Thu Jan 24 12:58:00 2002
ACTION: 'alter system flush shared_pool'
DATABASE USER: ''
OSPRIV: SYSDBA
CLIENT USER: jeff
CLIENT TERMINAL: pts/2
STATUS: 0

Thu Jan 24 12:58:00 2002
ACTION: 'update salary set base=1000 where name='myname''
DATABASE USER: ''
OSPRIV: SYSDBA
CLIENT USER: jeff
CLIENT TERMINAL: pts/2
STATUS: 0
```

Because of the superuser privileges available to users who connect as SYSDBA, Oracle recommends that DBAs rarely use this connection and only when necessary. Normal day to day maintenance activity can usually be done by DBAs, who are regular database users with the DBA role, or a DBA role (for example, mydba or jr_dba) that your organization customizes.

## Using Triggers

You can often use triggers to record additional customized information that is not automatically included in audit records, thereby customizing your own audit conditions and record contents. For example, you could define a trigger on the EMP table to generate an audit record whenever an employee's salary is increased by more than 10 percent. You can include selected information, such as the values of SALARY before and after it was changed:

```
CREATE TRIGGER audit_emp_salaries
AFTER INSERT OR DELETE OR UPDATE ON employee_salaries
for each row
begin
if (:new.salary> :old.salary * 1.10)
      then
      insert into emp_salary_audit values (
      :employee_no,
      :old.salary,
      :new.salary,
      user,
      sysdate);
      endif;
end;
```

Furthermore, you can use event triggers to enable auditing options for specific users on login, and disable them upon logoff.

However, while Oracle triggers can readily monitor DML actions such as INSERT, UPDATE, and DELETE, monitoring on SELECT can be costly and, in some cases, uncertain. Triggers do not enable businesses to capture the statement executed as well as the result set from a query. They also do not enable users to define their own alert action in addition to simply inserting an audit record into the audit trail.

For these capabilities, use Oracle's Fine-grained Auditing, which provides an extensible auditing mechanism supporting definition of key conditions for granular audit as well as an event handler to actively alert administrators to misuse of data access rights. See Fine-Grained Auditing on page 11-29.

## Decide Whether to Use the Database or Operating System Audit Trail

The data dictionary of every Oracle database has a table named `SYS.AUD$`, commonly referred to as the database **audit trail**, and viewable as `DBA_AUDIT_TRAIL`. This table is designed to store entries auditing database statements, privileges, or schema objects.

You can optionally choose to store the database audit information to an operating system file. If your operating system has an audit trail that stores audit records generated by the operating system auditing facility, and Oracle is allowed to write to it, you can choose to direct the database audit entries to this file. For example, the Windows operating system allows Oracle to write audit records as events to the Application Event Log, viewable by the Event Viewer.

Consider the advantages and disadvantages of using either the database or operating system audit trail to store database audit records.

Using the database audit trail offers the following advantages:

- You can view selected portions of the audit trail with the predefined audit trail views of the data dictionary, such as `DBA_AUDIT_TRAIL`.
- You can use Oracle tools (such as Oracle Reports) or third-party tools to generate audit reports.

Alternatively, your operating system audit trail may allow you to consolidate audit records from multiple sources including Oracle and other applications. Therefore, examining system activity might be more efficient because all audit records are in one place. Another advantage to this approach is achieving a separation of duty between a DBA and an auditor.

**See Also:**

- Your operating system specific documentation for information about its auditing capabilities.
- Audit Trail Views on page 11-22

## What Information is Contained in the Audit Trail?

Oracle can write records to either the database audit trail, an operating system file, or both. This section describes what information the audit trail contains. asdf

- Database Audit Trail Contents
- Audit Information Stored in an Operating System File

## Database Audit Trail Contents

The database audit trail is a single table named `SYS.AUD$` in the `SYS` schema of each Oracle database's data dictionary. Several predefined views are provided to help you use the information in this table, such as DBA_AUDIT_TRAIL.

Audit trail records can contain different types of information, depending on the events audited and the auditing options set. The partial list in the following section shows columns that always appear in the audit trail: if the data they represent is available, that data populates the corresponding column. (For certain columns, this list has the column name as it displays in the audit record, shown here inside parentheses.) Certain audit columns (marked with an `*` in the following list) appear only if you have specified AUDIT_TRAIL=DB_EXTENDED in the database initialization file, init.ora. The operating system audit trail has only those columns marked (os).

- Operating system login user name (`CLIENT USER`) (os)
- Database user name (`DATABASE USER`)
- Session identifier (os)
- Terminal identifier (os)
- Name of the schema object accessed (os)
- Operation performed or attempted (`ACTION`) (os)
- Completion code of the operation (os)
- Date and time stamp in UTC (Coordinated Universal Time) format
- System privileges used (`PRIVILEGE`) (os)
- Proxy Session's auditid
- Global User unique id
- Distinguished name (os)
- Instance number
- Process number
- TransactionId
- SCN (system change number) for the SQL statement
- (*) SQL text (the SQL text that triggered the auditing)
- (*) Bind values used for the SQL statement, if any

If the database destination for audit records becomes full or unavailable and therefore unable to accept new records, an audited action cannot complete. Instead, it causes an error message and is not done. In some cases, an operating system log allows such an action to complete.

The audit trail does not store information about any data values that might be involved in the audited statement. For example, old and new data values of updated rows are not stored when an `UPDATE` statement is audited. However, this specialized type of auditing can be performed using fine-grained auditing methods.

There is a new audit trail view that combines standard and fine-grained audit log records, named DBA_COMMON_AUDIT_TRAIL.

You can use the Flashback Query feature to show the old and new values of the updated rows, subject to any auditing policy presently in force. The current policies are enforced even if the flashback is to an old query that was originally subject to a different policy. Current business access rules always apply.

**See Also:**

- "Fine-Grained Auditing" on page 11-29 for more information about methods of fine-grained auditing
- "Flashback Queries" in *Oracle Database Administrator's Guide*

## Audit Information Stored in an Operating System File

The operating system file that contains the audit trail can contain any of the following:

- Audit records generated by the operating system
- Database audit trail records
- Database actions that are always audited
- Audit records for administrative users (`SYS`)

Audit trail records written to an operating system audit trail may contain encoded information, but this information can be decoded using data dictionary tables and error messages as follows:

| Encoded Information | How to Decode |
|---|---|
| Action code | Describes the operation performed or attempted. The `AUDIT_ACTIONS` data dictionary table contains a list of these codes and their descriptions. |

| Encoded Information | How to Decode |
|---|---|
| Privileges used | Describes any system privileges used to perform the operation. The `SYSTEM_PRIVILEGE_MAP` table lists all of these codes and their descriptions. |
| Completion code | Describes the result of the attempted operation. Successful operations return a value of zero; unsuccessful operations return the Oracle error code describing why the operation was unsuccessful. These codes are listed in *Oracle Database Error Messages*. |

## Managing the Standard Audit Trail

This section describes various aspects of managing standard audit trail information, and contains the following topics:

- Enabling and Disabling Standard Auditing
- Standard Auditing in a Multitier Environment
- Setting Standard Auditing Options
- Turning Off Standard Audit Options
- Controlling the Growth and Size of the Standard Audit Trail
- Protecting the Standard Audit Trail
- Auditing the Standard Audit Trail

### Enabling and Disabling Standard Auditing

Any authorized database user can set statement, privilege, and object auditing options at any time, but Oracle does not generate audit information for the standard database audit trail unless database auditing is enabled. The security administrator is normally responsible for controlling auditing.

This section discusses the initialization parameters that enable and disable standard auditing.

**Note:**

- the initialization parameters AUDIT_SYS_OPERATIONS and AUDIT_TRAIL affecting standard auditing are static. "Static" means that if you change their values, you must shut down and restart your database for the new values to take effect.
- The AUDIT_FILE_DEST initialization parameter can be changed with "Alter System set AUDIT_FILE_DEST = <dir> DEFERRED", meaning the new destination will be effective for all subsequent sessions.

### Setting the AUDIT_TRAIL Initialization Parameter

Database auditing is enabled and disabled by the AUDIT_TRAIL initialization parameter in the database's initialization parameter file. The parameter can be set to the following values:

| Parameter Value | Meaning |
|---|---|
| DB | Enables database auditing and directs all audit records to the database audit trail (SYS.AUD$), except for records that are always written to the operating system audit trail |
| DB_EXTENDED | Does all actions of AUDIT_TRAIL=DB and also populates the SQL bind and SQL text CLOB-type columns of the SYS.AUD$ table, wherever possible. (These columns are the ones referred to as the additional eight, populated only when this parameter is specified.) |
| OS | Enables database auditing and directs all audit records to an operating system file |
| NONE | Disables standard auditing (This value is the default.) |

Note that changes that alter what objects are audited do not require restarting the database, which is only required if a universal change is made, such as turning on or off *all* auditing.

**Note:** You do not need to set AUDIT_TRAIL to enable either fine-grained auditing or SYS auditing. For fine-grained auditing, you simply add and remove FGA policies as you see fit, applying them to the specific operations or objects you want to monitor. For SYS auditing, you just set the SYS audit parameter for SYS audit.

See the section titled Fine-Grained Auditing later in this chapter.

### Setting the AUDIT_FILE_DEST Initialization Parameter

The `AUDIT_FILE_DEST` initialization parameter specifies an operating system directory into which the audit trail is written when `AUDIT_TRAIL=OS` is specified. It is also the location to which mandatory auditing information is written and, if so specified by the `AUDIT_SYS_OPERATIONS` initialization parameter, audit records for user `SYS`. `AUDIT_FILE_DEST` can be changed with "Alter System set AUDIT_FILE_DEST = <dir> DEFERRED", meaning the new destination will be effective for all subsequent sessions.

If the `AUDIT_FILE_DEST` parameter is not specified, the default location on Solaris is `$ORACLE_HOME/rdbms/audit`.

In Windows, the default location to which audit records are written is the Event Viewer log file.

**Notes:**

- If your operating system supports an audit trail, then its location is operating system specific. For example, the Windows operating systems writes audit records as events to the application event log. You can view and manage these events using Event Viewer. You are not allowed to specify the `AUDIT_FILE_DEST` initialization parameter for Windows platforms. For more information, see *Oracle Database Platform Guide for Windows.*
- Some operating systems always log an audit record for instance connection and database startup to the default location `$ORACLE_HOME/rdbms/audit` regardless of the setting for `AUDIT_FILE_DEST`. This log action occurs because the parameter setting is not known until the database is mounted.

## Standard Auditing in a Multitier Environment

In a multitier environment, Oracle preserves the identity of the client through all tiers, which enables auditing of actions taken on behalf of the client. To do such auditing, you use the `BY` *`proxy`* clause in your `AUDIT` statement.

This clause allows you a few options. You can:

- Audit SQL statements issued by the specified proxy on its own behalf
- Audit statements executed on behalf of a specified user or users
- Audit all statements executed on behalf of any user

The following example audits `SELECT TABLE` statements issued on behalf of client `jackson` by the proxy application server `appserve`.

```
AUDIT SELECT TABLE
    BY appserve ON BEHALF OF jackson;
```

**See Also:** *Oracle Database Concepts* and *Oracle Database Application Developer's Guide - Fundamentals* for more information on proxies and multitier applications

## Setting Standard Auditing Options

You specify one of the three standard auditing options using the `AUDIT` statement:

| **Level** | **Effect** |
|---|---|
| Statement | Causes auditing of specific SQL statements or groups of statements that affect a particular type of database object. For example, `AUDIT TABLE` audits the `CREATE TABLE`, `TRUNCATE TABLE`, `COMMENT ON TABLE`, and `DELETE [FROM] TABLE` statements. |
| Privilege | Audits SQL statements that are authorized by the specified system privilege. For Example, `AUDIT CREATE ANY TRIGGER` audits statements issued using the `CREATE ANY TRIGGER` system privilege. |
| Object | Audits specific statements on specific objects, such as `ALTER TABLE` on the `emp` table |

To use the `AUDIT` statement to set statement and privilege options, you must have the `AUDIT SYSTEM` privilege. To use it to set object audit options, you must own the object to be audited or have the `AUDIT ANY` privilege.

Audit statements that set statement and privilege audit options can include a `BY` clause to specify a list of users or application proxies to limit the scope of the statement and privilege audit options.

When setting auditing options, you can also specify the following conditions for auditing:

- `BY SESSION`/`BY ACCESS`

  `BY SESSION` causes Oracle to write a single record for all SQL statements of the same type issued in the same session. `BY ACCESS` causes Oracle to write one record for each access.

  > **Note:** If you are using an operating system file for the audit trail (`AUDIT_TRAIL=OS`), multiple records may still be written to the audit trail when `BY SESSION` is specified. This occurs because while Oracle can write to the operating system file, it is unable to read it to detect that it has already written an audit entry for the action.

- `WHENEVER SUCCESSFUL`/`WHENEVER NOT SUCCESSFUL`

  `WHENEVER SUCCESSFUL` chooses auditing only for statements that succeed. `WHENEVER NOT SUCCESSFUL` chooses auditing only for statements that fail or result in errors.

The implications of your choice of auditing option and specification of `AUDIT` statement clauses is discussed in subsequent sections.

A new database session picks up auditing options from the data dictionary when the session is created. These auditing options remain in force for the duration of the database connection. Setting new system or object auditing options causes all subsequent database sessions to use these options; existing sessions continue using the audit options in place at session creation.

> **Caution:** The `AUDIT` statement only specifies auditing options; it does not enable auditing as a whole. To turn auditing on and control whether Oracle generates audit records based on the audit options currently set, set the initialization parameter `AUDIT_TRAIL` as described in "Enabling and Disabling Standard Auditing" on page 11-10.

**See Also:** *Oracle Database SQL Reference* for a complete description of the AUDIT statement

### Specifying Statement Auditing

Valid statement audit options that can be included in AUDIT and NOAUDIT statements are listed in the *Oracle Database SQL Reference*.

Two special cases of statement auditing are discussed in the following sections.

**Auditing Connections and Disconnections** The SESSION statement option is unique because it does not generate an audit record when a particular type of statement is issued; this option generates a single audit record for each session created by connections to an instance. An audit record is inserted into the audit trail at connect time and updated at disconnect time. Cumulative information about a session is stored in a single audit record that corresponds to the session. This record can include connection time, disconnection time, and logical and physical I/Os processed, among other information.

To audit all successful and unsuccessful connections to and disconnections from the database, regardless of user, BY SESSION (the default and only value for this option), enter the following statement:

```
AUDIT SESSION;
```

You can set this option selectively for individual users also, as in the next example:

```
AUDIT SESSION
BY jeff, lori;
```

**Auditing Statements That Fail Because an Object Does Not Exist** The NOT EXISTS statement option specifies auditing of all SQL statements that fail because the target object does not exist.

### Specifying Privilege Auditing

Privilege audit options exactly match the corresponding system privileges. For example, the option to audit use of the DELETE ANY TABLE privilege is DELETE ANY TABLE. To turn this option on, you use a statement similar to the following example:

```
AUDIT DELETE ANY TABLE
    BY ACCESS
    WHENEVER NOT SUCCESSFUL;
```

Oracle's system privileges are listed in the *Oracle Database SQL Reference*.

To audit all successful and unsuccessful uses of the DELETE ANY TABLE system privilege, enter the following statement:

```
AUDIT DELETE ANY TABLE;
```

To audit all unsuccessful SELECT, INSERT, and DELETE statements on all tables and unsuccessful uses of the EXECUTE PROCEDURE system privilege, by all database users, and by individual audited statement, issue the following statement:

```
AUDIT SELECT TABLE, INSERT TABLE, DELETE TABLE, EXECUTE PROCEDURE
      BY ACCESS
      WHENEVER NOT SUCCESSFUL;
```

The AUDIT SYSTEM system privilege is required to set any statement or privilege audit option. Normally, the security administrator is the only user granted this system privilege.

## Specifying Object Auditing

The *Oracle Database SQL Reference* lists valid object audit options and the schema object types for which each option is available.

A user can set any object audit option for the objects contained in the user's own schema. The AUDIT ANY system privilege is required to set an object audit option for an object contained in another user's schema or to set the default object auditing option. Normally, the security administrator is the only user granted the AUDIT ANY privilege.

To audit all successful and unsuccessful DELETE statements on the jeff.emp table, BY SESSION (the default value), enter the following statement:

```
AUDIT DELETE ON jeff.emp;
```

To audit all successful SELECT, INSERT, and DELETE statements on the dept table owned by user jward, BY ACCESS, enter the following statement:

```
AUDIT SELECT, INSERT, DELETE
     ON jward.dept
     BY ACCESS
     WHENEVER SUCCESSFUL;
```

To set the default object auditing options to audit all unsuccessful SELECT statements, BY SESSION (the default), enter the following statement:

```
AUDIT SELECT
```

```
    ON DEFAULT
    WHENEVER NOT SUCCESSFUL;
```

## Turning Off Standard Audit Options

The NOAUDIT statement turns off the various audit options of Oracle Database 10*g*. Use it to reset statement and privilege audit options, and object audit options. A NOAUDIT statement that sets statement and privilege audit options can include the BY *user* or BY *proxy* option to specify a list of users to limit the scope of the statement and privilege audit options.

You can use a NOAUDIT statement to disable an audit option selectively using the WHENEVER clause. If the clause is not specified, the auditing option is disabled entirely, for both successful and unsuccessful cases.

The BY SESSION/BY ACCESS option pair is *not* supported by the NOAUDIT statement; audit options, no matter how they were turned on, are turned off by an appropriate NOAUDIT statement.

**Caution:** The NOAUDIT statement only specifies auditing options; it does not disable auditing as a whole. To turn auditing off and stop Oracle from generating audit records, set the initialization parameter AUDIT_TRAIL in the database's initialization parameter file as described in "Enabling and Disabling Standard Auditing" on page 11-10.

**See Also:** *Oracle Database SQL Reference* for a complete syntax listing of the NOAUDIT statement

### Turning Off Statement and Privilege Auditing

The following statements turn off the corresponding audit options:

```
NOAUDIT session;
NOAUDIT session BY jeff, lori;
NOAUDIT DELETE ANY TABLE;
NOAUDIT SELECT TABLE, INSERT TABLE, DELETE TABLE,
    EXECUTE PROCEDURE;
```

The following statement turns off all statement audit options:

```
NOAUDIT ALL;
```

The following statement turns off all privilege audit options:

```
NOAUDIT ALL PRIVILEGES;
```

To disable statement or privilege auditing options, you must have the AUDIT SYSTEM system privilege.

### Turning Off Object Auditing

The following statements turn off the corresponding auditing options:

```
NOAUDIT DELETE
   ON emp;
NOAUDIT SELECT, INSERT, DELETE
   ON jward.dept;
```

Furthermore, to turn off all object audit options on the emp table, enter the following statement:

```
NOAUDIT ALL
   ON emp;
```

To turn off all default object audit options, enter the following statement:

```
NOAUDIT ALL
   ON DEFAULT;
```

All schema objects created before this NOAUDIT statement is issued continue to use the default object audit options in effect at the time of their creation, unless overridden by an explicit NOAUDIT statement after their creation.

To disable object audit options for a specific object, you must be the owner of the schema object. To disable the object audit options of an object in another user's schema or to disable default object audit options, you must have the AUDIT ANY system privilege. A user with privileges to disable object audit options of an object can override the options set by any user.

## Controlling the Growth and Size of the Standard Audit Trail

If the audit trail becomes completely full and no more audit records can be inserted, audited statements cannot be successfully executed until the audit trail is purged. Warnings are returned to all users that issue audited statements. Therefore, the security administrator must control the growth and size of the audit trail.

When auditing is enabled and audit records are being generated, the audit trail grows according to two factors:

- The number of audit options turned on
- The frequency of execution of audited statements

To control the growth of the audit trail, you can use the following methods:

- Enable and disable database auditing. If it is enabled, audit records are generated and stored in the audit trail; if it is disabled, audit records are not generated.
- Be very selective about the audit options that are turned on. If more selective auditing is performed, useless or unnecessary audit information is not generated and stored in the audit trail.
- Tightly control the ability to perform object auditing. This can be done two different ways:
  - A security administrator owns all objects and the `AUDIT ANY` system privilege is never granted to any other user. Alternatively, all schema objects can belong to a schema for which the corresponding user does not have `CREATE SESSION` privilege.
  - All objects are contained in schemas that do not correspond to real database users (that is, the `CREATE SESSION` privilege is not granted to the corresponding user) and the security administrator is the only user granted the `AUDIT ANY` system privilege.

  In both scenarios, object auditing is controlled entirely by the security administrator.

The maximum size of the database audit trail (`SYS.AUD$` table) is determined by the default storage parameters of the `SYSTEM` tablespace, in which it is stored.

> **See Also:** Your operating system specific Oracle documentation for more information about managing the operating system audit trail when you are directing audit records to that location

## Purging Audit Records from the Audit Trail

After auditing is enabled for some time, the security administrator may want to delete records from the database audit trail both to free audit trail space and to facilitate audit trail management.

For example, to delete *all* audit records from the audit trail, enter the following statement:

```
DELETE FROM SYS.AUD$;
```

Alternatively, to delete all audit records from the audit trail generated as a result of auditing the table `emp`, enter the following statement:

```
DELETE FROM SYS.AUD$
    WHERE obj$name='EMP';
```

---

**Note:** All deletes from the audit trail are audited without exception: see this chapter's sections entitled Auditing the Standard Audit Trail on page 11-21 and Auditing Administrative Users on page 11-4.

---

Only the user `SYS`, a user who has the `DELETE ANY TABLE` privilege, or a user to whom `SYS` has granted `DELETE` privilege on `SYS.AUD$` can delete records from the database audit trail.

---

**Note:** If the audit trail is completely full and connections are being audited (that is, if the `SESSION` option is set), typical users cannot connect to the database because the associated audit record for the connection cannot be inserted into the audit trail. In this case, the security administrator must connect as `SYS` (operations by `SYS` are not audited) and make space available in the audit trail.

---

**See Also:** *Oracle Database Utilities* for information about exporting tables

### Archiving Audit Trail Information

If audit trail information must be archived for historical purposes, the security administrator can copy the relevant records to a normal database table (for example, using `INSERT INTO` *`table`* `SELECT ... FROM SYS.AUD$ ...`) or export the audit trail table to an operating system file.

### Reducing the Size of the Audit Trail

As with any database table, after records are deleted from the database audit trail, the extents allocated for this table still exist.

If the database audit trail has many extents allocated for it, but many of them are not being used, the space allocated to the database audit trail can be reduced by following these steps:

1. If you want to save information currently in the audit trail, copy it to another database table or export it using the EXPORT utility.
2. Connect as a user with administrator privileges.
3. Truncate SYS.AUD$ using the TRUNCATE statement.
4. Reload archived audit trail records generated from Step 1.

The new version of SYS.AUD$ is allocated only as many extents as are necessary to contain current audit trail records.

**Note:** SYS.AUD$ is the only SYS object that should ever be directly modified.

## Protecting the Standard Audit Trail

When auditing for suspicious database activity, protect the integrity of the audit trail's records to guarantee the accuracy and completeness of the auditing information.

Audit records generated as a result of object audit options set for the SYS.AUD$ table can only be deleted from the audit trail by someone connected with administrator privileges, which itself has protection against unauthorized use.

## Auditing the Standard Audit Trail

If an application needs to give SYS.AUD$ access to regular users (non-SYSDBA users), then such access needs to be audited.

To do so, you turn on the relevant auditing options for SYS.AUD$, which work a little differently because they are auditing actions on the audit trail(aud$) itself:

1. Connect sys/passw as SYSDBA.
2. Issue the following command:

   ```
   AUDIT SELECT, INSERT, UPDATE, DELETE ON sys.aud$ BY ACCESS;
   ```

Please note that this command will AUDIT actions performed by non-SYSDBA users only.

Then if a regular user has select, update, insert and delete privileges on SYS.AUD$ and executes a SELECT operation, the audit trail will have a record of that operation. That is, SYS.AUD$ will have a row identifying the SELECT action on itself, as say row1.

If a user later tries to DELETE this row1 from SYS.AUD$, the DELETE will succeed, since the user has the privilege to perform this action. However, this DELETE action on SYS.AUD$ is also recorded in the audit trail.

Setting up this type of auditing acts as a safety feature, potentially revealing unusual or unauthorized actions.

A logfile for an illustrative test case appears at the end of this chapter, at Example of Auditing Table SYS.AUD$.

## Viewing Database Audit Trail Information

The database audit trail (SYS.AUD$) is a single table in each Oracle database's data dictionary. Several predefined views are available to present auditing information from this table in a meaningful way. If you decide not to use auditing, you can later delete these views. The following subsections show you what's in these views, how to use them, and how to delete them:

- Audit Trail Views
- Using Audit Trail Views to Investigate Suspicious Activities
- Deleting the Audit Trail Views

### Audit Trail Views

The following views are created upon installation:

| View | Description |
|---|---|
| STMT_AUDIT_OPTION_MAP | Contains information about auditing option type codes. Created by the SQL.BSQ script at CREATE DATABASE time. |
| AUDIT_ACTIONS | Contains descriptions for audit trail action type codes |
| ALL_DEF_AUDIT_OPTS | Contains default object-auditing options that will be applied when objects are created |
| DBA_STMT_AUDIT_OPTS | Describes current system auditing options across the system and by user |
| DBA_PRIV_AUDIT_OPTS | Describes current system privileges being audited across the system and by user |

| View | Description |
|---|---|
| DBA_OBJ_AUDIT_OPTS<br>USER_OBJ_AUDIT_OPTS | Describes auditing options on all objects. USER view describes auditing options on all objects owned by the current user. |
| DBA_AUDIT_TRAIL<br>USER_AUDIT_TRAIL | Lists all audit trail entries. USER view shows audit trail entries relating to current user. |
| DBA_AUDIT_OBJECT<br>USER_AUDIT_OBJECT | Contains audit trail records for all objects in the system. USER view lists audit trail records for statements concerning objects that are accessible to the current user. |
| DBA_AUDIT_SESSION<br>USER_AUDIT_SESSION | Lists all audit trail records concerning CONNECT and DISCONNECT. USER view lists all audit trail records concerning connections and disconnections for the current user. |
| DBA_AUDIT_STATEMENT<br>USER_AUDIT_STATEMENT | Lists audit trail records concerning GRANT, REVOKE, AUDIT, NOAUDIT, and ALTER SYSTEM statements throughout the database, or for the USER view, issued by the user |
| DBA_AUDIT_EXISTS | Lists audit trail entries produced BY AUDIT NOT EXISTS |
| DBA_AUDIT_POLICIES | Shows all the auditing policies on the system. |
| DBA_FGA_AUDIT_TRAIL | Lists audit trail records for value-based auditing. |
| DBA_COMMON_AUDIT_<br>TRAIL | Combines standard and fine-grained audit log records |

**See Also:** *Oracle Database Reference* for more detailed descriptions of the Oracle provided predefined views

## Using Audit Trail Views to Investigate Suspicious Activities

This section offers examples that demonstrate how to examine and interpret the information in the audit trail. Consider the following situation.

You would like to audit the database for the following suspicious activities:

- Passwords, tablespace settings, and quotas for some database users are being altered without authorization.
- A high number of deadlocks are occurring, most likely because of users acquiring exclusive table locks.
- Rows are arbitrarily being deleted from the `emp` table in `jeff`'s schema.

You suspect the users `jward` and `swilliams` of several of these detrimental actions.

To enable your investigation, you issue the following statements (in order):

```
AUDIT ALTER, INDEX, RENAME ON DEFAULT
    BY SESSION;
CREATE VIEW jeff.employee AS SELECT * FROM jeff.emp;
AUDIT SESSION BY jward, swilliams;
AUDIT ALTER USER;
AUDIT LOCK TABLE
    BY ACCESS
    WHENEVER SUCCESSFUL;
AUDIT DELETE ON jeff.emp
    BY ACCESS
    WHENEVER SUCCESSFUL;
```

The following statements are subsequently issued by the user `jward`:

```
ALTER USER tsmith QUOTA 0 ON users;
DROP USER djones;
```

The following statements are subsequently issued by the user `swilliams`:

```
LOCK TABLE jeff.emp IN EXCLUSIVE MODE;
DELETE FROM jeff.emp WHERE mgr = 7698;
ALTER TABLE jeff.emp ALLOCATE EXTENT (SIZE 100K);
CREATE INDEX jeff.ename_index ON jeff.emp (ename);
CREATE PROCEDURE jeff.fire_employee (empid NUMBER) AS
  BEGIN
    DELETE FROM jeff.emp WHERE empno = empid;
  END;
/

EXECUTE jeff.fire_employee(7902);
```

The following sections display the information relevant to your investigation that can be viewed using the audit trail views in the data dictionary:

- Listing Active Statement Audit Options
- Listing Active Privilege Audit Options
- Listing Active Object Audit Options for Specific Objects
- Listing Default Object Audit Options
- Listing Audit Records
- Listing Audit Records for the AUDIT SESSION Option

### Listing Active Statement Audit Options

The following query returns all the statement audit options that are set:

```
SELECT * FROM DBA_STMT_AUDIT_OPTS;

USER_NAME               AUDIT_OPTION        SUCCESS         FAILURE
--------------------    -------------------  ----------      ---------
JWARD                   SESSION             BY SESSION      BY SESSION
SWILLIAMS               SESSION             BY SESSION      BY SESSION
                        LOCK TABLE          BY ACCESS       NOT SET
```

Notice that the view reveals the statement audit options set, whether they are set for success or failure (or both), and whether they are set for BY SESSION or BY ACCESS.

### Listing Active Privilege Audit Options

The following query returns all the privilege audit options that are set:

```
SELECT * FROM DBA_PRIV_AUDIT_OPTS;

USER_NAME           PRIVILEGE           SUCCESS      FAILURE
------------------- -------------------- ---------   ----------
ALTER USER          BY SESSION          BY SESSION
```

### Listing Active Object Audit Options for Specific Objects

The following query returns all audit options set for any objects whose name starts with the characters emp and which are contained in jeff's schema:

```
SELECT * FROM DBA_OBJ_AUDIT_OPTS
    WHERE OWNER = 'JEFF' AND OBJECT_NAME LIKE 'EMP%';

OWNER OBJECT_NAME OBJECT_TY ALT AUD COM DEL GRA IND INS LOC ...
----- ----------- --------- --- --- --- --- --- --- --- --- ...
JEFF EMP         TABLE     S/S -/- -/- A/- -/- S/S -/- -/- ...
JEFF EMPLOYEE    VIEW      -/- -/- -/- A/- -/- S/S -/- -/- ...
```

Notice that the view returns information about all the audit options for the specified object. The information in the view is interpreted as follows:

- The character "-" indicates that the audit option is not set.
- The character "S" indicates that the audit option is set, BY SESSION.
- The character "A" indicates that the audit option is set, BY ACCESS.

- Each audit option has two possible settings, WHENEVER SUCCESSFUL and WHENEVER NOT SUCCESSFUL, separated by "/". For example, the DELETE audit option for jeff.emp is set BY ACCESS for successful delete statements and not set at all for unsuccessful delete statements.

### Listing Default Object Audit Options

The following query returns all default object audit options:

```
SELECT * FROM ALL_DEF_AUDIT_OPTS;

ALT AUD COM DEL GRA IND INS LOC REN SEL UPD REF EXE FBK
--- --- --- --- --- --- --- --- --- --- --- --- --- ---
S/S -/- -/- -/- -/- S/S -/- -/- S/S -/- -/- -/- -/- /-
```

Notice that the view returns information similar to the USER_OBJ_AUDIT_OPTS and DBA_OBJ_AUDIT_OPTS views (see previous example).

### Listing Audit Records

The following query lists audit records generated by statement and object audit options:

```
    SELECT * FROM DBA_AUDIT_OBJECT;
```

### Listing Audit Records for the AUDIT SESSION Option

The following query lists audit information corresponding to the AUDIT SESSION statement audit option:

```
SELECT USERNAME, LOGOFF_TIME, LOGOFF_LREAD, LOGOFF_PREAD,
    LOGOFF_LWRITE, LOGOFF_DLOCK
    FROM DBA_AUDIT_SESSION;

USERNAME   LOGOFF_TI LOGOFF_LRE LOGOFF_PRE LOGOFF_LWR LOGOFF_DLO
---------- --------- ---------- ---------- ---------- ----------
JWARD      02-AUG-91         53          2         24          0
SWILLIAMS  02-AUG-91       3337        256        630          0
```

## Deleting the Audit Trail Views

If you disable auditing and no longer need the audit trail views, delete them by connecting to the database as SYS and running the script file CATNOAUD.SQL. The name and location of the CATNOAUD.SQL script are operating system dependent.

## Example of Auditing Table SYS.AUD$

The code in this section illustrates the auditing of changes made to SYS.AUD$.

```
SQL> @t
SQL>
SQL> SET FEEDBACK 1
SQL> SET NUMWIDTH 10
SQL> SET LINESIZE 80
SQL> SET TRIMSPOOL ON
SQL> SET TAB OFF
SQL> SET PAGESIZE 100
SQL>
SQL> column username format a10
SQL> column owner  format a10
SQL> column obj_name format a6
SQL> column action_name format a17
SQL> SET ECHO ON
SQL>
SQL> connect sys/newdbapassword as sysdba
Connected.
SQL> grant select, insert, update, delete on sys.aud$ to jeff;

Grant succeeded.

SQL> grant select on dba_audit_trail to jeff;

Grant succeeded.

SQL> audit select, update, delete on sys.aud$ by access;

Audit succeeded.

SQL> truncate table sys.aud$;

Table truncated.

SQL>
SQL> connect jeff/wolf
Connected.
SQL> select count(*) from emp

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

```
         0

1 row selected.

SQL>
SQL> select statementid,entryid,username,action_name,returncode,owner,
  2  obj_name,substr(priv_used,1,8) priv,  SES_ACTIONS
  3  from dba_audit_trail
  4  order by sessionid,entryid;

STATEMENTID    ENTRYID USERNAME   ACTION_NAME       RETURNCODE OWNER      OBJ_NA
----------- ---------- ---------- ----------------- ---------- ---------- ------
PRIV     SES_ACTIONS
-------- -------------------
          8          1 JEFF      SELECT                     0 SYS        AUD$


1 row selected.

SQL>
SQL> update sys.aud$ set userid = 0;

2 rows updated.

SQL> select statementid,entryid,username,action_name,returncode,owner,
  2  obj_name,substr(priv_used,1,8) priv,  SES_ACTIONS
  3  from dba_audit_trail
  4  order by sessionid,entryid;

STATEMENTID    ENTRYID USERNAME   ACTION_NAME       RETURNCODE OWNER      OBJ_NA
----------- ---------- ---------- ----------------- ---------- ---------- ------
PRIV     SES_ACTIONS
-------- -------------------
          8          1 0          SELECT                     0 SYS        AUD$

          9          2 0          SELECT                     0 SYS        AUD$

         10          3 JEFF       UPDATE                     0 SYS        AUD$

3 rows selected.

SQL>
SQL> delete from sys.aud$;

3 rows deleted.
```

```
SQL> select statementid,entryid,username,action_name,returncode,owner,
  2  obj_name,substr(priv_used,1,8) priv,  SES_ACTIONS
  3  from dba_audit_trail
  4  order by sessionid,entryid;

STATEMENTID    ENTRYID USERNAME   ACTION_NAME       RETURNCODE OWNER      OBJ_NA
----------- ---------- ---------- ----------------- ---------- ---------- ------
PRIV     SES_ACTIONS
-------- -------------------
         10          3 JEFF      UPDATE                      0 SYS        AUD$

         12          5 JEFF      DELETE                      0 SYS        AUD$

2 rows selected.

SQL>
SQL> connect sys/newdbapassword as sysdba
Connected.
SQL> noaudit insert, select, update, delete on sys.aud$;

Noaudit succeeded.

SQL>
SQL> spool off
```

## Fine-Grained Auditing

As described earlier in this chapter and in Chapter 8, standard Oracle auditing  is highly configurable. Its audit trail provides a fixed set of facts that monitor privileges, object access, or (optionally) SQL usage, including information about the environment or query results. The scope of standard auditing can be substantially expanded by using triggers, providing additional customized information.

However, there is no mechanism to specify audit conditions so as to minimize unhelpful audits, and reconstructing events from access logs often fails to prove access rights were violated.

Oracle's Fine-Grained Auditing addresses these needs, taking you beyond standard auditing and enabling you to minimize false or unhelpful audits by specifying more detailed audit conditions. You do not need to set `AUDIT_TRAIL` to enable fine-grained auditing. You simply add and remove FGA policies as you see fit, applying them to the specific operations or objects you want to monitor. A built-in audit mechanism in the database prevents users from bypassing the audit.

Fine-grained auditing records are stored in the DBA_FGA_AUDIT_TRAIL view, and also in the DBA_COMMON_AUDIT_TRAIL view, which combines standard and fine-grained audit log records.

**See Also:** To add, drop, enable, or disable policies, you use the package described later in this chapter: The DBMS_FGA Package

## Policies in Fine-Grained Auditing

Policies you establish with fine-grained auditing can monitor data access based on content. Using policies, you can establish what columns and conditions you want audit records for. Your conditions can include limiting the audit to specific types of DML statements used in connection with the columns you specify. You can also provide the name of the routine you want called when an audit event occurs, to notify or alert administrators or to handle errors or anomalies.

For example, most companies logically want to limit access to the specifications for a product under development, or its test results, and prefer that salary information remain private. Auditors want enough detail to be able to determine what data was accessed. Knowing only that SELECT privilege was used by a specific user on a particular table is not specific enough to provide accountability.

A central tax authority has similar privacy concerns, needing to track access to tax returns so that employees don't snoop. Similarly, a government agency needs detailed tracking of access to its database of informants. Such agencies also need enough detail to determine what data was accessed, not simply that the SELECT privilege was used by JEFF on the TAXPAYERS or INFORMANTS table.

### Advantages of Fine-Grained Auditing over Triggers

Fine-grained auditing meets these needs by providing functionality (and efficiency) beyond what triggers can do. Triggers incur a PL/SQL process call for every row processed, and create an audit record only when a relevant column is changed by a DML statement.

An FGA policy, on the other hand, does not incur this cost for every row. Instead, it audits only once for every policy. Specifically, it audits when a specified relevant column occurs in a specified type of DML statement, either being changed by the statement or being in its selection criteria. This combination of criteria uncovers users who hope their information gathering will be masked because they only use the selection criteria of a DML statement. Triggers also cannot monitor the activity of another "instead-of" trigger on the same object, while fine-grained auditing supports tables and views.

### Extensible Interface Using Event Handler Functions

Organizations can thus use fine-grained auditing to define policies specifying the data access conditions that are to trigger audit events. These policies can use flexible event handlers that notify administrators when a triggering event has occurred. For example, an organization may allow HR clerks to access employee salary information, but trigger an audit event when salaries are greater than $500K are accessed. The audit policy (where SALARY > 500000) is applied to the EMPLOYEES table through an audit policy interface (DBMS_FGA, a PL/SQL package).

The audit function (handler_module) is an alerting mechanism for the administrator. The required interface for such a function is as follows:

```
PROCEDURE <fname> ( object_schema VARCHAR2, object_name VARCHAR2, policy_
name VARCHAR2 )  AS ...
```

where fname is the name of the procedure, object_schema is the name of the schema of the table audited, object_name is the name of the table to be audited, and policy_name is the name of the policy being enforced.

### Functions and Relevant Columns in Fine-Grained Auditing

For additional flexibility in implementation, organizations can employ a user-defined function to determine the policy condition, and identify an audit column (called a *relevant column*) to further refine the audit policy. For example, the function could cause an audit record only when a salary greater than $250,000 is accessed.

Specifying a relevant column helps reduce the instances of false or unnecessary audit records, because the audit need only be triggered when a particular column is referenced in the query. For example, an organization may only wish to audit executive salary access when an employee name is accessed, because accessing salary information alone is not meaningful unless an HR clerk also selects the corresponding employee name. You can, however, specify that auditing occur only when all relevant columns are referenced.

If more than one relevant audit column is specified, Oracle produces an audit record if the SQL statement references any of those audit columns.

The DBMS_FGA package administers these value-based audit policies. The security administrator creates an audit policy on the target object using the functions in the DBMS_FGA package.

**See also:** The DBMS_FGA Package (the next major section)

### Audit Records in Fine-Grained Auditing

If any rows returned from a query block match the audit condition, then an audit event entry is inserted into the fine-grained audit trail. This entry includes username, SQL text, bind variable, policy name, session ID, time stamp, and other attributes. Only one row of audit information is inserted into the audit trail for every FGA policy that evaluates to TRUE. As part of the extensibility framework, administrators can also optionally define an appropriate **audit event handler** to process the event, for example sending an alert page to the administrator.

### NULL Audit Conditions

To guarantee auditing of the specified actions ("statement_types") affecting the specified columns ("audit_column"), specify the audit_condition as NULL (or omit it), which is interpreted as TRUE. Only specifying NULL will guarantee auditing of the specified actions ("statement_types") affecting the specified columns ("audit_ column"). The former practice of specifying an audit condition of "1=1" to force such auditing should no longer be used and will not reliably achieve the desired result. NULL will cause audit even if no rows were processed, so that all actions on an audit_column with this policy are audited.

**Note:** Using an empty string is not equivalent to NULL and will not reliably cause auditing of all actions on a table with this policy.

The audit function is executed as an autonomous transaction, committing only the actions of the handler_module and not any user transaction. This function has no effect on any user SQL transaction.

If NULL or no audit condition is specified, then any action on a table with that policy causes an audit record to be created, whether or not rows are returned.

### Defining FGA Policies

The administrator uses the `DBMS_FGA.ADD_POLICY` interface to define each FGA policy for a table or view, identifying any combination of *select, update, delete,* or *insert* statements. Oracle supports `MERGE` statements as well, by auditing the underlying actions of `INSERT` and `UPDATE`. To audit `MERGE`s, set up FGA on these `INSERT`s and `UPDATE`s. Only one record is generated, for each policy, for successful `MERGE`s.

FGA policies associated with a table or view may also specify *relevant columns*, so that any specified statement type affecting a particular column is audited. More than one column can be included as relevant columns in a single FGA policy.

Examples include privacy-relevant columns, such as those containing social security numbers, salaries, patient diagnoses, and so on. If no *relevant column* is specified, auditing applies to all columns. That is, auditing occurs whenever any specified statement type affects any column, unless you specify in the policy that auditing is to occur only when all relevant columns are referenced.

## An Added Benefit to Fine-Grained Auditing

In general, fine-grained auditing policies are based on simple user-defined SQL predicates on table objects as conditions for selective auditing. During fetching, whenever policy conditions are met for a returning row, the query is audited. Later, Oracle can execute a user-defined audit event handler, if specified in the policy, using autonomous transactions to process the event.

Fine-grained auditing can be implemented in user applications using the `DBMS_FGA` package or by using database triggers.

The following example shows how you can audit statements (`INSERT`, `UPDATE`, `DELETE`, and `SELECT`) on table *hr.emp* to monitor any query that accesses the *salary* column of the employee records which belong to *sales* department:

```
DBMS_FGA.ADD_POLICY(
object_schema => 'hr',
object_name   => 'emp',
policy_name   => 'chk_hr_emp',
audit_condition => 'dept = ''SALES'' ',
audit_column => 'salary'
statement_types => 'insert,update,delete,select');
```

Then, any of the following SQL statements will cause the database to log an audit event record.

```
SELECT count(*) FROM hr.emp WHERE dept = 'SALES' and salary > 10000000;

SELECT salary FROM hr.emp WHERE dept = 'SALES';

DELETE from hr.emp where salary >1000000

With all the relevant information available, and a trigger-like mechanism to
use, the administrator can define what to record and how to process the audit
event.
Consider the following commands:
/* create audit event handler */
CREATE PROCEDURE sec.log_id (schema1 varchar2, table1 varchar2, policy1
varchar2) AS
```

```
BEGIN
UTIL_ALERT_PAGER(schema1, table1, policy1);      -- send an alert note to my
pager
END;

/* add the policy */
DBMS_FGA.ADD_POLICY(
object_schema => 'hr',
object_name   => 'emp',
policy_name   => 'chk_hr_emp',
audit_condition => 'dept = ''SALES'' ',
audit_column => 'salary',
handler_schema => 'sec',
handler_module => 'log_id',
enable               =>  TRUE);
```

---

**Note:** Since the words "schema" and "table" are reserved words, they cannot be used as variables without some alteration, such as appending "1" as is done here.

---

What happens when these commands are issued? After the fetch of the first interested row, the event is recorded, and the audit function `SEC.LOG_ID` is executed. The audit event record generated is stored in `DBA_FGA_AUDIT_TRAIL` (fga_log$), which has reserved columns (such as `SQL_TEXT` and `SQL_BIND`) for recording SQL text, policy name, and other information. The query's `SQLBIND` and `SQLTEXT` are recorded in the `LSQLTEXT` and `LSQLBIND` columns of fga_log$ only if the policy specified audit_trail = `DBMS_FGA.DB_EXTENDED`.

---

**Note:** Fine-grained auditing is supported only with cost-based optimization. For queries using rule-based optimization, audit will check before applying row filtering, which could result in an unnecessary audit event trigger.

---

**See Also:**

- *Oracle Database Application Developer's Guide - Fundamentals* for information about using fine-grained auditing
- The DBMS_FGA chapter in *PL/SQL Packages and Types Reference*

**Note:** Policies currently in force on an object involved in a flashback query are applied to the data returned from the specified flashback snapshot (based on time or SCN).

## The DBMS_FGA Package

The DBMS_FGA package provides fine-grained security functions. Execute privilege on DBMS_FGA is needed for administering audit policies. Because the audit function can potentially capture all user environment and application context values, policy administration should be executable by privileged users only.

This feature is available for only cost-based optimization. The rule-based optimizer may generate unnecessary audit records since audit monitoring can occur before row filtering. For both the rule-based optimizer and the cost-based optimizer, you can refer to DBA_FGA_AUDIT_TRAIL to analyze the SQL text and corresponding bind variables that are issued.

The procedures for this package are described in the following subsections:

- ADD_POLICY Procedure
- DROP_POLICY Procedure
- ENABLE_POLICY Procedure
- DISABLE_POLICY Procedure

The syntax, parameters, and usage notes accompanying each procedure description also discuss the defaults and restrictions that apply to it.

### ADD_POLICY Procedure

This procedure creates an audit policy using the supplied predicate as the audit condition. The maximum number of FGA policies on any table or view object is 256.

#### Syntax

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

## Parameters

***Table 11–1 ADD_POLICY Procedure Parameters***

| Parameter | Description | Default Value |
|---|---|---|
| `object_schema` | The schema of the object to be audited. (If NULL, the current effective user schema is assumed.) | NULL |
| `object_name` | The name of the object to be audited. | - |
| `policy_name` | The unique name of the policy. | - |
| `audit_condition` | A condition in a row that indicates a monitoring condition. NULL is allowed and acts as TRUE. | NULL |
| `audit_column` | The columns to be checked for access. These can include hidden columns. The default, `NULL`, causes audit if any column is accessed or affected. | NULL |
| `handler_schema` | The schema that contains the event handler. The default, NULL, causes the current schema to be used. | NULL |
| `handler_module` | The function name of the event handler; includes the package name if necessary. This is fired only after the first row that matches the audit condition is processed in the query. If the procedure fails with exception, the user SQL statement will fail as well. | NULL |
| `enable` | Enables the policy if TRUE, which is the default. | TRUE |
| `statement_types` | The SQL statement types to which this policy is applicable: insert, update, delete, or select only. | SELECT |
| `audit_trail` | Whether to populate `LSQLTEXT` and `LSQLBIND` in fga_log$. | DB_EXTENDED |
| `audit_column_opts` | Establishes whether a statement is audited when the query references *any* column specified in the audit_column parameter or only when *all* such columns are referenced. | ANY_COLUMNS |

## Usage Notes

- Sample command: `DBMS_FGA.ADD_POLICY(object_schema => 'scott', object_name=>'emp', policy_name => 'mypolicy1',` `audit_condition => 'sal < 100', audit_column =>'comm, credit_card, expirn_date', handler_schema => NULL, handler_module => NULL, enable => TRUE, statement_types=> 'INSERT, UPDATE'`);

- An FGA policy should not be applied to out-of-line columns such as LOB columns.
- The audit_condition must be a boolean expression that can be evaluated using the values in the row being inserted, updated, or deleted. This condition can be NULL (or omitted), which is interpreted as `TRUE`, but it cannot contain the following elements:
  - Subqueries or sequences
  - Any direct use of SYSDATE, UID, USER or USERENV functions. However, a user-defined function and other SQL functions can use these functions to return the desired information.
  - Any use of the pseudocolumns LEVEL, PRIOR, or ROWNUM.

  Specifying an audit condition of "1=1" to force auditing of all specified statements ("statement_types") affecting the specified column ("audit_column") is no longer needed to achieve this purpose. `NULL` will cause audit even if no rows were processed, so that all actions on a table with this policy are audited.
- If object_schema is NULL, the current effective user schema is assumed.
- The audit function (handler_module) is an alerting mechanism for the administrator. The required interface for such a function is as follows:

  ```
  PROCEDURE <fname> ( object_schema VARCHAR2, object_name VARCHAR2, policy_
  name VARCHAR2 )  AS ...
  ```

  where `fname` is the name of the procedure, `object_schema` is the name of the schema of the table audited, `object_name` is the name of the table to be audited, and `policy_name` is the name of the policy being enforced.
- Each audit policy is applied to the query individually. However, at most one audit record may be generated for each policy, no matter how many rows being returned satisfy that policy's audit_condition. In other words, whenever any number of rows being returned satisfy an audit condition defined on the table, a single audit record will be generated for each such policy.
- If a table with an FGA policy defined on it receives a Fast Path insert or a vectored update, the hint is automatically disabled before any such operations.

Disabling the hint allows auditing to occur according to the policy's terms. (One example of a Fast Path insert is the statement `INSERT-WITH-APPEND`-hint.)

- The audit_trail parameter specifies whether to record the query's Sql Text and Sql Bind variable information in the FGA audit trail (fga_log$) columns named `LSQLTEXT` and `LSQLBIND`:
  - To populate, set to `DBMS_FGA.DB_EXTENDED` (the default)
  - To leave unpopulated, set to `DBMS_FGA.DB.`

  The audit_trail parameter appears in the ALL_AUDIT_POLICIES view.
- The audit_column_opts parameter establishes whether a statement is audited
  - when the query references any column specified in the audit_column parameter (audit_column_opts = `DBMS_FGA.ANY_COLUMNS`), or
  - only when all such columns are referenced (audit_column_opts = `DBMS_FGA.ALL_COLUMNS`).

  The default is `DBMS_FGA.ANY_COLUMNS`.

  The ALL_AUDIT_POLICIES view also shows audit_column_opts.

## DROP_POLICY Procedure

This procedure drops an audit policy.

### Syntax

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

### Parameters

***Table 11–2 DROP_POLICY Procedure Parameters***

| Parameter | Description |
|---|---|
| `object_schema` | The schema of the object to be audited. (If NULL, the current effective user schema is assumed.) |
| `object_name` | The name of the object to be audited. |
| `policy_name` | The unique name of the policy. |

### Usage Notes

The DBMS_FGA procedures cause current DML transactions, if any, to commit before the operation. However, the procedures do not cause a commit first if they are inside a DDL event trigger. With DDL transactions, the DBMS_FGA procedures are part of the DDL transaction. The default value for object_schema is NULL. (If NULL, the current effective user schema is assumed.)

## ENABLE_POLICY Procedure

This procedure enables an audit policy.

### Syntax

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

### Parameters

*Table 11–3 ENABLE_POLICY Procedure Parameters*

| Parameter | Description |
|---|---|
| object_schema | The schema of the object to be audited. (If NULL, the current effective user schema is assumed.) |
| object_name | The name of the object to be audited. |
| policy_name | The unique name of the policy. |
| enable | Defaults to TRUE to enable the policy. |

## DISABLE_POLICY Procedure

This procedure disables an audit policy.

### Syntax

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

## Parameters

***Table 11–4 DISABLE_POLICY Procedure Parameters***

| Parameter | Description |
|---|---|
| `object_schema` | The schema of the object to be audited. (If NULL, the current effective user schema is assumed.) |
| `object_name` | The name of the object to be audited. |
| `policy_name` | The unique name of the policy. |

The default value for object_schema is NULL. (If NULL, the current effective user schema is assumed.)

# 12 Introducing Database Security for Application Developers

Creating an application security policy is the first step when writing secure database applications. An application security policy is a list of application security requirements and rules that regulate user access to database objects.

This chapter discusses aspects of application security and Oracle Database features that you should consider when drafting security policies for database applications. It contains the following topics:

- About Application Security Policies
- Considerations for Using Application-Based Security
- Managing Application Privileges
- Creating Secure Application Roles
- Associating Privileges with the User's Database Role
- Protecting Database Objects Through the Use of Schemas
- Managing Object Privileges

## About Application Security Policies

You should draft security policies for each database application. For example, each database application should have one or more database roles that provide different levels of security when executing the application. The database roles can be granted to user roles, or directly to specific usernames.

Applications that potentially allow unrestricted SQL statement execution (through tools such as SQL*Plus) also need security policies that prevent malicious access to confidential or important schema objects.

The following sections describe aspects of application security and the Oracle Database features that you can use to plan and develop secure database applications.

**See Also:**

- Chapter 7, "Security Policies" for an overview of database security policies
- "Application Developer Security" on page 7-9 for a discussion of application developer database privileges
- "Application Administrator Security" on page 7-11 for a description of the security-related tasks of an application administrator

## Considerations for Using Application-Based Security

Two main issues to consider when you formulate and implement application security are listed as follows:

- Are Application Users Also Database Users?
- Is Security Enforced in the Application or in the Database?

### Are Application Users Also Database Users?

Oracle Corporation recommends that, where possible, you build applications in which application users are database users. In this way you can leverage the intrinsic security mechanisms of the database.

For many commercial packaged applications, application users are not database users. For these applications, multiple users authenticate themselves to the application, and the application then connects to the database as a single, highly-privileged user. We will call this the "One Big Application User" model.

Applications built in this fashion generally cannot use many of the intrinsic security features of the database, because the identity of the user is not known to the database.

For example, use of the following features is compromised by the One Big Application User model:

| Oracle Feature | Limitations of "One Big Application User" Model |
|---|---|
| Auditing | A basic principle of security is accountability through auditing. If all actions in the database are performed by One Big Application User, then database auditing cannot hold individual users accountable for their actions. The application must implement its own auditing mechanisms to capture individual users' actions. |
| Oracle Advanced Security enhanced authentication | Strong forms of authentication supported by Oracle Advanced Security (such as, client authentication over SSL, tokens, and so on) cannot be used if the client authenticating to the database is the application, rather than an individual user. |
| Roles | Roles are assigned to database users. Enterprise roles are assigned to enterprise users who, though not created in the database, are known to the database. If application users are not database users, then the usefulness of roles is diminished. Applications must then craft their own mechanisms to distinguish between the privileges which various application users need to access data within the application. |
| Enterprise user management feature of Oracle Advanced Security | This feature enables an Oracle database to use the Oracle Identity Management Infrastructure by securely storing and managing user information and authorizations in an LDAP-based directory such as Oracle Internet Directory. While enterprise users do not need to be created in the database, they do need to be known to the database. The One Big Application User model cannot take advantage of Oracle Identity Management. |

## Is Security Enforced in the Application or in the Database?

Applications whose users are also database users can either build security into the application, or rely upon intrinsic database security mechanisms such as granular privileges, virtual private database (fine-grained access control with application context), roles, stored procedures, and auditing (including fine-grained auditing). To the extent possible, Oracle recommends that applications utilize the security enforcement mechanisms of the database.

When security is enforced in the database itself, rather than in the application, it cannot be bypassed. The main shortcoming of application-based security is that security is bypassed if the user bypasses the application to access data. For example, a user who has SQL*Plus access to the database can execute queries without going through the Human Resources application. The user thus bypasses all of the security measures in the application.

Applications that use the One Big Application User model must build security enforcement into the application rather than use database security mechanisms. Because it is the application—and not the database—which recognizes users, the application itself must enforce security measures for each user.

This approach means that each application which accesses data must re-implement security. Security becomes expensive because organizations must implement the same security policies in multiple applications. Each new application requires an expensive re-implementation.

**See Also:** "Use of Ad Hoc Tools a Potential Security Problem" on page 13-18

## Managing Application Privileges

Most database applications involve different privileges on different schema objects. Keeping track of which privileges are required for each application can be complex. In addition, authorizing users to run an application can involve many `GRANT` operations.

To simplify application privilege management, you can create a role for each application and grant that role all the privileges a user needs to run the application. In fact, an application might have a number of roles, each granted a specific subset of privileges that allow greater or lesser capabilities while running the application.

For example, suppose that every administrative assistant uses the Vacation application to record vacation taken by members of the department. To best manage this application, you should:

1. Create a `VACATION` role.
2. Grant all privileges required by the Vacation application to the `VACATION` role.
3. Grant the `VACATION` role to all administrative assistants or to a role named `ADMIN_ASSISTS` (if previously defined).

Grouping application privileges in a role aids privilege management. Consider the following administrative options:

- You can grant the role, rather than many individual privileges, to those users who run the application. Then, as employees change jobs, you need to grant or revoke only one role, rather than many privileges.
- You can change the privileges associated with an application by modifying only the privileges granted to the role, rather than the privileges held by all users of the application.
- You can determine which privileges are necessary to run a particular application by querying the `ROLE_TAB_PRIVS` and `ROLE_SYS_PRIVS` data dictionary views.
- You can determine which users have privileges on which applications by querying the `DBA_ROLE_PRIVS` data dictionary view.

**See Also:** Chapter 10, "Administering User Privileges, Roles, and Profiles" for a complete discussion of creating, enabling, and disabling roles, and granting and revoking privileges

## Creating Secure Application Roles

After database access privileges are grouped into roles, the roles are granted to the application user. Securing these roles can be accomplished in two ways:

- Roles secured by embedding passwords inside their applications, which are called *application roles*, or
- Application developers can create application roles and specify which PL/SQL package is authorized to enable the roles, which are called *secure application roles*.

Within the package that implements the secure application role:

- The application must do the necessary validation. For example, the application must validate that the user is in a particular department, the user session was created by proxy, the request comes from a particular IP address, or that the user was authenticated using an X.509 certificate. To perform the validation, applications can use session information accessible by using the `SYS_CONTEXT` SQL function with the USERENV namespace attributes (`'userenv', <session_attribute>`). The information returned by this function can indicate the way the user was authenticated, the IP address of the client, and whether the user was proxied.

- The application must issue a `SET_ROLE` command using dynamic SQL (`DBMS_SESSION.SET ROLE`).

**Note:** Because users cannot change the security domain inside **definer's rights procedures**, secure application roles can only be enabled inside **invoker's rights procedures**.

**See Also:**

- "Secure Application Roles" on page 5-27 for conceptual information about this topic
- Table 13–1, " Key to Predefined Attributes in USERENV Namespace" on page 13-12
- *PL/SQL User's Guide and Reference* for information about definer's rights versus invoker's rights procedures and how to create them

## Example of Creating a Secure Application Role

**Note:** You need to set up the following data structures for the examples in this section to work:

```
CREATE OR REPLACE PACKAGE hr_logon IS
PROCEDURE hr_set_responsibility;
END;
/

CREATE OR REPLACE PACKAGE BODY hr_logon IS
PROCEDURE hr_set_responsibility IS
   BEGIN
      DBMS_SESSION.SET_IDENTIFIER (1234);
   END;
END;
/
```

To create a secure application role:

1. Create the roles as application roles and specify the authorized package that will enable the roles. In this example, `hr.hr_admin` is the example authorized package.

   ```
   CREATE ROLE admin_role IDENTIFIED USING hr.hr_admin;
   CREATE ROLE staff_role IDENTIFIED USING hr.hr_admin;
   ```

2. Create an invoker's right procedure.

   ```
   /* Create a dedicated authentication function for manageability so that
   changes in authentication policies would not affect the source code of the
   application - this design is up to the application developers */
   /* the only policy in this function is that current user must have been
   authenticated using the proxy user 'SCOTT' */
   CREATE OR REPLACE FUNCTION hr.MySecurityCheck RETURN BOOLEAN
   AS
   BEGIN
       /* a simple check to see if current session is authenticated
       by the proxy user 'SCOTT' */
       if (sys_context('userenv','proxy_user') = 'SCOTT')
       then
           return TRUE;
       else
           return FALSE;
       end IF;
   END;

   GRANT EXECUTE ON hr.MySecurityCheck TO PUBLIC;

   /*Create the procedure*/
   CREATE OR REPLACE PACKAGE hr_admin
   AUTHID CURRENT_USER
   IS
   PROCEDURE hr_app_report;
   END;
   /
   CREATE OR REPLACE PACKAGE BODY hr_admin IS
   PROCEDURE hr_app_report IS
   BEGIN
       /* set application context in 'responsibility' namespace */
       hr_logon.hr_set_responsibility;
       /* authentication check here */
       if (hr.MySecurityCheck = TRUE)
       then
   ```

```
        /* check 'responsibility' being set, then enable the roles without
        supplying the password */
        if (sys_context('hr','role') = 'admin' )
    then
        dbms_session.set_role('admin_role');
    else
        dbms_session.set_role('staff_role');
        end if;
    end if;
END;
END;
```

When enabling the secure application role, the database verifies that the authorized PL/SQL package is on the calling stack. This step verifies that the authorized PL/SQL package is issuing the command to enable the role. Also, when enabling the user's default roles, no checking is performed for application roles.

You can use secure application role to ensure a database connection. Because a secure application role is a role implemented by a package, the package can validate that users can connect to the database through a middle tier or from a specific IP address. In this way, the secure application role prevents users from accessing data outside an application. They are forced to work within the framework of the application privileges that they have been granted.

## Associating Privileges with the User's Database Role

A single user can use many applications and associated roles. However, you should ensure that the user has only the privileges associated with the running database role. Consider the following scenario:

- The `ORDER` role (for the Order application) contains the `UPDATE` privilege for the `INVENTORY` table
- The `INVENTORY` role (for the Inventory application) contains the `SELECT` privilege for the `INVENTORY` table
- Several order entry clerks have been granted both the `ORDER` and `INVENTORY` roles

In this scenario, an order entry clerk who has been granted both roles, can use the privileges of the `ORDER` role when running the `INVENTORY` application to update the `INVENTORY` table. The problem is that updating the `INVENTORY` table is not an authorized action when using the `INVENTORY` application, but only when using the `ORDER` application.

To avoid such problems, consider using either the SET ROLE statement or the SET_ROLE procedure as explained in the following section. You can also use the secure application role feature to allow roles to be set based on criteria you define.

Topics in this section include:

- Using the SET ROLE Statement
- Using the SET_ROLE Procedure
- Examples of Assigning Roles with Static and Dynamic SQL

## Using the SET ROLE Statement

Use a SET ROLE statement at the beginning of each application to automatically enable its associated role and to disable all others. In this way, each application dynamically enables particular privileges for a user only when required.

The SET ROLE statement simplifies privilege management. You control what information users can access and when they can access it. The SET ROLE statement also keeps users operating in a well-defined privilege domain. If a user obtains privileges only from roles, the user cannot combine these privileges to perform unauthorized operations.

**See Also:**

- "When Do Grants and Revokes Take Effect?" on page 10-35 for information about enabling and disabling roles
- "The SET ROLE Statement" on page 10-35

## Using the SET_ROLE Procedure

The PL/SQL package DBMS_SESSION.SET_ROLE is functionally equivalent to the SET ROLE statement in SQL. Roles are not supported in definer's rights procedures, so the DBMS_SESSION.SET_ROLE command cannot be called from them. However, the DBMS_SESSION.SET_ROLE command can be called from the following:

- Anonymous PL/SQL blocks
- Invoker's rights stored procedures (except those invoked from within definer's rights procedures)

SET ROLE takes effect only at execution time. Because anonymous blocks compile and execute simultaneously, roles are set before security checks are performed, so the block completes successfully. With respect to invoker's rights stored procedures, if they contain static SQL statements and access to objects in the SQL are authorized through roles, then the procedure may fail during compilation. (Because the roles are not enabled until the procedure executes.) To resolve this problem, replace static SQL with dynamic SQL by using the `DBMS_SQL` package. Then security checks are performed at execution, at the same time the `SET ROLE` statement enables roles.

**Note:** If you use `DBMS_SESSION.SET_ROLE` within an invoker's rights procedure, the role remains in effect until you explicitly disable it. In keeping with the *least privilege* principle, (that users should have the fewest privileges they need to do their jobs), you should explicitly disable roles set within an invoker's rights procedure, at the end of the procedure.

**See Also:** *PL/SQL Packages and Types Reference* for information about the `DBMS_SESSION` and the `DBMS_SQL` packages

## Examples of Assigning Roles with Static and Dynamic SQL

This section shows how static and dynamic SQL affect the assignment of roles.

**Note:** You need to set up the following data structures for the examples in this section to work:

```
CONNECT system/manager
DROP USER joe CASCADE;
CREATE USER joe IDENTIFIED BY joe;
GRANT CREATE SESSION, RESOURCE, UNLIMITED TABLESPACE TO joe;
GRANT CREATE SESSION, RESOURCE, UNLIMITED TABLESPACE TO scott;
DROP ROLE acct;
CREATE ROLE acct;
GRANT acct TO scott;
ALTER USER scott DEFAULT ROLE ALL EXCEPT acct;

CONNECT joe/joe;
CREATE TABLE finance (empno NUMBER);
GRANT SELECT ON finance TO acct;
CONNECT scott/tiger
```

Suppose you have a role named ACCT that has been granted privileges allowing you to select from table FINANCE in the JOE schema. In this case, the following procedure that uses static SQL fails:

```
CREATE OR REPLACE PROCEDURE statSQL_proc
AUTHID CURRENT_USER AS
    n NUMBER;
BEGIN
    SYS.DBMS_SESSION.SET_ROLE('acct');
    SELECT empno INTO n FROM JOE.FINANCE;
END;
```

The procedure fails because the security check which verifies that you have the SELECT privilege on table JOE.FINANCE occurs at compile time. At compile time, however, the ACCT role is not yet enabled. The role is not enabled until the procedure is executed.

In contrast, the DBMS_SQL package, which uses dynamic SQL, is not subject to this restriction. When you use this package, the security checks are performed when the procedure executes—not when it is compiled. Thus, the following block is successful:

```
CREATE OR REPLACE PROCEDURE dynSQL_proc
AUTHID CURRENT_USER AS
   n NUMBER;
BEGIN
   SYS.DBMS_SESSION.SET_ROLE('acct');
   EXECUTE IMMEDIATE 'select empno from joe.finance' INTO n;
    --other calls to SYS.DBMS_SQL
END;
/
```

**See Also:** "Choosing Between Native Dynamic SQL and the DBMS_SQL Package" in *Oracle Database Application Developer's Guide - Fundamentals*

## Protecting Database Objects Through the Use of Schemas

A *schema* is a security domain that can contain database objects. The privileges granted to each user or role control access to these database objects. This section covers:

- Unique Schemas
- Shared Schemas

### Unique Schemas

Most schemas can be thought of as usernames: the accounts which enable users to connect to a database and access the database objects. However, *unique schemas* do not allow connections to the database, but are used to contain a related set of objects. Schemas of this sort are created as normal users, yet are not granted the CREATE SESSION system privilege (either explicitly or through a role). However, you must temporarily grant the CREATE SESSION and RESOURCE privilege to such schemas if you want to use the CREATE SCHEMA statement to create multiple tables and views in a single transaction.

For example, the schema objects for a specific application might be owned by a given schema. If application users have the privileges to do so, then they can connect to the database using typical database usernames and use the application and the corresponding objects. However, no user can connect to the database using the schema set up for the application. This configuration prevents access to the associated objects through the schema, and provides another layer of protection for schema objects. In this case, the application could issue an `ALTER SESSION SET CURRENT_SCHEMA` statement to connect the user to the correct application schema.

### Shared Schemas

For many applications, users do not need their own accounts— their own schemas—in a database. These users only need to access an application schema. For example, users John, Firuzeh, and Jane are all users of the Payroll application, and they need access to the `Payroll` schema on the `Finance` database. None of them need to create their own objects in the database. They need only access `Payroll` objects. To address this issue, Oracle Advanced Security provides enterprise users (schema-independent users).

Enterprise users, users managed in a directory service, do not need to be created as database users because they use a shared database schema. To reduce administration costs, administrator can create an enterprise user once in the directory, and point the user at a shared schema that many other enterprise users can also access.

**See Also:** *Oracle Advanced Security Administrator's Guide* for information about how shared schemas are created and used for Enterprise User Security

## Managing Object Privileges

As part of designing your application, you need to determine the types of users who will be working with the application, and the level of access they need to accomplish their designated tasks. You must categorize these users into role groups, and then determine the privileges that must be granted to each role. This section covers:

- What Application Developers Need to Know About Object Privileges
- SQL Statements Permitted by Object Privileges

### What Application Developers Need to Know About Object Privileges

End users are typically granted object privileges. An object privilege allows a user to perform a particular action on a specific table, view, sequence, procedure, function, or package. Table 12–1 summarizes the object privileges available for each type of object.

***Table 12–1 How Privileges Relate to Schema Objects***

| Object Privilege | Applies to Table? | Applies to View? | Applies to Sequence? | Applies to Procedure?[1] |
|---|---|---|---|---|
| `ALTER` | Yes | No | Yes | No |
| `DELETE` | Yes | Yes | No | No |
| `EXECUTE` | No | No | No | Yes |
| `INDEX` | Yes[2] | No | No | No |
| `INSERT` | Yes | Yes | No | No |
| `REFERENCES` | Yes[2] | No | No | No |
| `SELECT` | Yes | Yes[3] | Yes | No |
| `UPDATE` | Yes | Yes | No | No |

[1] Stand-alone stored procedures, functions, and public package constructs.
[2] Privilege that cannot be granted to a role.
[3] Can also be granted for snapshots.

## SQL Statements Permitted by Object Privileges

As you implement and test your application, you should create each necessary role. Test the usage scenario for each role to be certain that the users of your application will have proper access to the database. After completing your tests, coordinate with the administrator of the application to ensure that each user is assigned the proper roles.

Table 12–2 lists the SQL statements permitted by the object privileges shown in Table 12–1.

***Table 12–2 SQL Statements Permitted by Database Object Privileges***

| Object Privilege | SQL Statements Permitted |
|---|---|
| `ALTER` | `ALTER` object (table or sequence) |
| | `CREATE TRIGGER ON` object (tables only) |
| `DELETE` | `DELETE FROM` object (table, view, or synonym) |
| `EXECUTE` | `EXECUTE` object (procedure or function) |
| | References to public package variables |
| `INDEX` | `CREATE INDEX ON` object (table, view, or synonym) |
| `INSERT` | `INSERT INTO` object (table, view, or synonym) |
| `REFERENCES` | `CREATE` or `ALTER TABLE` statement defining a `FOREIGN KEY` integrity constraint on object (tables only) |
| `SELECT` | `SELECT...FROM` object (table, view, synonym, or snapshot) |
| | SQL statements using a sequence |

**See Also:** "Understanding User Privileges and Roles" on page 10-15 for a discussion of object privileges

# 13 Using Virtual Private Database to Implement Application Security Policies

Oracle Database provides the necessary tools to build secure applications, such as Virtual Private Database (VPD), which is the combination of fine-grained access control and application context. Fine-grained access control enables you to associate security policies to database objects. Application context enables you to define and access application or database session attributes. VPD combines these two features, enabling you to enforce security policies to control access at the row level, based on application or session attributes.

This chapter introduces these features, and then explains how and why you would use them in the following topics:

- About Virtual Private Database, Fine-Grained Access Control, and Application Context
- Introduction to Fine-Grained Access Control
- Introduction to Application Context
- Introduction to Global Application Context
- Enforcing Application Security
- User Models and Virtual Private Database

## About Virtual Private Database, Fine-Grained Access Control, and Application Context

*Virtual Private Database (VPD)* is the aggregation of server-enforced fine-grained access control and a secure application context in the Oracle database server. VPD enables you to build applications that enforce row-level security policies at the object level by dynamically appending predicates (WHERE clauses) to SQL statements that query data you want to protect. *Application context* is a feature that allows application developers to define, set, and access application attributes and then use these attributes to supply the predicate values for fine-grained access control policies. Local, or session-based, application contexts are stored in the UGA and are invoked each time an application user connects to the database. For multitiered environments where users access the database by way of connection pooling, non-session-based *global application context*, which stores the application context in the SGA, can be used. Although application context is an integral part of VPD, it can be implemented alone, without fine-grained access control. When application context is implemented alone, it can be used to access session information, such as the client identifier, to preserve user identity across multitiered environments.

The remainder of this chapter discusses how VPD works and introduces its main components—fine-grained access control and application context.

**See Also:**

- Chapter 14, "Implementing Application Context and Fine-Grained Access Control" for information about using local application context and global application context with or without VPD fine-grained access control policies.
- "Using the CLIENT_IDENTIFIER Attribute to Preserve User Identity" on page 15-12 for information about using the client identifier attribute to preserve user identity across multitiered environments.

### Introduction to VPD

Virtual private database (VPD) enables you to enforce security, to a fine level of granularity, directly on tables, views, or synonyms. Because security policies are attached directly to tables, views, or synonyms and automatically applied whenever a user accesses data, there is no way to bypass security.

When a user directly or indirectly accesses a table, view, or synonym that is protected with a VPD policy, the server dynamically modifies the user's SQL statement. The modification is based on a `WHERE` condition (known as a predicate) returned by a function which implements the security policy. The statement is modified dynamically, transparently to the user, using any condition which can be expressed in, or returned by a function. VPD policies can be applied to `SELECT`, `INSERT`, `UPDATE`, `INDEX`, and `DELETE` statements.

> **Note:** Users need full table access to create table indexes. Consequently, a user who has privileges to maintain an index can see all the row data although the user does not have full table access under a regular query. To prevent this, apply VPD policies to `INDEX` statements.

Functions which return predicates can also include callouts to other functions. Within your PL/SQL package, you can embed a C or Java callout that can either access operating system information, or return `WHERE` clauses from an operating system file or central policy store. A policy function can return different predicates for each user, for each group of users, or for each application. Using policy functions over synonyms can substitute for maintaining a separate view for each user or class of users, saving substantial overhead in memory and processing resources.

Application context enables you to securely access the attributes on which you base your security policies. For example, users with the position attribute of `manager` would have a different security policy than users with the position attribute of `employee`.

Consider an HR clerk who is only allowed to see employee records in the Retail Division. When the user initiates the query

```
SELECT * FROM emp;
```

the function implementing the security policy returns the predicate `division = 'RETAIL'`, and the database transparently rewrites the query. The query actually executed becomes:

```
SELECT * FROM emp WHERE division = 'RETAIL';
```

### Column-level VPD

Column-level VPD enables you to enforce row-level security when a security-relevant column is referenced in a query. You can apply column-level VPD to tables and views, but not to synonyms. By specifying the security-relevant column name with the `sec_relevant_cols` parameter of the `DBMS_RLS.ADD_POLICY` procedure, the security policy is applied whenever the column is referenced, explicitly or implicitly, in a query.

For example, users outside of the HR department typically are allowed to view only their own Social Security numbers. When a sales clerk initiates the query

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

the function implementing the security policy returns the predicate `ssn='my_ssn'` and the database rewrites the query and executes

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

> **See Also:** "Adding Policies for Column-Level VPD" on page 14-40 for information about how to add column-level VPD policies

### Column-level VPD with Column Masking Behavior

If a query references a security-relevant column, then the default behavior of column-level VPD restricts the number of rows returned. With column masking behavior, which can be enabled by using the `sec_relevant_cols_opt` parameter of the `DBMS_RLS.ADD_POLICY` procedure, all rows display, even those that reference security relevant columns. However, the sensitive columns display as `NULL` values.

To illustrate this, consider the results of the sales clerk's query, described in the previous example. If column masking behavior is used, then instead of only seeing the row containing the sales clerk's own Social Security number, the clerk would see all rows from `emp`, but the `ssn` column values would be returned as `NULL`. Note that this behavior is fundamentally different from all other types of VPD policies, which return only a subset of rows.

> **See Also:** "Column Masking Behavior" on page 14-42 for information about how to add column-level VPD policies with column masking behavior.

### VPD Security Policies and Applications

The security policy is applied within the database itself, rather than within an application. This means that use of a different application will not bypass the security policy. Security can thus be built once, in the database, instead of being implemented again in multiple applications. Virtual Private Database therefore provides far stronger security than application-based security, at a lower cost of ownership.

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

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

## Introduction to Fine-Grained Access Control

Fine-grained access control enables you to build applications that enforce security policies at a low level of granularity. (These policies are also referred to as VPD policies.) You can use it, for example, to restrict a customer who is accessing an Oracle database server to see only his own account, a physician to see only the records of her own patients, or a manager to see only the records of employees who work for him.

When you use fine-grained access control, you create security policy functions attached to the table, view, or synonym on which you have based your application. Then, when a user enters a `SELECT` or a DML statement (`INSERT`, `UPDATE`, or `DELETE`) on that object, Oracle dynamically modifies the user's statement—transparently to the user—so that the statement implements the correct access control. You can also enforce security policies on index maintenance operations performed with the DDL statements `CREATE INDEX` and `ALTER INDEX`.

## Features of Fine-Grained Access Control

Fine-grained access control provides the following capabilities:

- Table-, View-, or Synonym-Based Security Policies
- Multiple Policies for Each Table, View, or Synonym
- Grouping of Security Policies
- High Performance
- Default Security Policies

### Table-, View-, or Synonym-Based Security Policies

Attaching security policies to tables, views, or synonyms rather than to applications provides greater security, simplicity, and flexibility.

**Security** Attaching a policy to a table, view, or synonym overcomes a potentially serious application security problem. Suppose a user is authorized to use an application, and then, drawing on the privileges associated with that application, wrongfully modifies the database by using an ad hoc query tool, such as SQL*Plus. By attaching security policies to tables, views, or synonyms, fine-grained access control ensures that the same security is in force, no matter how a user accesses the data.

**Simplicity** Adding the security policy to the table, view, or synonym means that you make the addition only once, rather than repeatedly adding it to each of your table-, view-, or synonym-based applications.

**Flexibility** You can have one security policy for `SELECT` statements, another for `INSERT` statements, and still others for `UPDATE` and `DELETE` statements. For example, you might want to enable a Human Resources clerk to `SELECT` all employee records in her division, but to `UPDATE` only salaries for those employees in her division whose last names begin with "A" through "F".

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

## Multiple Policies for Each Table, View, or Synonym

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

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

## Grouping of Security Policies

Since multiple applications, with multiple security policies, can share the same table, view, or synonym, it is important to identify those policies which should be in effect when the table, view, or synonym is accessed.

For example, in a hosting environment, Company A can host the `BENEFIT` table for Company B and Company C. The table is accessed by two different applications, Human Resources and Finance, with two different security policies. The Human Resources application authorizes users based on ranking in the company, and the

Finance application authorizes users based on department. To integrate these two policies into the BENEFIT table would require joint development of policies between the two companies, which is not a feasible option. By defining an application context to drive the enforcement of a particular set of policies to the base objects, each application can implement a private set of security policies.

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

### High Performance

With fine-grained access control, each policy function for a given query is evaluated only once, at statement parse time. Also, the entire dynamically modified query is optimized and the parsed statement can be shared and reused. This means that rewritten queries can take advantage of Oracle's high performance features, such as dictionary caching and shared cursors.

### Default Security Policies

While partitioning security policies by application is desirable, it is also useful to have security policies that are always in effect. In the previous example, a hosted application can always enforce data separation by subscriber_ID, whether you are using the Human Resources application or the Finance application. Default security policies allow developers to have base security enforcement under all conditions, while partitioning of security policies by application (using security groups) enables layering of additional, application-specific security on top of default security policies. To implement default security policies, you add the policy to the SYS_DEFAULT policy group.

**See Also:** The following topics for information about how to implement fine-grained access control:

- "How Fine-Grained Access Control Works" on page 14-29
- "How to Establish Policy Groups" on page 14-30
- "How to Add a Policy to a Table, View, or Synonym" on page 14-35
- "Examples: Application Context Within a Fine-Grained Access Control Function" on page 14-7

## About Creating a Virtual Private Database Policy with Oracle Policy Manager

To implement Virtual Private Database (VPD), developers can use the `DBMS_RLS` package to apply security policies to tables and views. They can also use the `CREATE CONTEXT` command to create application contexts.

Alternatively, developers can use the Oracle Policy Manager graphical user interface, accessed from Oracle Enterprise Manager, to apply security policies to schema objects, such as tables and views, and to create application contexts. Oracle Policy Manager provides an easy-to-use interface to manage security policies and application contexts, and therefore makes VPD easier to develop.

To create VPD policies, users must provide the schema name, table (or view or synonym) name, policy name, the function name that generates the predicate, and the statement types to which the policy applies (that is, `SELECT`, `INSERT`, `UPDATE`, `DELETE`). Oracle Policy Manager then executes the function `DBMS_RLS.ADD_POLICY`. You create an application context by providing the name of the context and the package that implements the context.

Oracle Policy Manager is also the administration tool for Oracle Label Security. 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 row-level 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.

**See Also:** *Oracle Label Security Administrator's Guide* for information about using Oracle Policy Manager

## Introduction to Application Context

Application context enables you to define, set, and access application attributes that you can use as a secure data cache which is available in UGA and SGA.

Most applications contain the kind of information that can be used for access control. For example, in an order entry application customers can be limited to accessing their own orders (ORDER_NUMBER) and customer number (CUSTOMER_NUMBER). These can be used as security attributes.

As an additional example, consider a user running a human resources application. Part of the application's initialization process is to determine the kind of responsibility that the user can assume, based on the user's identity. This responsibility ID becomes part of the human resource application context. It affects what data the user can access throughout the session.

You use the SQL function SYS_CONTEXT to configure application context with the following syntax:

```
SYS_CONTEXT ('namespace','parameter'[,length])
```

This section describes application context and how to use it. It includes:

- Features of Application Context
- Ways to Use Application Context with Fine-Grained Access Control

**See Also:** *Oracle Database SQL Reference* for detailed information about using SYS_CONTEXT

## Features of Application Context

Application context provides the following important security features:

- Specifying Attributes for Each Application
- Providing Access to Predefined Attributes through the USERENV Namespace
- Externalized Application Contexts

### Specifying Attributes for Each Application

Each application can have its own context with its own attributes. Suppose, for example, you have three applications: General Ledger, Order Entry, and Human Resources. You can specify different attributes for each application. Thus,

- For the General Ledger application context, you can specify the attributes SET_OF_BOOKS and TITLE.
- For the Order Entry application context, you can specify the attribute CUSTOMER_NUMBER.
- For the Human Resources application context, you can specify the attributes ORGANIZATION_ID, POSITION, and COUNTRY.

In each case, you can adapt the application context to your precise security needs.

### Providing Access to Predefined Attributes through the USERENV Namespace

Oracle database server provides a built-in application context namespace, USERENV, which provides access to predefined attributes. These attributes are session primitives—information which the database captures regarding a user's session. For example, the IP address from which a user connected, the username, and a proxy username (in cases where a user connection is proxied through a middle tier), are all available as predefined attributes through the USERENV application context.

Predefined attributes are useful for access control. For example, if you are using a three-tier application which creates lightweight user sessions through OCI or thick JDBC, you can access the PROXY_USER attribute in the USERENV application context to determine whether the user's session was created by a middle tier application. Your policy function could allow a user to access data only for connections where the user is proxied. If the user connects directly to the database, then she would not be able to access any data.

You can use the PROXY_USER attribute within VPD to ensure that users only access data through a particular middle-tier application. As a different approach, you can develop a secure application role. Then rather than each policy ensuring that users access the database through a specific proxy, the secure application role enforces this.

You can access predefined attributes through the USERENV application context, but you cannot change them. They are listed in Table 13–1.

Use the following syntax to return information about the current session.

```
SYS_CONTEXT('userenv', 'attribute')
```

**Note:** The USERENV application context namespace replaces the USERENV function provided in earlier database releases.

**See Also:**

- Chapter 15, "Preserving User Identity in Multitiered Environments" for information about proxy authentication and about using the USERENV attribute, CLIENT_IDENTIFIER, to preserve user identity across multiple tiers
- SYS_CONTEXT in the *Oracle Database SQL Reference* for complete details about the USERENV namespace and its predefined attributes

***Table 13–1 Key to Predefined Attributes in USERENV Namespace***

| Predefined Attribute | Meaning |
|---|---|
| AUDITED_CURSORID | Returns the fine-grained auditing cursor ID. |
| AUTHENTICATION_DATA | Returns the data being used to authenticate the login user. If the user has been authenticated through SSL, or if the user's certificate was proxied to the database, this includes the user's X.509 certificate |
| AUTHENTICATION_TYPE | Shows how the user was authenticated (DATABASE, OS, NETWORK, or PROXY) |
| BG_JOB_ID | Returns the background job ID |
| CLIENT_IDENTIFIER | User-defined client identifier for the session |
| CLIENT_INFO | Returns up to 64 bytes of user session information that can be stored by an application using the DBMS_APPLICATION_INFO package |
| CURRENT_BIND | Returns the bind variables for fine-grained auditing. Maximum length is 4K. |
| CURRENT_SCHEMA | Returns the name of the default schema being used in the current session. This can be changed with an ALTER SESSION SET SCHEMA statement. |
| CURRENT_SCHEMAID | Returns the identifier of the default schema being used in the current session. This can be changed with an ALTER SESSION SET SCHEMAID statement. |
| CURRENT_SQL | Returns SQL text of the query that triggers fine-grained audit or row-level security (RLS) policy functions or event handlers. Only valid inside the function or event handler. |

***Table 13–1 (Cont.) Key to Predefined Attributes in USERENV Namespace***

| Predefined Attribute | Meaning |
|---|---|
| `CURRENT_SQL1` to `CURRENT_SQL7` | Returns 4K length substrings of the SQL query text that triggers fine-grained audit or row-level security (RLS) policy functions or audit event handlers. Only valid inside the RLS policy function or event handler. Maximum length is 32K. For example, if a user issued a 32 K length SQL statement, then `CURRENT_SQL` returns 0 to 4K, `CURRENT_SQL1` returns 5K to 8K, `CURRENT_SQL2` returns 9K to 12K, and so on. |
| `CURRENT_SQL_LENGTH` | Returns the length of the current SQL statement that triggers fine-grained audit or row-level security (RLS) policy functions or event handlers. Only valid inside the function or event handler. |
| `CURRENT_USER` | Returns name of user under whose privilege the current session runs. Can be different from `SESSION_USER` from within a stored procedure (such as an invoker's rights procedure). |
| `CURRENT_USERID` | Returns the user ID of the user under whose privilege the current session runs. Can be different from `SESSION_USERID` from within a stored procedure (such as an invoker's rights procedure). |
| `DB_DOMAIN` | Returns the domain of the database as specified in the `DB_DOMAIN` initialization parameter |
| `DB_NAME` | Returns the name of the database as specified in the `DB_NAME` initialization parameter |
| `ENTRYID` | Returns available auditing entry identifier. Incremented for every audit record for a SQL statement. Note: there can be more than one audit record for the same SQL statement. |
| `EXTERNAL_NAME` | Returns the external name of the database user |
| `FG_JOB_ID` | Returns the foreground job ID |
| `GLOBAL_CONTEXT_MEMORY` | Amount of shared memory used by global application context, in bytes |
| `GLOBAL_UID` | Returns the user Login name from Oracle Internet Directory. |
| `HOST` | Returns the name of the host machine on which the database is running |
| `INSTANCE` | Returns instance identification number of the current instance |
| `INSTANCE_NAME` | Returns the name of the instance. |

***Table 13–1 (Cont.) Key to Predefined Attributes in USERENV Namespace***

| Predefined Attribute | Meaning |
|---|---|
| `IP_ADDRESS` | Returns the IP address (when available) of the machine from which the client is connected |
| `ISDBA` | Returns `TRUE` if you currently have the `DBA` role enabled and `FALSE` if you do not. |
| `LANG` | Returns abbreviation for the language name |
| `LANGUAGE` | Returns the language and territory currently used by the session, along with the database character set in the form: *language_territory.characterset* |
| `NETWORK_PROTOCOL` | Returns the protocol named in the connect string (`PROTOCOL`=*protocol*) |
| `NLS_TERRITORY` | Returns the territory of the current session |
| `NLS_CURRENCY` | Returns the currency symbol of the current session |
| `NLS_CALENDAR` | Returns the calendar used for dates in the current session |
| `NLS_DATE_FORMAT` | Returns the current date format of the current session |
| `NLS_DATE_LANGUAGE` | Returns language used to express dates in the current session |
| `NLS_SORT` | Indicates whether the sort base is binary or linguistic |
| `OS_USER` | Returns the operating system username of the client process that initiated the database session |
| `POLICY_INVOKER` | Returns the invoker of row-level security policy functions. |
| `PROXY_USER` | Returns the name of the database user (typically middle tier) who opened the current session on behalf of `SESSION_USER` |
| `PROXY_USERID` | Returns identifier of the database user (typically middle tier) who opened the current session on behalf of `SESSION_USER` |
| `SERVER_HOST` | Returns the hostname of machine on which the instance is running. |
| `SESSION_USER` | Returns the database user name by which the current user is authenticated |
| `SESSION_USERID` | Returns the identifier of the database user name by which the current user is authenticated |
| `SESSIONID` | Returns auditing session identifier |
| `SID` | Returns the session number (different from the session ID). |

*Table 13–1 (Cont.) Key to Predefined Attributes in USERENV Namespace*

| Predefined Attribute | Meaning |
|---|---|
| `STATEMENTID` | Returns available auditing statement identifier. Incremented once for every SQL statement audited in a session. |
| `TERMINAL` | Returns the operating system identifier for the client of the current session. "Virtual" in TCP/IP |

### Externalized Application Contexts

Many applications store attributes used for fine-grained access control within a database metadata table. For example, an `EMPLOYEES` table could include cost center, title, signing authority, and other information useful for fine-grained access control. Organizations also centralize user information for user management and access control in LDAP-based directories, such as Oracle Internet Directory. Application context attributes can be stored in Oracle Internet Directory and assigned to one or more enterprise users. They can be retrieved automatically upon login for an enterprise user and then used to initialize an application context.

**Note:** Enterprise User Security is a feature of Oracle Advanced Security.

**See Also:**

- "Initializing Application Context Externally" on page 14-18 for information about initializing local application context through external resources such as an OCI interface, a job queue process, or a database link.
- "Initializing Application Context Globally" on page 14-19 for information about initializing local application context through a centralized resource, such as Oracle Internet Directory.
- *Oracle Advanced Security Administrator's Guide* for information about enterprise users.

## Ways to Use Application Context with Fine-Grained Access Control

To simplify security policy implementation, you can use application context within a fine-grained access control function.

Application context can be used in the following ways with fine-grained access control:

- Using Application Context as a Secure Data Cache
- Using Application Context to Return a Specific Predicate (Security Policy)
- Using Application Context to Provide Attributes Similar to Bind Variables in a Predicate

### Using Application Context as a Secure Data Cache

Accessing an application context inside your fine-grained access control policy function is like writing down an often-used phone number and posting it next to your phone, where you can find it easily rather than looking it up every time you need it.

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

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

### Using Application Context to Return a Specific Predicate (Security Policy)

You can use application context to return the correct security policy, enforced through a predicate here.

Consider an order entry application which enforces the rules, "customers only see their own orders, and clerks see all orders for all customers." These are two different policies. You could define an application context with a `Position` attribute, and this attribute could be accessed within the policy function to return the correct predicate, depending on the value of the attribute. Thus, you can enable a user in the `Clerk` position to retrieve all orders, but a user in the `Customer` position to see his own records only.

To design a fine-grained access control policy to return a specific predicate for an attribute, access the application context within the function that implements the policy. For example, to limit customers to seeing their own records only, use fine-grained access control to dynamically modify the user's query from this:

```
SELECT * FROM Orders_tab
```

to this:

```
SELECT * FROM Orders_tab
   WHERE Custno = SYS_CONTEXT ('order_entry', 'cust_num');
```

### Using Application Context to Provide Attributes Similar to Bind Variables in a Predicate

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

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

Note that the `SYS_CONTEXT` function works much like a bind variable, but only if the `SYS_CONTEXT` arguments are constants.

> **See Also:** "Examples: Application Context Within a Fine-Grained Access Control Function" on page 14-7 which provides a code example.

## Introduction to Global Application Context

In many application architectures, the middle tier application is responsible for managing session pooling for application users. Users authenticate themselves to the application, which uses a single identity to log in to the database and maintains all the connections. In this environment, it is not possible to maintain application attributes using session-dependent application context (local application context) because of the sessionless model of the application.

Another scenario is when a user is connected to the database through an application (such as Oracle Forms) which then spawns other applications (such as Oracle

Reports) to connect to the database. These applications may need to share the session attributes such that they appear to be sharing the same database session.

Global application context is a type of secure application context that can be shared among trusted sessions. In addition to driving the enforcement of the fine-grained access control policies, applications (especially middle-tier products) can use this support to manage application attributes securely and globally.

**Note:**

- Global application context is not available in Real Application Clusters.
- Oracle Connection Manager, a router provided with Oracle Net Services, cannot be used with global application context.

## Enforcing Application Security

This section contains information about enforcing application security. This section consists of the following topics:

- Use of Ad Hoc Tools a Potential Security Problem
- Restricting SQL*Plus Users from Using Database Roles
- Virtual Private Database and Oracle Label Security Exceptions and Exemptions

### Use of Ad Hoc Tools a Potential Security Problem

Prebuilt database applications explicitly control the potential actions of a user, including the enabling and disabling of the user's roles while using the application. By contrast, ad hoc query tools, such as SQL*Plus, allow a user to submit any SQL statement (which may or may not succeed), including the enabling and disabling of any granted role.

Potentially, an application user can exercise the privileges attached to that application to issue destructive SQL statements against database tables by using an ad hoc tool.

For example, consider the following scenario:

- The Vacation application has a corresponding `VACATION` role.
- The `VACATION` role includes the privileges to issue `SELECT`, `INSERT`, `UPDATE`, and `DELETE` statements against the `EMP_TAB` table.

- The Vacation application controls the use of privileges obtained through the VACATION role.

Now, consider a user who has been granted the VACATION role. Suppose that, instead of using the Vacation application, the user executes SQL*Plus. At this point, the user is restricted only by the privileges granted to him explicitly or through roles, including the VACATION role. Because SQL*Plus is an ad hoc query tool, the user is not restricted to a set of predefined actions, as with designed database applications. The user can query or modify data in the EMP_TAB table as he or she chooses.

## Restricting SQL*Plus Users from Using Database Roles

This section presents features that you may use in order to restrict SQL*Plus users from using database roles and thus, prevent serious security problems. These features include the following:

- Limit Roles Through PRODUCT_USER_PROFILE
- Use Stored Procedures to Encapsulate Business Logic
- Use Virtual Private Database for Highest Security

### Limit Roles Through PRODUCT_USER_PROFILE

DBAs can use PRODUCT_USER_PROFILE to disable certain SQL and SQL*Plus commands in the SQL*Plus environment for each user. SQL*Plus, not the Oracle Database, enforces this security. DBAs can even restrict access to the GRANT, REVOKE, and SET ROLE commands in order to control users' ability to change their database privileges.

The PRODUCT_USER_PROFILE table enables you to list roles which you do not want users to activate with an application. You can also explicitly disable use of various commands, such as SET ROLE.

For example, you could create an entry in the PRODUCT_USER_PROFILE table to:

- Disallow use of the CLERK and MANAGER roles with SQL*Plus
- Disallow use of SET ROLE with SQL*Plus

Suppose user Jane connects to the database using SQL*Plus. Jane has the CLERK, MANAGER, and ANALYST roles. As a result of the preceding entry in PRODUCT_ USER_PROFILE, Jane is only able to exercise her ANALYST role with SQL*Plus. Also, when Jane attempts to issue a SET ROLE statement, she is explicitly prevented from doing so because of the entry in the PRODUCT_USER_PROFILE table prohibiting use of SET ROLE.

Use of the PRODUCT_USER_PROFILE table does not completely guarantee security, for multiple reasons. In the preceding example, while SET ROLE is disallowed with SQL*Plus, if Jane had other privileges granted to her directly, she could exercise these using SQL*Plus.

**See Also:** *SQL*Plus User's Guide and Reference* for more information about the PRODUCT_USER_PROFILE table

### Use Stored Procedures to Encapsulate Business Logic

Stored procedures encapsulate use of privileges with business logic so that privileges are only exercised in the context of a well-formed business transaction. For example, an application developer might create a procedure to update employee name and address in the EMPLOYEES table, which enforces that the data can only be updated in normal business hours. Also, rather than grant a human resources clerk the UPDATE privilege on the EMPLOYEES table, a developer (or application administrator) may grant the privilege on the procedure only. Then, the human resources clerk can exercise the privilege only in the context of the procedures, and cannot update the EMPLOYEES table directly.

### Use Virtual Private Database for Highest Security

VPD provides the benefit of strong security policies, which apply directly to data. When you use VPD, you can enforce security no matter how a user gets to the data: whether through an application, through a query, or by using a report-writing tool.

**See Also:**

- "Introduction to VPD" on page 13-2
- "Ways to Use Application Context with Fine-Grained Access Control" on page 13-16
- "How to Add a Policy to a Table, View, or Synonym" on page 14-35 for information about using the DBMS_RLS.ADD_POLICY procedure to add policies for VPD.

## Virtual Private Database and Oracle Label Security Exceptions and Exemptions

Virtual Private Database and Oracle Label Security are not enforced during DIRECT path export. Also, Virtual Private Database policies and Oracle Label Security policies cannot be applied to objects in the SYS schema. As a consequence, the SYS user and users making a DBA-privileged connection to the database (for example, CONNECT/AS SYSDBA) do not have VPD or Oracle Label Security policies applied to their actions. The database user SYS is thus always exempt from VPD or Oracle Label Security enforcement, regardless of the export mode, application, or utility used to extract data from the database. However, SYSDBA actions can be audited by enabling such auditing upon installation and specifying that this audit trail be stored in a secure location in the operating system.

Similarly, database users granted the EXEMPT ACCESS POLICY privilege, either directly or through a database role, are exempt from VPD enforcements. They are also exempt from some Oracle Label Security policy enforcement controls — READ_CONTROL and CHECK_CONTROL — regardless of the export mode, application, or utility used to access the database or update its data. However, the following policy enforcement options remain in effect even when EXEMPT ACCESS POLICY is granted:

- INSERT_CONTROL, UPDATE_CONTROL, DELETE_CONTROL, WRITE_CONTROL, LABEL_UPDATE, and LABEL_DEFAULT.
- If the Oracle Label Security policy specifies the ALL_CONTROL option, then all enforcement controls are applied except READ_CONTROL and CHECK_CONTROL.

EXEMPT ACCESS POLICY is a very powerful privilege and should be carefully managed. It is inadvisable to grant this privilege WITH ADMIN OPTION because very few users should have this exemption.

**Note:**

- The EXEMPT ACCESS POLICY privilege does not affect the enforcement of 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.
- The SYS_CONTEXT values that VPD uses are not propagated to secondary databases for failover.

**See Also:** *Oracle Label Security Administrator's Guide*

## User Models and Virtual Private Database

Whether the user is a database user or an application user unknown to the database, Oracle provides different ways in which applications can enforce fine-grained access control for each user.

For applications in which the application users are also database users, VPD enforcement is relatively simple. Users connect to the database, and the application sets up application contexts for each session. Each session is initiated under a different username, so that it is simple to enforce different fine-grained access control conditions for different users. This is also possible when using proxy authentication, because each session in OCI or thick JDBC is a distinct database session, and has its own application context.

When proxy authentication is integrated with Enterprise User Security, user roles and other attributes can be retrieved from Oracle Internet Directory to enforce VPD. (In addition, globally initialized application context can also be retrieved from the directory.)

For applications in which a single user (for example, One Big Application User) connects to the database on behalf of all users, it is possible to have fine-grained access control for each user. An application developer can create a global application context attribute to represent the application user (for example, REALUSER). Although all database sessions and audit records are initiated as One Big Application User, each session can have attributes that vary, depending on who the real end user is. This model works best for applications that have a limited number of users and where sessions are not reused. In this model, the option to use roles and perform database auditing is diminished because each session is created as the same database user.

Web-based applications typically have hundreds if not thousands of users. There may be a persistent connection to the database (to support data retrieval for a number of user requests), but these connections are not specific to each Web-based user. To provide scalability, Web-based applications typically set up and reuse connections instead of having different sessions for each user. For example, Web user Jane and Ajit connect to a middle tier application, which establishes a session in the database used by the application on behalf of both users. Typically, neither Jane nor Ajit are known to the database. The application is responsible for switching the username on the connection, so that, at any given time, it's either Jane or Ajit using the session.

Oracle Database VPD capabilities facilitate connection pooling by allowing multiple connections to access one or more global application contexts, instead of setting up an application context for each distinct user session.

# 14 Implementing Application Context and Fine-Grained Access Control

Application context can be implemented with fine-grained access control as part of Virtual Private Database (VPD) or by itself to provide application developers a way to define, set, and access application attributes. When used alone, application context can serve as a secure data cache, saving the overhead of multiple queries to the database each time an application needs to access application attributes.

This chapter discusses how to implement application context and fine-grained access control. It contains the following topics:

| Topic Category | Links to Topics |
|---|---|
| Application Context | ■ About Implementing Application Context |
| | ■ How to Use Application Context |
| | ■ Examples: Application Context Within a Fine-Grained Access Control Function |
| | ■ Initializing Application Context Externally |
| | ■ Initializing Application Context Globally |
| | ■ How to Use Global Application Context |
| Fine-Grained Access Control | ■ How Fine-Grained Access Control Works |
| | ■ How to Establish Policy Groups |
| | ■ How to Add a Policy to a Table, View, or Synonym |
| | ■ How to Check for Policies Applied to a SQL Statement |
| | ■ Users Who Are Exempt from VPD Policies |
| | ■ Automatic Reparse |
| | ■ VPD Policies and Flashback Query |

## About Implementing Application Context

Application context can be used for the following purposes:

- Enforce fine-grained access control
- Preserve user identity across multitier environments
- Serve as a secure data cache, saving the overhead of multiple queries to the database each time an application needs to access application attributes

When application context is used as a secure data cache, applications can use the attributes stored in the context for PL/SQL control structures that use conditional statements or loops, or for fine-grained auditing.

There are two types of application contexts, depending on where the context information is stored:

- Session-based application context, where the context information is stored in the database user session (UGA), and
- Non-session-based, or *global application context*, where the context information is stored in the shared area (SGA).

Session-based application contexts can be initialized from external sources or they can be initialized globally. In either case, the context information is stored in the user session. Those session-based application contexts that are initialized externally can accept initialization of attributes and values through external resources such as an OCI interface, a job queue process, or a connected user database link. Those that are initialized globally can accept initialization of attributes and values from a centralized location, such as an LDAP directory.

Table 14–1 summarizes the different types of application contexts.

***Table 14–1 Types of Application Contexts***

| Application Context Type | Stored in UGA | Stored in SGA | Supports Connected User Database Links | Supports Centralized Storage of Users' Application Context | Supports Sessionless Multitier Applications |
|---|---|---|---|---|---|
| Application Context | X | | | | |
| Application Context Initialized Externally | X | | X | | |
| Application Context Initialized Globally | X | | | X | |
| Global Application Context | | X | | | X |

**See Also:**

- "Introduction to Application Context" on page 13-10 for conceptual information about session-based application context
- "Introduction to Global Application Context" on page 13-17 for conceptual information about non-session-based application context
- "Using the CLIENT_IDENTIFIER Attribute to Preserve User Identity" on page 15-12 for a discussion of using the `CLIENT_IDENTIFIER` attribute of the predefined `USERENV` application context
- "Fine-Grained Auditing" on page 11-29 for information about using application context with fine-grained auditing

## How to Use Application Context

To use application context, you perform the following tasks:

- Task 1: Create a PL/SQL Package that Sets the Context for Your Application
- Task 2: Create a Unique Context and Associate It with the PL/SQL Package
- Task 3: Set the Context Before the User Retrieves Data
- Task 4. Use the Context in a VPD Policy Function

### Task 1: Create a PL/SQL Package that Sets the Context for Your Application

Begin by creating a PL/SQL package with functions that set the context for your application. This section presents an example for creating the PL/SQL package, followed by a discussion of the syntax and behavior of the `SYS_CONTEXT` SQL function.

**Note:** A logon trigger can be used because the user's context (information such as `EMPNO`, `GROUP`, `MANAGER`) should be set before the user accesses any data.

#### SYS_CONTEXT Example

The following example creates the package `App_security_context`.

```
CREATE OR REPLACE PACKAGE App_security_context IS
   PROCEDURE Set_empno;
END;

CREATE OR REPLACE PACKAGE BODY App_security_context IS
   PROCEDURE Set_empno
   IS
   Emp_id NUMBER;
   BEGIN
    SELECT Empno INTO Emp_id FROM Emp
       WHERE Ename = SYS_CONTEXT('USERENV',
                                 'SESSION_USER');
    DBMS_SESSION.SET_CONTEXT('app_context', 'empno', Emp_id);
   END;
END;
```

> **See Also:** *PL/SQL Packages and Types Reference* for information about the DBMS_SESSION.SET_CONTEXT procedure.

## SYS_CONTEXT Syntax

The syntax for this function is:

SYS_CONTEXT ('*namespace*', '*attribute*', [*length*])

This function returns the value of attribute as defined in the package currently associated with the context namespace. It is evaluated once for each statement execution, and is treated like a constant during type checking for optimization. You can use the pre-defined namespace USERENV to access primitive contexts such as userid and Globalization Support parameters.

> **See Also:**
>
> - "Providing Access to Predefined Attributes through the USERENV Namespace" on page 13-11 for information about the USERENV application context namespace and a complete list of its predefined attributes.
> - *Oracle Database SQL Reference* for details about USERENV predefined attributes

## Using Dynamic SQL with SYS_CONTEXT

**Note:** This feature is applicable when COMPATIBLE is set to either 8.0 or 8.1.

During a session in which you expect a change in policy between executions of a given query, that query must use dynamic SQL. You must use dynamic SQL because static SQL and dynamic SQL parse statements differently.

- Static SQL statements are parsed at compile time. They are not reparsed at execution for performance reasons.
- Dynamic SQL statements are parsed every time they are executed.

Consider a situation in which policy A is in force when you compile a SQL statement—and then you switch to policy B and execute the statement. With static SQL, policy A remains in force: the statement is parsed at compile time and not reparsed upon execution. With dynamic SQL, the statement is parsed upon execution, and so the switch to policy B takes effect.

For example, consider the following policy:

```
EMPLOYEE_NAME = SYS_CONTEXT ('USERENV', 'SESSION_USER')
```

The policy "Employee name matches database user name" is represented in the form of a SQL predicate: the predicate is basically a policy. If the predicate changes, the statement must be reparsed in order to produce the correct result.

**See Also:** "Automatic Reparse" on page 14-46

## Using SYS_CONTEXT in a Parallel Query

If SYS_CONTEXT is used inside a SQL function which is embedded in a parallel query, the function picks up the application context.

Consider a user-defined function within a SQL statement, which sets the user's ID to 5:

```
CREATE FUNCTION proc1 AS RETURN NUMBER;
BEGIN
     IF SYS_CONTEXT ('hr', 'id') = 5
     THEN RETURN 1; ELSE RETURN 2;
     END
END;
```

Now consider the statement:

```
SELECT * FROM EMP WHERE proc1( ) = 1;
```

When this statement is run as a parallel query, the user session, which contains the application context information, is propagated to the parallel execution servers (query slave processes).

### Using SYS_CONTEXT with Database Links

Session-based local application context can be accessed by SQL statements within a user session by using the SYS_CONTEXT SQL function. When these SQL statements involve database links, then the SYS_CONTEXT SQL function is executed at the database link's initiating site and captures the context information there (on the initiating site).

If remote PL/SQL procedure calls are executed over a database link, then any SYS_CONTEXT function inside such a procedure is executed at the database link's destination site. In this case, only externally initialized application contexts are available at the database link's destination site. For security reasons, only the externally initialized application context information is propagated to the destination site from the initiating database link site.

## Task 2: Create a Unique Context and Associate It with the PL/SQL Package

To perform this task, use the CREATE CONTEXT statement. Each context must have a unique attribute and belong to a namespace. That is, context names must be unique within the database, not just within a schema. Contexts are always owned by the schema SYS.

For example:

```
CREATE CONTEXT order_entry USING App_security_context;
```

where order_entry is the context namespace, and App_security_context is the trusted package that can set attributes in the context namespace.

After you have created the context, you can set or reset the context attributes by using the DBMS_SESSION.SET_CONTEXT package. The values of the attributes you set remain either until you reset them, or until the user ends the session.

You can only set the context attributes inside the trusted procedure you named in the CREATE CONTEXT statement. This prevents a malicious user from changing context attributes without proper attribute validation.

Alternatively, you can use the Oracle Policy Manager graphical user interface to create a context and associate it with a PL/SQL package. Oracle Policy Manager, accessed from Oracle Enterprise Manager, enables you to apply policies to database objects and create application contexts. It also can be used to create and manage Oracle Label Security policies.

### Task 3: Set the Context Before the User Retrieves Data

Always use an event trigger on login to pull session information into the context. This sets the user's security-limiting attributes for the database to evaluate, and thus enables it to make the appropriate security decisions.

Other considerations come into play if you have a changing set of books, or if positions change constantly. In these cases, the new attribute values may not be picked up right away, and you must force a cursor reparse to pick them up.

### Task 4. Use the Context in a VPD Policy Function

Now that you have set up the context and the PL/SQL package, your VPD policy functions can use the application context to make policy decisions based on different context values.

## Examples: Application Context Within a Fine-Grained Access Control Function

This section provides three examples that use session-based application context within a fine-grained access control function.

- Example 1: Implementing the Policy
- Example 2: Controlling User Access by Way of an Application
- Example 3: Event Triggers, Application Context, Fine-Grained Access Control, and Encapsulation of Privileges

### Example 1: Implementing the Policy

This example uses application context to implement the policy, "Customers can see their own orders only."

This example guides you through the following steps in building the application:

- Step 1. Create a PL/SQL Package Which Sets the Context for the Application

- Step 2. Create an Application Context
- Step 3. Access the Application Context Inside the Package
- Step 4. Create the New Security Policy

The procedure in this example assumes a one-to-one relationship between users and customers. It finds the user's customer number (`Cust_num`), and caches the customer number in the application context. You can later refer to the `cust_num` attribute of your order entry context (`oe_ctx`) inside the security policy function.

Note that you could use a logon trigger to set the initial context.

## Step 1. Create a PL/SQL Package Which Sets the Context for the Application

Create the package as follows:

**Note:** You may need to set up the following data structures for the following examples to work:

```
CREATE TABLE apps.customers (cust_no NUMBER(4), cust_name
VARCHAR2(20));
CREATE TABLE scott.orders_tab (order_no NUMBER(4));
```

```
CREATE OR REPLACE PACKAGE apps.oe_ctx AS
   PROCEDURE set_cust_num;
END;

CREATE OR REPLACE PACKAGE BODY apps.oe_ctx AS
   PROCEDURE set_cust_num IS
     custnum NUMBER;
   BEGIN
      SELECT cust_no INTO custnum FROM customers WHERE cust_name =
         SYS_CONTEXT('USERENV', 'SESSION_USER');
     /* SET cust_num attribute in 'order_entry' context */
         DBMS_SESSION.SET_CONTEXT('order_entry', 'cust_num', custnum);
         DBMS_SESSION.SET_CONTEXT('order_entry', 'cust_num', custnum);
   END set_cust_num;
 END;
```

**Note:** This example does not treat error handling.

You can access predefined attributes—such as session user—by using SYS_CONTEXT('USERENV', *session_primitive*).

For more information, see Table 13–1, " Key to Predefined Attributes in USERENV Namespace" on page 13-12 and *Oracle Database SQL Reference*

### Step 2. Create an Application Context

Create an application context by entering:

```
CREATE CONTEXT Order_entry USING apps.oe_ctx;
```

Alternatively, you can use Oracle Policy Manager to create an application context.

### Step 3. Access the Application Context Inside the Package

Access the application context inside the package that implements the security policy on the database object.

**Note:** You may need to set up the following data structures for certain examples to work:

```
CREATE OR REPLACE PACKAGE Oe_security AS
FUNCTION Custnum_sec (D1 VARCHAR2, D2 VARCHAR2)
RETURN VARCHAR2;
END;
```

The package body appends a dynamic predicate to SELECT statements on the ORDERS_TAB table. This predicate limits the orders returned to those of the user's customer number by accessing the cust_num context attribute, instead of a subquery to the customers table.

```
CREATE OR REPLACE PACKAGE BODY Oe_security AS

/* limits select statements based on customer number: */
FUNCTION Custnum_sec (D1 VARCHAR2, D2 VARCHAR2) RETURN VARCHAR2
IS
    D_predicate VARCHAR2 (2000);
    BEGIN
     D_predicate := 'cust_no = SYS_CONTEXT(''order_entry'', ''cust_num'')';
```

```
      RETURN D_predicate;
    END Custnum_sec;
END Oe_security;
```

### Step 4. Create the New Security Policy

Create the policy as follows:

---

**Note:** You may need to set up the following data structures for certain examples to work:

```
CONNECT sys/xIcf1T9u AS sysdba;
CREATE USER secusr IDENTIFIED BY secusr;
```

---

```
BEGIN
DBMS_RLS.ADD_POLICY ('scott', 'orders_tab', 'oe_policy', 'secusr',
                     'oe_security.custnum_sec', 'select');
END;
```

This statement adds a policy named OE_POLICY to the ORDERS_TAB table for viewing in schema SCOTT. The SECUSR.OE_SECURITY.CUSTNUM_SEC function implements the policy, is stored in the SECUSR schema, and applies to SELECT statements only.

Now, any select statement by a customer on the ORDERS_TAB table automatically returns only that customer's orders. In other words, the dynamic predicate modifies the user's statement from this:

```
SELECT * FROM Orders_tab;
```

to this:

```
SELECT * FROM Orders_tab
   WHERE Custno = SYS_CONTEXT('order_entry','cust_num');
```

Note the following with regard to this example:

- In reality, you might have several predicates based on a user's position. For example, a sales representative would be able to see records only for his customers, and an order entry clerk would be able to see any customer order. You could expand the custnum_sec function to return different predicates based on the user's position context value.
- The use of application context in a fine-grained access control package effectively gives you a bind variable in a parsed statement. For example:

```
SELECT * FROM Orders_tab
   WHERE Custno = SYS_CONTEXT('order_entry', 'cust_num')
```

This is fully parsed and optimized, but the evaluation of the user's CUST_NUM attribute value for the ORDER_ENTRY context takes place at execution. This means that you get the benefit of an optimized statement which executes differently for each user who executes the statement.

**Note:** You can improve the performance of the function in this example even more by indexing CUST_NO.

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

  **See Also:**

  - Compare and contrast this example, which uses an application context within the dynamically generated predicate, with "How Fine-Grained Access Control Works" on page 14-29, which uses a subquery in the predicate
  - "Using Triggers" in *Oracle Database Application Developer's Guide - Fundamentals*
  - "Optimizing Performance by Enabling Static and Context Sensitive Policies" on page 14-38
  - "Adding Policies for Column-Level VPD" on page 14-40

## Example 2: Controlling User Access by Way of an Application

This example uses application context to control user access by way of a Human Resources application. It guides you through the following three tasks, each of which is described more fully in the following sections.

- Step 1. Create a PL/SQL Package to Set the Context
- Step 2. Create the Context and Associate It with the Package
- Step 3. Create the Initialization Script for the Application

In this example, assume that the application context for the Human Resources application is assigned to the HR_CTX namespace.

### Step 1. Create a PL/SQL Package to Set the Context

Create a PL/SQL package with a number of functions that set the context for the application

---

**Note:** You may need to set up the following data structures for certain examples to work:

```
DROP USER apps CASCADE;
CREATE USER apps IDENTIFIED BY welcome1;

CREATE OR REPLACE PACKAGE apps.hr_sec_ctx IS
   PROCEDURE set_resp_id (respid NUMBER);
   PROCEDURE set_org_id (orgid NUMBER);
  /* PROCEDURE validate_respid (respid NUMBER); */
  /* PROCEDURE validate_org_id (orgid NUMBER); */
END hr_sec_ctx;
```

---

APPS is the schema owning the package.

```
CREATE OR REPLACE PACKAGE BODY apps.hr_sec_ctx IS
/* function to set responsibility id */
PROCEDURE set_resp_id (respid NUMBER) IS
BEGIN

/* validate respid based on primitive and other context */
/*    validate_respid (respid); */
/* set resp_id attribute under namespace 'hr_ctx'*/

    DBMS_SESSION.SET_CONTEXT('hr_ctx', 'resp_id', respid);
END set_resp_id;

/* function to set organization id */
PROCEDURE set_org_id (orgid NUMBER) IS
BEGIN

/* validate organization ID */
/*    validate_org_id(orgid); /*
/* set org_id attribute under namespace 'hr_ctx' */

    DBMS_SESSION.SET_CONTEXT('hr_ctx', 'org_id', orgid);
END set_org_id;

/* more functions to set other attributes for the HR application */
```

```
END hr_sec_ctx;
```

### Step 2. Create the Context and Associate It with the Package

For example:

```
CREATE CONTEXT Hr_ctx USING apps.hr_sec_ctx;
```

### Step 3. Create the Initialization Script for the Application

Suppose that the execute privilege on the package HR_SEC_CTX has been granted to the schema running the application. Part of the script will make calls to set various attributes of the HR_CTX context. Here, we do not show how the context is determined. Normally, it is based on the primitive context or other derived context.

```
APPS.HR_SEC_CTX.SET_RESP_ID(1);
APPS.HR_SEC_CTX.SET_ORG_ID(101);
```

The SYS_CONTEXT function can be used for data access control based on this application context. For example, the base table HR_ORGANIZATION_UNIT can be secured by a view that restricts access to rows based on attribute ORG_ID:

**Note:** You may need to set up data structures for certain examples to work:

```
CREATE TABLE hr_organization_unit (organization_id NUMBER);
```

```
CREATE VIEW Hr_organization_secv AS
   SELECT * FROM hr_organization_unit
      WHERE Organization_id = SYS_CONTEXT('hr_ctx','org_id');
```

## Example 3: Event Triggers, Application Context, Fine-Grained Access Control, and Encapsulation of Privileges

This example illustrates use of the following security features in Oracle Database:

- Event triggers
- Application context (session-based)
- Fine-grained access control
- Encapsulation of privileges in stored procedures

In this example, we associate a security policy with the table called DIRECTORY which has the following columns:

| Column | Description |
|---|---|
| EMPNO | identification number for each employee |
| MGRID | employee identification number for the manager of each employee |
| RANK | position of the employee in the corporate hierarchy |

**Note:** You may need to set up the following data structures for certain examples to work:

```
CREATE TABLE Payroll(
   Srate  NUMBER,
   Orate  NUMBER,
   Acctno NUMBER,
   Empno  NUMBER,
   Name   VARCHAR2(20));
CREATE TABLE Directory_u(
   Empno NUMBER,
   Mgrno NUMBER,
   Rank  NUMBER);
CREATE SEQUENCE Empno_seq;
CREATE SEQUENCE Rank_seq;
```

The security policy associated with this table has two elements:

- All users can find the MGRID for a specific EMPNO. To implement this, we create a definer's right package in the human resources schema (HR) to perform SELECT on the table.
- Managers can update the positions in the corporate hierarchy of only their direct subordinates. To do this they must use only the designated application. You can implement this as follows:
    - Define fine-grained access control policies on the table based on EMPNO and application context.
    - Set EMPNO by using a logon trigger.

* Set the application context by using the designated package for processing the updates (event triggers and application context).

---

**Note:** In this example, we grant UPDATE privileges on the table to PUBLIC, because fine-grained access control prevents an unauthorized user from wrongly modifying a given row.

---

```
CONNECT system/yJdg2U1v AS sysdba
GRANT CONNECT,RESOURCE,UNLIMITED TABLESPACE,CREATE ANY CONTEXT, CREATE
PROCEDURE, CREATE ANY TRIGGER TO HR IDENTIFIED BY HR;
CONNECT hr/hr;
CREATE TABLE Directory (Empno   NUMBER(4) NOT NULL,
                        Mgrno   NUMBER(4) NOT NULL,
                        Rank    NUMBER(7,2) NOT NULL);

CREATE TABLE Payroll (Empno  NUMBER(4) NOT NULL,
                      Name   VARCHAR(30) NOT NULL );

/* seed the tables with a couple of managers: */
INSERT INTO Directory VALUES (1, 1, 1.0);
INSERT INTO Payroll VALUES (1, 'KING');
INSERT INTO Directory VALUES (2, 1, 5);
INSERT INTO Payroll VALUES (2, 'CLARK');

/* Create the sequence number for EMPNO: */
CREATE SEQUENCE Empno_seq START WITH 5;

/* Create the sequence number for RANK:  */
CREATE SEQUENCE Rank_seq START WITH 100;

CREATE OR REPLACE CONTEXT Hr_app USING Hr.Hr0_pck;
CREATE OR REPLACE CONTEXT Hr_sec USING Hr.Hr1_pck;

CREATE or REPLACE PACKAGE Hr0_pck IS
PROCEDURE adjustrankby1(Empno NUMBER);
END;

CREATE or REPLACE PACKAGE BODY Hr0_pck IS
/* raise the rank of the empno by 1:  */
PROCEDURE Adjustrankby1(Empno NUMBER)
IS
   Stmt   VARCHAR2(100);
   BEGIN
   /*Set context to indicate application state */
   DBMS_SESSION.SET_CONTEXT('hr_app','adjstate',1);
   /* Now we can issue DML statement:  */
   Stmt := 'UPDATE Directory d SET Rank = Rank + 1 WHERE d.Empno = '
   || Empno;
   EXECUTE IMMEDIATE STMT;

/* Re-set application state: */
   DBMS_SESSION.SET_CONTEXT('hr_app','adjstate',0);
   END;
END;

CREATE or REPLACE PACKAGE hr1_pck IS PROCEDURE setid;
END;
/
/* Based on userid, find EMPNO, and set it in application context */

CREATE or REPLACE PACKAGE BODY Hr1_pck IS
PROCEDURE setid
  IS
  id NUMBER;
  BEGIN
    SELECT Empno INTO id FROM Payroll WHERE Name =
      SYS_CONTEXT('userenv','session_user') ;
    DBMS_SESSION.SET_CONTEXT('hr_sec','empno',id);
    DBMS_SESSION.SET_CONTEXT('hr_sec','appid',id);
  EXCEPTION
    /* For purposes of demonstration insert into payroll table
    /  so that user can continue on and run example. */
    WHEN NO_DATA_FOUND THEN
      INSERT INTO Payroll (Empno, Name)
         VALUES (Empno_seq.NEXTVAL, SYS_CONTEXT('userenv','session_user'));
      INSERT INTO Directory (Empno, Mgrno, Rank)
         VALUES (Empno_seq.CURRVAL, 2, Rank_seq.NEXTVAL);
      SELECT Empno INTO id FROM Payroll WHERE Name =
        sys_context('userenv','session_user') ;
      DBMS_SESSION.SET_CONTEXT('hr_sec','empno',id);
      DBMS_SESSION.SET_CONTEXT('hr_sec','appid',id);
    WHEN OTHERS THEN
      NULL;
    /* If this is to be fired by using a "logon" trigger,
    /  you need to handle exceptions if you want the user to continue
    /  logging into the database. */
  END;
```

```
END;

GRANT EXECUTE ON Hr1_pck TO public;

CONNECT system/yJdg2U1v AS sysdba

CREATE OR REPLACE TRIGGER Databasetrigger

AFTER LOGON
ON DATABASE
BEGIN
   hr.Hr1_pck.Setid;
END;

/* Creates the package for finding the MGRID for a particular EMPNO
using definer's right (encapsulated privileges). Note that users are
granted EXECUTE privileges only on this package, and not on the table
(DIRECTORY) it is querying. */

CONNECT hr/hr

CREATE or REPLACE PACKAGE hr2_pck IS
   FUNCTION Findmgr(Empno NUMBER) RETURN NUMBER;
END;

CREATE or REPLACE PACKAGE BODY hr2_pck IS
   /* insert a new employee record: */
   FUNCTION findmgr(empno number) RETURN NUMBER IS
   Mgrid NUMBER;
   BEGIN
      SELECT mgrno INTO mgrid FROM directory WHERE mgrid = empno;
   RETURN mgrid;
   END;
END;

CREATE or REPLACE FUNCTION secure_updates(ns varchar2,na varchar2)
  RETURN VARCHAR2 IS
     Results VARCHAR2(100);
    BEGIN
       /* Only allow updates when designated application has set the session
       state to indicate we are inside it. */
       IF (sys_context('hr_app','adjstate') = 1)
          THEN results := 'mgrno = SYS_CONTEXT("hr_sec","empno")';
       ELSE results := '1=2';
      END IF;
      RETURN Results;
   END;

/* Attaches fine-grained access policy to all update operations on
hr.directory */

CONNECT system/yJdg2U1v AS sysdba;
BEGIN
   DBMS_RLS.ADD_POLICY('hr','directory','secure_update','hr',
                       'secure_updates','update',TRUE,TRUE);
END;
```

## Initializing Application Context Externally

This feature lets you specify a special type of namespace that accepts initialization of attribute values from external resources and stores them in the user's local session. This enhances performance and enables the automatic propagation of attributes from one session to the other. Only those application contexts initialized from OCI-based external sources support connected user database links.

This section contains these topics:

- Obtaining Default Values from Users
- Obtaining Values from Other External Resources

### Obtaining Default Values from Users

Sometimes it is desirable to obtain default values from users. Initially, these default values may serve as hints or preferences, and then after validation become trusted contexts. Similarly, it may be more convenient for clients to initialize some default values, and then rely on a login event trigger or applications to validate the values.

For job queues, the job submission routine records the context being set at the time the job is submitted, and restores it when executing the batched job. To maintain the integrity of the context, job queues cannot bypass the designated PL/SQL package to set the context. Rather, externally initialized application context accepts initialization of context values from the job queue process.

Automatic propagation of context to a remote session may create security problems. Developers or administrators can effectively handle this type of context that takes default values from resources other than the designated PL/SQL procedure by using logon triggers to reset the context when users logon.

## Obtaining Values from Other External Resources

In addition to using the designated trusted package, externally initialized application context can also accept initialization of attributes and values through external resources such as an OCI interface, a job queue process, or a database link. It provides:

- For remote sessions, automatic propagation of context values that are in the externally initialized context namespace
- For job queues, restoration of context values that are in the externally initialized context namespace
- For OCI interfaces, a mechanism to initialize context values that are in the externally initialized context namespace

Although this type of namespace can be initialized by any client program using OCI, there are login event triggers that can verify the values. It is up to the application to interpret and trust the values of the attributes.

Middle-tier servers can actually initialize context values on behalf of database users. Context attributes are propagated for the remote session at initialization time, and the remote database accepts the values if the namespace is externally initialized.

**See Also:**

*Oracle Database JDBC Developer's Guide and Reference*

*Oracle Call Interface Programmer's Guide*

## Initializing Application Context Globally

This feature uses a centralized location to store the user's application context, enabling applications to set up the user's contexts during initialization based upon the user's identity. In particular, it supports Oracle Label Security labels and privileges. This feature makes it much easier for the administrator to manage contexts for large numbers of users and databases. For example, many organizations want to manage user information centrally, in an LDAP-based directory. Enterprise User Security, a feature of Oracle Advanced Security, supports centralized user and authorization management in Oracle Internet Directory. However, there may be additional attributes an application wishes to retrieve from LDAP to use for VPD enforcement, such as the user's title, organization, or physical location.

This section contains these topics:

- Application Context Utilizing LDAP
- How Globally Initialized Application Context Works
- Example: Initializing Application Context Globally

## Application Context Utilizing LDAP

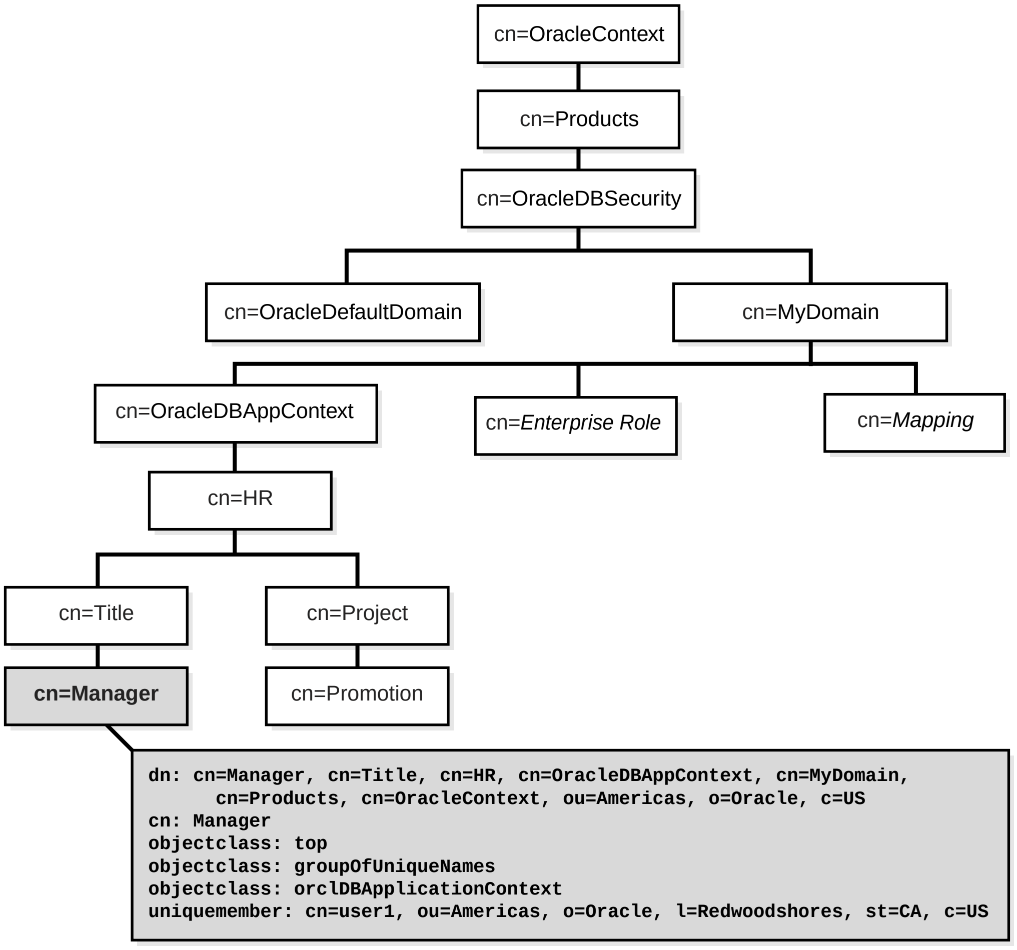
Session-based application context initialized globally utilizes the Lightweight Directory Access Protocol (LDAP). LDAP is a standard, extensible, and efficient directory access protocol. The LDAP directory stores a list of users to which this application is assigned. An Oracle database server can use Oracle Internet Directory, or third-party directories such as Microsoft Active Directory and Sun Microsystems iPlanet, as the directory service for authentication and authorization of enterprise users. (Enterprise User Security requires Oracle Advanced Security.)

The LDAP object `orclDBApplicationContext` (a subclass of `groupOfUniqueNames`) has been defined to store the application context values in the directory. The location of the application context object is described in Figure 14–1, which is based upon the Human Resources example.

An internal C function is required to retrieve the `orclDBApplicationContext` value, which returns a list of application context values to the RDBMS.

**Note:** In this example, HR is the namespace, Title and Project are the attributes, and Manager and Promotion are the values.

***Figure 14–1 Location of Application Context in LDAP Directory Information Tree (DIT)***



```
dn: cn=Manager, cn=Title, cn=HR, cn=OracleDBAppContext, cn=MyDomain,
      cn=Products, cn=OracleContext, ou=Americas, o=Oracle, c=US
cn: Manager
objectclass: top
objectclass: groupOfUniqueNames
objectclass: orclDBApplicationContext
uniquemember: cn=user1, ou=Americas, o=Oracle, l=Redwoodshores, st=CA, c=US
```

## How Globally Initialized Application Context Works

The administrator configures Enterprise User Security, a feature of Oracle Advanced Security. Then she sets up the user's application context values in the database and the directory.

When a global user (enterprise user) connects to the database, the Oracle Advanced Security Enterprise User Security feature performs authentication to verify the identity of the user connecting to the database. After authentication, the user's global roles and application context are retrieved from the directory. When the user logs on to the database, her global roles and initial application context are already set up.

> **See Also:** *Oracle Advanced Security Administrator's Guide* for a complete discussion of Enterprise User Security and how to configure this feature.

## Example: Initializing Application Context Globally

The initial application context for a user, such as department name and title, can be set up and stored in the LDAP directory. The values are retrieved during user login so that the context is set properly. In addition, any information related to the user is retrieved and stored in the application context namespace SYS_USER_DEFAULTS. The following example shows how this is done.

1. Create an application context in the database.

   ```
   CREATE CONTEXT HR USING hrapps.hr_manage_pkg INITIALIZED GLOBALLY;
   ```

2. Create and add new entries in the LDAP directory.

   An example of the entries added to the LDAP directory follows. These entries create an attribute name `Title` with attribute value `Manager` for the application (namespace) HR, and assign usernames `user1` and `user2`.

   ```
   dn:
   cn=OracleDBAppContext,cn=myDomain,cn=OracleDBSecurity,cn=Products,cn=OracleC
   ontext,ou=Americas,o=oracle,c=US
   changetype: add
   cn: OracleDBAppContext
   objectclass: top
   objectclass: orclContainer

   dn:
   cn=HR,cn=OracleDBAppContext,cn=myDomain,cn=OracleDBSecurity,cn=Products,cn=O
   racleContext,ou=Americas,o=oracle,c=US
   ```

```
changetype: add
cn: HR
objectclass: top
objectclass: orclContainer

dn:
cn=Title,cn=HR,cn=OracleDBAppContext,cn=myDomain,cn=OracleDBSecurity,cn=Prod
ucts,cn=OracleContext,ou=Americas,o=oracle,c=US
changetype: add
cn: Title
objectclass: top
objectclass: orclContainer

dn:
cn=Manager,cn=Title,cn=HR,cn=OracleDBAppContext,cn=myDomain,cn=OracleDBSecur
ity,cn=Products,cn=OracleContext,ou=Americas,o=oracle,c=US
cn: Manager
objectclass: top
objectclass: groupofuniquenames
objectclass: orclDBApplicationContext
uniquemember: CN=user1,OU=Americas,O=Oracle,L=Redwoodshores,ST=CA,C=US
uniquemember: CN=user2,OU=Americas,O=Oracle,L=Redwoodshores,ST=CA,C=US
```

3. If an LDAP `inetOrgPerson` object entry exists for the user, the connection will also retrieve all the attributes from `inetOrgPerson` and assign them to the namespace SYS_LDAP_USER_DEFAULT. The following is an example of an `inetOrgPerson` entry:

```
dn: cn=user1,ou=Americas,O=oracle,L=redwoodshores,ST=CA,C=US
changetype: add
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
cn: user1
sn: One
givenName: User
initials: UO
title: manager, product development
uid: uone
mail: uone@us.oracle.com
telephoneNumber: +1 650 123 4567
employeeNumber: 00001
employeeType: full time
```

4. Connect to the database.

   When `user1` connects to a database that belongs to domain `myDomain`, `user1` will have his `Title` set to `Manager`. Any information related to `user1` will be retrieved from the LDAP directory. The value can be obtained using the syntax

   ```
   SYS_CONTEXT('namespace','attribute name')
   ```

   For example:

   ```
   DECLARE
   tmpstr1 VARCHAR2(30);
   tmpstr2 VARCHAR2(30);
   BEGIN
   tmpstr1 = SYS_CONTEXT('HR','TITLE);
   tmpstr2 = SYS_CONTEXT('SYS_LDAP_USER_DEFAULT','telephoneNumber');
   DBMS_OUTPUT.PUT_LINE('Title is ' || tmpstr1);
   DBMS_OUTPUT.PUT_LINE('Telephone Number is ' || tmpstr2);
   END;
   ```

   The output of the preceding example is:

   ```
   Title is Manager
   Telephone Number is +1 650 123 4567
   ```

## How to Use Global Application Context

Global application context stores context information in the SGA so it can be used for applications which use a sessionless model, such as middle-tier applications in a three-tiered architecture. These applications cannot use session-based application context because users authenticate to the application and then it typically connects to the database as a single identity. Global application context uses the `CLIENT_IDENTIFIER` USERENV namespace attribute, set with the `DBMS_SESSION` interface, to associate the database session with a particular user or group. The following sections explain how to use the DBMS_SESSION interface to set the `CLIENT_IDENTIFIER` and then examples are provided:

- Using the DBMS_SESSION Interface to Manage Application Context in Client Sessions
- Examples: Global Application Context

**See Also:** "Introduction to Global Application Context" on page 13-17 for conceptual information about this feature.

## Using the DBMS_SESSION Interface to Manage Application Context in Client Sessions

The DBMS_SESSION interface for managing application context has a client identifier for each application context. In this way, application context can be managed globally, yet each client sees only his or her assigned application context. The following interfaces in DBMS_SESSION enable the administrator to manage application context in client sessions:

- SET_CONTEXT
- CLEAR_CONTEXT
- CLEAR_ALL_CONTEXT (can also be used with session-based application context)
- SET_IDENTIFIER
- CLEAR_IDENTIFIER

The middle-tier application server can use SET_CONTEXT to set application context for a specific client ID. Then, when assigning a database connection to process the client request, the application server needs to issue a SET_IDENTIFIER to denote the ID of the application session. From then on, every time the client invokes SYS_CONTEXT, only the context that was associated with the set identifier is returned.

**See Also:**

- *PL/SQL Packages and Types Reference* for reference information and a complete description of the DBMS_SESSION package.
- "Using CLIENT_IDENTIFIER Independent of Global Application Context" on page 15-12 for information about setting this USERENV namespace attribute with the DBMS_SESSION interface.

## Examples: Global Application Context

This section provides two examples that use global application context.

- Example 1: Global Application Context
- Example 2: Global Application Context for Lightweight Users

### Example 1: Global Application Context

The following steps outline the global application context process:

1. Consider the application server, AppSvr, that has assigned the client identifier 12345 to client SCOTT. It then issues the following statement to indicate that, for this client identifier, there is an application context RESPONSIBILITY with a value of 13 in the HR namespace.

   ```
   DBMS_SESSION.SET_CONTEXT( 'HR', 'RESPONSIBILITY' , '13', 'SCOTT', '12345' );
   ```

   Note that HR must be a global context namespace created as follows:

   ```
   CREATE CONTEXT hr USING hr.init ACCESSED GLOBALLY;
   ```

2. Then, the following command is issued to indicate the connecting client's identity each time SCOTT uses AppSvr to connect to the database:

   ```
   DBMS_SESSION.SET_IDENTIFIER('12345');
   ```

3. When there is a SYS_CONTEXT('HR','RESPONSIBILITY') call within the database session, the database engine matches the client identifier 12345 to the global context, and returns the value 13.
4. When exiting this database session, AppSvr clears the client identifier by issuing:

   ```
   DBMS_SESSION.CLEAR_IDENTIFIER( );
   ```

After a session's client identifier is cleared, it takes on a NULL value. This implies that subsequent SYS_CONTEXT calls only retrieve application contexts with NULL client identifiers, until the client identifier is set again using the SET_IDENTIFIER interface.

### Example 2: Global Application Context for Lightweight Users

The following steps outline the global application context process for a lightweight user application:

1. The administrator creates the global context namespace by issuing:

   ```
   CREATE CONTEXT hr USING hr.init ACCESSED GLOBALLY;
   ```

2. The HR application server (AppSvr) starts up and establishes multiple connections to the HR database as user APPSMGR.
3. User SCOTT logs on to the HR application server.
4. AppSvr authenticates SCOTT to the application.
5. AppSvr assigns a temporary session ID (or simply uses the application user ID), 12345, for this connection.
6. The session ID is returned to SCOTT's browser as part of a cookie or maintained by AppSvr.

   **Note:** If the application generates a session ID for use as a CLIENT_IDENTIFIER, the session ID must be suitably random, and protected over the network through encryption. If the session ID is not random, then a malicious user could guess the session ID and access another user's data. If the session ID is not encrypted over the network, then a malicious user could retrieve the session ID and access the connection.

7. AppSvr initializes application context for this client calling the HR.INIT package, which issues:

   ```
   DBMS_SESSION.SET_CONTEXT( 'hr', 'id', 'scott', 'APPSMGR', 12345 );
   DBMS_SESSION.SET_CONTEXT( 'hr', 'dept', 'sales', 'APPSMGR', 12345 );
   ```

8. AppSvr assigns a database connection to this session, and initializes the session by issuing:

   ```
   DBMS_SESSION.SET_IDENTIFIER( 12345 );
   ```

9. All SYS_CONTEXT calls within this database session will return application context values belonging to the client session only. For example, SYS_CONTEXT('hr','id') will return the value SCOTT.

10. When done with the session, AppSvr can issue the following statement to clean up the client identity:

    ```
    DBMS_SESSION.CLEAR_IDENTIFIER ( );
    ```

Note that even if another database user (ADAMS) had logged into the database, he cannot access the global context set by AppSvr because AppSvr has specified that only the application with logged in user APPSMGR can see it. If AppSvr has used the following, then any user session with client ID set to 12345 can see the global context.

```
DBMS_SESSION.SET_CONTEXT( 'hr', 'id', 'scott', NULL , 12345 );
DBMS_SESSION.SET_CONTEXT( 'hr', 'dept', 'sales', NULL , 12345 );
```

This approach enables different users to share the same context.

Users should be aware of the security implication of different settings of the global context. NULL in the username means that any user can access the global context. A NULL client ID in the global context means that only a session with an uninitialized client ID can access the global context.

Users can query the client identifier set in the session as follows:

```
SYS_CONTEXT('USERENV','CLIENT_IDENTIFIER')
```

The DBA can see which sessions have the client identifier set by querying the V$SESSION view's CLIENT_IDENTIFIER and USERNAME.

When a user wants to see how much global context area (in bytes) is being used, she can use SYS_CONTEXT('USERENV','GLOBAL_CONTEXT_MEMORY')

**See Also:** For more information about using the CLIENT_IDENTIFIER predefined attribute of the USERENV application context:

- "Using the CLIENT_IDENTIFIER Attribute to Preserve User Identity" on page 15-12
- *Oracle Database SQL Reference*
- *PL/SQL Packages and Types Reference*
- *Oracle Database JDBC Developer's Guide and Reference*
- *Oracle Call Interface Programmer's Guide*

## How Fine-Grained Access Control Works

Fine-grained access control is based on dynamically modified statements. Suppose you want to attach to the ORDERS_TAB table the following security policy: "Customers can see only their own orders." The process is described in this section.

1. Create a function to add a predicate to a user's DML statement.

   **Note:** A predicate is the WHERE clause (a selection criterion clause) based on one of the operators (=, !=, IS, IS NOT, >, >=, EXIST, BETWEEN, IN, NOT IN, and so on). For a complete list of operators, see the *Oracle Database SQL Reference*

   In this case, you might create a function that adds the following predicate:

   ```
    Cust_no = (SELECT Custno FROM Customers WHERE Custname =
              SYS_CONTEXT ('userenv','session_user'))
   ```

2. A user enters the statement:

   ```
   SELECT * FROM Orders_tab;
   ```

3. The Oracle database server calls the function you created to implement the security policy.
4. The function dynamically modifies the user's statement to read:

   ```
   SELECT * FROM Orders_tab WHERE Custno = (
      SELECT Custno FROM Customers
          WHERE Custname = SYS_CONTEXT('userenv', 'session_user'))
   ```

5. The Oracle database server executes the dynamically modified statement.

Upon execution, the function employs the username returned by SYS_CONTEXT ('userenv','session_user') to look up the corresponding customer and to limit the data returned from the ORDERS_TAB table to that customer's data only.

**See Also:** For more information on using fine-grained access control:

- "Introduction to Fine-Grained Access Control" on page 13-6
- "Introduction to Global Application Context" on page 13-17
- *PL/SQL Packages and Types Reference.*

## How to Establish Policy Groups

A policy group is a set of security policies which belong to an application. You can designate an application context (known as a driving context) to indicate the policy group in effect. Then, when the table, view, or synonym column is accessed, the server looks up the driving context (which are also known as policy contexts) to determine the policy group in effect. It enforces all the associated policies which belong to that policy group.

This section contains the following topics:

- The Default Policy Group: SYS_DEFAULT
- New Policy Groups
- How to Implement Policy Groups
- Validation of the Application Used to Connect

### The Default Policy Group: SYS_DEFAULT

In the Oracle Policy Manager tree structure, the Fine-Grained Access Control Policies folder contains the Policy Groups folder. The Policy Groups folder contains an icon for each policy group, as well as an icon for the SYS_DEFAULT policy group.

By default, all policies belong to the SYS_DEFAULT policy group. Policies defined in this group for a particular table, view, or synonym will always be executed along with the policy group specified by the driving context. The SYS_DEFAULT policy group may or may not contain policies. If you attempt to drop the SYS_DEFAULT policy group, an error will be raised.

If, to the SYS_DEFAULT policy group, you add policies associated with two or more objects, then each such object will have a separate SYS_DEFAULT policy group associated with it. For example, the EMP table in the SCOTT schema has one SYS_DEFAULT policy group, and the DEPT table in the SCOTT schema has a different

SYS_DEFAULT policy group associated with it. These are displayed in the tree structure as follows:

```
SYS_DEFAULT
  - policy1 (SCOTT/EMP)
  - policy3 (SCOTT/EMP)
SYS_DEFAULT
  - policy2 (SCOTT/DEPT)
```

**Note:** Policy groups with identical names are supported. When you select a particular policy group, its associated schema and object name are displayed in the property sheet on the right-hand side of the screen.

## New Policy Groups

When adding the policy to a table, view, or synonym, you can use the DBMS_RLS.ADD_GROUPED_POLICY interface to specify the group to which the policy belongs. To specify which policies will be effective, you add a driving context using the DBMS_RLS.ADD_POLICY_CONTEXT interface. If the driving context returns an unknown policy group, an error is returned.

If the driving context is not defined, then all policies are executed. Likewise, if the driving context is NULL, then policies from all policy groups are enforced. In this way, an application accessing the data cannot bypass the security setup module (which sets up application context) to avoid any applicable policies.

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

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

## How to Implement Policy Groups

To create policy groups, the administrator must do two things:

- Set up a driving context to identify the effective policy group.
- Add policies to policy groups, as required.

The following example shows how to perform these tasks.

---

**Note:** You need to set up the following data structures for the examples in this section to work:

```
DROP USER finance CASCADE;
CREATE USER finance IDENTIFIED BY welcome2;
GRANT RESOURCE TO apps;
DROP TABLE apps.benefit;
CREATE TABLE apps.benefit (c NUMBER);
```

---

### Step 1: Set Up a Driving Context

Begin by creating a namespace for the driving context. For example:

```
CREATE CONTEXT appsctx USING apps.apps_security_init;
```

Create the package that administers the driving context. For example:

```
CREATE OR REPLACE PACKAGE apps.apps_security_init IS
PROCEDURE setctx (policy_group VARCHAR2);
END;

CREATE OR REPLACE PACKAGE BODY apps.apps_security_init AS
PROCEDURE setctx ( policy_group varchar2 ) IS
BEGIN

REM  Do some checking to determine the current application.
REM  You can check the proxy if using the proxy authentication feature.
REM  Then set the context to indicate the current application.
.
.
.
DBMS_SESSION.SET_CONTEXT('APPSCTX','ACTIVE_APPS', policy_group);
END;
END;
```

Define the driving context for the table `APPS.BENEFIT`.

```
BEGIN
DBMS_RLS.ADD_POLICY_CONTEXT('apps','benefit','APPSCTX','ACTIVE_APPS');
END;
```

### Step 2: Add a Policy to the Default Policy Group.

Create a security function to return a predicate to divide the data by company.

```
CREATE OR REPLACE FUNCTION by_company (sch varchar2, tab varchar2)
RETURN VARCHAR2 AS
BEGIN
  RETURN 'COMPANY = SYS_CONTEXT(''ID'',''MY_COMPANY'')';
END;
```

Since policies in SYS_DEFAULT are always executed (except for SYS, or users with the EXEMPT ACCESS POLICY system privilege), this security policy (named SECURITY_BY_COMPANY), will always be enforced regardless of the application running. This achieves the universal security requirement on the table: namely, that each company should see its own data, regardless of the application running. The function APPS.APPS_SECURITY_INIT.BY_COMPANY returns the predicate to make sure that you can only see your company's data.

```
BEGIN
DBMS_RLS.ADD_GROUPED_POLICY('apps','benefit','SYS_DEFAULT',
'security_by_company',
'apps','by_company');
END;
```

### Step 3: Add a Policy to the HR Policy Group

First, create the HR group:

```
CREATE OR REPLACE FUNCTION hr.security_policy
RETURN VARCHAR2
AS
BEGIN
 RETURN 'SYS_CONTEXT(''ID'',''TITLE'') = ''MANAGER'' ';
END;
```

The following creates the policy group and adds a policy named HR_SECURITY to the HR policy group. The function HR.SECURITY_POLICY returns the predicate to enforce HR's security on the table APPS.BENEFIT:

```
BEGIN
DBMS_RLS.CREATE_POLICY_GROUP('apps','benefit','HR');
DBMS_RLS.ADD_GROUPED_POLICY('apps','benefit','HR',
'hr_security','hr','security_policy');
END;
```

### Step 4: Add a Policy to the FINANCE Policy Group

Create the `FINANCE` policy:

```
CREATE OR REPLACE FUNCTION finance.security_policy
RETURN VARCHAR2
AS
BEGIN
 RETURN ('SYS_CONTEXT(''ID'',''DEPT'') = ''FINANCE'' ');
END;
```

Create a policy group named `FINANCE` and add the `FINANCE` policy to the `FINANCE` group:

```
BEGIN
DBMS_RLS.CREATE_POLICY_GROUP('apps','benefit','FINANCE');
DBMS_RLS.ADD_GROUPED_POLICY('apps','benefit','FINANCE',
'finance_security','finance', 'security_policy');
END;
```

As a result, when the database is accessed, the application initializes the driving context after authentication. For example, with the `HR` application:

```
execute apps.security_init.setctx('HR');
```

## Validation of the Application Used to Connect

The package implementing the driving context must correctly validate the application which is being used. Although the database always ensures that the package implementing the driving context sets context attributes (by checking the call stack), this cannot protect against inadequate validation within the package.

For example, in applications where database users or enterprise users are known to the database, the user needs `EXECUTE` privilege on the package which sets the driving context. Consider a user who knows that:

- The company's `BENEFITS` application allows more liberal access than its `HR` application, and
- The `setctx` procedure (which sets the correct policy group within the driving context) does not perform any validation to determine which application is actually connecting. That is, the procedure does not check the IP address of the

incoming connection (for a three-tier system), or the `proxy_user` attribute of the user session.

In this situation, the user could pass to the driving context package an argument which sets the context to the more liberal `BENEFITS` policy group even though this user will access the `HR` application. In this way the user can bypass the more restrictive security policy because the package inadequately validates the application.

By contrast, if you implement proxy authentication with VPD, then you can determine the identity of the middle tier (and the application) which is actually connecting to the database on a user's behalf. In this way, the correct policy will be applied for each application to mediate data access. For example, a developer using the proxy authentication feature could determine that the application (the middle tier) connecting to the database is `HRAPPSERVER`. The package which implements the driving context can thus verify that the `proxy_user` in the user session is `HRAPPSERVER` before setting the driving context to use the `HR` policy group, or can disallow access if `proxy_user` is not `HRAPPSERVER`.

In this case, when the following query is executed

```
SELECT * FROM APPS.BENEFIT;
```

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

```
SELECT * FROM APPS.BENEFIT WHERE COMPANY = SYS_CONTEXT('ID','MY_COMPANY') and
SYS_CONTEXT('ID','TITLE') = 'MANAGER';
```

## How to Add a Policy to a Table, View, or Synonym

The `DBMS_RLS` package enables you to administer security policies. This package's procedures allow you to specify the table, view, or synonym to which you are adding a policy and various data pertinent to that policy. These data include the names of the policy, the policy group, the function implementing the policy, and the type of statement the policy controls (`SELECT`, `INSERT`, `UPDATE`, `DELETE`, `CREATE INDEX`, or `ALTER INDEX`). Table 14–2 lists these procedures.

***Table 14–2 DBMS_RLS Procedures***

| Procedure | Purpose |
|---|---|
| `DBMS_RLS.ADD_POLICY` | Use this procedure to add a policy to a table, view, or synonym. |

***Table 14–2 (Cont.) DBMS_RLS Procedures***

| Procedure | Purpose |
|---|---|
| `DBMS_RLS.DROP_POLICY` | Use this procedure to drop a policy from a table, view, or synonym. |
| `DBMS_RLS.REFRESH_POLICY` | Use this procedure to invalidate cursors associated with non-static policies. |
| `DBMS_RLS.ENABLE_POLICY` | Use this procedure to enable (or disable) a policy you previously added to a table, view, or synonym. |
| `DBMS_RLS.CREATE_POLICY_GROUP` | Use this procedure to create a policy group. |
| `DBMS_RLS.ADD_GROUPED_POLICY` | Use this procedure to add a policy to the specified policy group. |
| `DBMS_RLS.ADD_POLICY_CONTEXT` | Use this procedure to add the context for the active application. |
| `DBMS_RLS.DELETE_POLICY_GROUP` | Use this procedure to drop a policy group. |
| `DBMS_RLS.DROP_GROUPED_POLICY` | Use this procedure to drop a policy which is a member of the specified group. |
| `DBMS_RLS.DROP_POLICY_CONTEXT` | Use this procedure to drop the context for the application. |
| `DBMS_RLS.ENABLE_GROUPED_POLICY` | Use this procedure to enable a policy within a group. |
| `DBMS_RLS.DISABLE_GROUPED_POLICY` | Use this procedure to disable a policy within a group. |
| `DBMS_RLS.REFRESH_GROUPED_POLICY` | Use this procedure to reparse the SQL statements associated with a refreshed policy. |

**See Also:** *PL/SQL Packages and Types Reference* for information about using the DBMS_RLS package and all of its procedures and parameters.

Alternatively, you can use Oracle Policy Manager to administer security policies.

## DBMS_RLS.ADD_POLICY Procedure Policy Types

The execution of policy functions can consume a significant amount of system resources. If you can minimize the number of times policy functions must execute, then you can optimize your database server's performance. To avoid unnecessary policy function execution, you can choose from five different policy types, which enable you to precisely specify how and how often a policy predicate should change. You can enable these different types of policies, which are listed in Table 14–3 on page 14-37, by setting the `policy_type` parameter of the `DBMS_RLS.ADD POLICY` procedure.

***Table 14–3 DBMS_RLS.ADD_POLICY Policy Types At a Glance***

| Policy Types | When Policy Function Executes... | Usage Example | Shared Across Multiple Objects? |
|---|---|---|---|
| `STATIC` | Once, then the predicate is cached in the SGA.[1] | View replacement | No |
| `SHARED_STATIC` | Same as `STATIC` | Hosting environments, such as data warehouses where the same predicate must be applied to multiple database objects. | Yes |
| `CONTEXT_SENSITIVE` | ■ At statement parse time<br>■ At statement execution time when the local application context has changed since the last use of the cursor | 3-tier, session pooling applications where policies enforce two or more predicates for different users or groups. | No |
| `SHARED_CONTEXT_SENSITIVE` | First time the object is reference in a database session. Predicates are cached in the session's private memory UGA so policy functions can be shared among objects. | Same as `CONTEXT_SENSITIVE`, but multiple objects can share the policy function from the session UGA. | Yes |
| `DYNAMIC` | Policy function re-executes every time a policy-protected database object is accessed. | Applications where policy predicates must be generated for each query, such as time-dependent policies where users are denied access to database objects at certain times during the day | No |

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

Static and context sensitive policies enable you to optimize server performance because they do not execute the policy function each time protected database objects are accessed. However, Oracle recommends that before you enable policies as either static or context sensitive, you first test them as `DYNAMIC` policy types, which execute every time. Testing policy functions as `DYNAMIC` policies first enables you to observe how the policy function affects each query because nothing is cached. This ensures that the functions work properly before you enable them as static or context sensitive policy types to optimize performance.

Dynamic policies are the system default. If you do not specify a policy type with the DBMS_RLS.ADD_POLICY procedure, then by default your policy will be dynamic. You can specifically configure a policy to be dynamic by setting the policy_type parameter of the DBMS_RLS.ADD_POLICY procedure to DYNAMIC. Refer to Example 14–1 for the syntax.

***Example 14–1 Syntax for Enabling Policy Types with DBMS_RLS.ADD_POLICY***

```
DBMS_RLS.ADD_POLICY (
    .
    .
    .
    policy_type => dbms_rls.POLICY_TYPE);
```

---

**Note:** The DBMS_RLS.ADD POLICY policy_type parameter is intended to replace the static_policy parameter, which may be desupported in future releases.

---

**See Also:** The following topics for a more detailed discussion of static and context sensitive policies:

- "About Static Policies" on page 14-39
- "About Context Sensitive Policies" on page 14-39

## Optimizing Performance by Enabling Static and Context Sensitive Policies

In previous releases, policies were dynamic, which means the database executes the policy function for each query or DML statement. In addition to dynamic policies, the current release of the Oracle database provides static and context sensitive policies. These policy types provide a means to improve server performance because they do not always reexecute policy functions for each DML statement and can be shared across multiple database objects.

---

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

---

### About Static Policies

Static policy predicates are cached in SGA, so policy functions do not reexecute for each query, resulting in faster performance. When you specify a static policy, the same predicate is always enforced for all users in the instance. However, each execution of the same cursor could produce a different row set even for the same predicate because the predicate may filter the data differently based on attributes such as `SYS_CONTEXT` or `SYSDATE`.

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

```
where dept=SYS_CONTEXT ('HR_APP','deptno')
```

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

You can enable static policies by setting the `policy_type` parameter of the `DBMS_RLS.ADD_POLICY` procedure to either `STATIC` or `SHARED_STATIC`, depending on whether you want the policy to be shared across multiple objects. (See Example 14–1 on page 14-38 for the syntax.)

**When to Use Static Policies** Static policies are ideal for environments where every query requires the same predicate and fast performance is essential, such as hosting environments. For these situations when the policy function appends the same predicate to every query, reexecuting the policy function each time adds unnecessary overhead to the system. For example, consider a data warehouse that contains market research data for customer organizations who are competitors to one another. The warehouse must enforce the policy that each organization can see only their own market research, which is expressed by the predicate `where subscriber_id=SYS_CONTEXT('customer', 'cust_num')`. Using `SYS_CONTEXT` for the application context enables the database to dynamically change which organization's rows are returned. There is no need to reexecute the function, so the predicate can be cached in the SGA, thus conserving system resources and improving performance.

### About Context Sensitive Policies

In contrast to static policies, context sensitive policies do not always cache the predicate. With context sensitive policies, the server assumes that the predicate will change after statement parse time. But if there is no change in local application context, the server does not reexecute the policy function within the user session. If there has been a change in context, then the server reexecutes the policy function to ensure it captures any changes to the predicate since the initial parsing. These policies are useful where different predicates should apply depending on which user is executing the query. For example, consider the case where managers should always have the predicate `where group=managers` and employees should always have the predicate `where empno=emp_id`.

Shared context sensitive policies operate in the same way as regular context sensitive policies, except they can be shared across multiple database objects. For this policy type, all objects can share the policy function from the UGA, where the predicate is cached until the local session context changes.

You can enable context sensitive policies by setting the `policy_type` parameter of the `DBMS_RLS.ADD_POLICY` procedure to either `CONTEXT_SENSITIVE` or `SHARED_CONTEXT_SENSITIVE`. (See Example 14–1 on page 14-38 for the syntax.)

**When to Use Context Sensitive Policies** This type of policy is useful when a predicate need not change for a user's session, but the policy must enforce two or more different predicates for different users or groups. For example, consider a `SALES_HISTORY` table with a single policy of "analysts see only their own products" and "regional employees see only their own region." In this case, the server must reexecute the policy function each time the type of user changes. The performance gain is realized when a user can log in and issue several DML statements against the protected object without causing the server to reexecute the policy function.

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

## Adding Policies for Column-Level VPD

Column-level VPD, which can be applied to a table or a view, enables you to enforce security when a security-relevant column is referenced in a query, resulting in row-level security. Column-level VPD cannot be applied to a synonym.

It can be configured to produce two distinct behaviors as follows:

- Default Behavior

  Restricts the number of rows returned by queries that reference columns containing sensitive information. Set this behavior by specifying the security-relevant column names with the `sec_relevant_cols` parameter of the `DBMS_RLS.ADD_POLICY` procedure.

- Column Masking Behavior

  Returns all rows for queries, but it returns `NULL` values for the columns that contain sensitive information. Set this behavior by setting the `sec_relevant_cols_opt` parameter of the `DBMS_RLS.ADD_POLICY` procedure to `dbms_rls.ALL_ROWS`.

The following example shows a VPD policy in which sales department users should not see the salaries of people outside their own department (department number 30). The relevant columns for such a policy are `SAL` and `COMM`. First, the VPD policy function is created and then added by using the `DBMS_RLS` PL/SQL package as shown in Example 14–2:

***Example 14–2 Creating and Adding a Column-Level VPD Policy***

```
*/Create a policy function which does not expose salaries of employees outside
the sales department (department 30)/*

CREATE OR REPLACE FUNCTION pf1 (oowner IN VARCHAR2, ojname IN VARCHAR2)
RETURN VARCHAR2 AS
con VARCHAR2 (200);
BEGIN
   con := 'deptno=30';
   RETURN (con);
END pf1;
```

Then the policy is added with the `DBMS_RLS` package as follows:

```
BEGIN
   DBMS_RLS.ADD_POLICY (object_schema=>'scott', object_name=>'emp',
                        policy_name=>'sp', function_schema=>'pol_admin',
                        policy_function=>'pf1',
                        sec_relevant_cols=>'sal,comm');
END;
```

The two different behaviors of column-level VPD are discussed in the following sections using Example 14–2 on page 14-41 as a starting point for discussion.

## Default Behavior

The default behavior for column-level VPD is to restrict the number of rows returned for a query that references columns containing sensitive information.

These security-relevant columns are specified with the `sec_relevant_cols` parameter of the `DBMS_RLS.ADD_POLICY` procedure.

For an example of column-level VPD default behavior, consider sales department users with `SELECT` privilege on the `emp` table, which is protected with the column-level VPD policy created in Example 14–2. When these users perform the following query:

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

the database returns a subset of rows as follows:

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

Only the rows display in which the user should have access to all columns.

### Column Masking Behavior

In contrast to the default behavior of column-level VPD, the column masking behavior displays all rows, but returns sensitive column values as `NULL`. To set this behavior set the `sec_relevant_cols_opt` parameter of the `DBMS_RLS.ADD_POLICY` procedure to `dbms_rls.ALL_ROWS` in addition to setting the default behavior parameter.

For an example of column-level VPD column masking behavior, consider that the same VPD policy (created Example 14–2 on page 14-41) applies, but it has been added with the `sec_relevant_cols_opt` parameter specified also. See Example 14–3 on page 14-42.

***Example 14–3 Adding a Column-level VPD Policy with Column Masking Behavior***

```
*/add the ALL_ROWS policy/*
BEGIN
     DBMS_RLS.ADD_POLICY(object_schema=>'scott', object_name=>'emp',
                         policy_name=>'sp', function_schema=>'pol_admin',
                         policy_function=>'pf1',
                         sec_relevant_cols=>'sal,comm',
                         sec_relevant_cols_opt=>dbms_rls.ALL_ROWS);
```

```
END;
```

Now a sales department user with SELECT privilege on the emp table, performs the following query:

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

The database returns all rows specified in the query, but certain values are masked because of the VPD policy:

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

With column masking behavior, sales users see all rows returned by a query, but the SAL and COMM columns become NULL for rows containing information about employees outside the sales department.

Column masking behavior is subject to the following restrictions:

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

```
select * from employees
where salary = 10
```

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

**See Also:** The chapter on the `DBMS_RLS` package in the *PL/SQL Packages and Types Reference* for a discussion of the `DBMS_RLS.ADD_POLICY` procedure parameters and usage examples.

## Enforcing VPD Policies on Specific SQL Statement Types

VPD policies can be enforced for `SELECT`, `INSERT`, `UPDATE`, `INDEX`, and `DELETE` statements. Specify any combination of these statement types with the `DBMS_RLS.ADD_POLICY` procedure `statement_types` parameter as follows:

```
DBMS_RLS.ADD_POLICY (
         .
         .
         .
   statement_types=>'SELECT,INDEX');
```

### Enforcing Policies on Index Maintenance

A user who has privileges to maintain an index can see all the row data even if the user does not have full table access under a regular query, such as `SELECT`. For example, a user can create a function-based index which contains a user defined function with column values as its arguments. During index creation, the server passes column values of every row into the user function, making the row data available to the user who creates the index. Administrators can enforce VPD policies on index maintenance operations by specifying `INDEX` with the `statement_types` parameter as shown in the previous section.

## How to Check for Policies Applied to a SQL Statement

`V$VPD_POLICY` allows one to perform a dynamic view in order to check what policies are being applied to a SQL statement. When debugging, in your attempt to find which policy corresponds to a particular SQL statement, you should use the following table.

***Table 14–4 V$VPD_POLICY***

| Column Name | Type |
|---|---|
| ADDRESS | RAW(4\|8) |
| PARADDR | RAW(4\|8) |
| SQL_HASH | NUMBER |
| SQL_ID | VARCHAR2(13) |
| CHILD_NUMBER | NUMBER |
| OBJECT_OWNER | VARCHAR2(30) |
| OBJECT_NAME | VARCHAR2(30) |
| POLICY_GROUP | VARCHAR2(30) |
| POLICY | VARCHAR2(30) |
| POLICY_FUNCTION_OWNER | VARCHAR2(30) |
| PREDICATE | VARCHAR2(4000) |
| DBMS_RLS.REFRESH_GROUPED_POLICY | VARCHAR2(4096) |

**See Also:** *Oracle Database Reference* for more information about the V$VPD_POLICY view

## Users Who Are Exempt from VPD Policies

Two classes of users are exempt from VPD policies: the SYS user is exempt by default, and any other user can be exempt if granted the EXEMPT ACCESS POLICY system privilege. These two cases are discussed in the following sections.

### SYS User Exempted from VPD Policies

The database user SYS is always exempt from VPD or Oracle Label Security policy enforcement, regardless of the export mode, application, or utility that is used to extract data from the database. However, SYSDBA actions can be audited.

## EXEMPT ACCESS POLICY System Privilege

The system privilege EXEMPT ACCESS POLICY allows a user to be exempted from all fine-grained access control policies on any SELECT or DML operation (INSERT, UPDATE, and DELETE). This provides ease of use for such administrative activities as installation, and import and export of the database through a non-SYS schema.

Also, regardless of the utility or application that is being used, if a user is granted the EXEMPT ACCESS POLICY privilege, then the user is exempt from VPD and Oracle Label Security policy enforcement. That is, the user will not have any VPD or Oracle Label Security policies applied to their data access.

Since EXEMPT ACCESS POLICY negates the effect of fine-grained access control, this privilege should only be granted to users who have legitimate reasons for bypassing fine-grained access control enforcement. This privilege should not be granted WITH ADMIN OPTION, so that users cannot pass on the EXEMPT ACCESS POLICY privilege to other users, and thus propagate the ability to bypass fine-grained access control.

## Automatic Reparse

**Note:** This feature is applicable when COMPATIBLE is set to 9.0.1.

Starting from Oracle9*i*, queries against objects enabled with fine-grained access control always execute the policy function to make sure the most current predicate is used for each policy. For example, in the case of a time-based policy function, in which queries are only allowed between 8:00 a.m. and 5:00 p.m., a cursor execution parsed at noon cause the policy function to execute, ensuring the policy is consulted again for the query.

Automatic reparse does not occur under the following conditions:

- If you specified STATIC_POLICY=TRUE when adding the policy to indicate that the policy function always returns the same predicate.
- If you set the _dynamic_rls_policies parameter to FALSE in the initialization parameters files. Typically, this parameter is set to FALSE for users whose security policies do not return different predicates within a database session to reduce the execution overhead.

For deployment environments where the latest application context value is always the desired value, the _app_ctx_vers parameter can be set to FALSE in the initialization parameters file to reduce the overhead of application context scoping. By default, it is set to `TRUE` and changes of value within a SQL statement are not visible. This default may change in the future, thus developers should be careful not to allow changes of application context values within a SQL statement using a user defined function. In general, you should not depend on the order of SQL statement execution, which can yield inconsistent results depending on query plans.

> **See Also:** "Using Dynamic SQL with SYS_CONTEXT" on page 14-5

## VPD Policies and Flashback Query

By default, operations on the database use the most recent committed data available. The flashback query feature enables you to query the database as it was at some time in the past. To write an application that uses flashback query, you can use the `AS OF` clause in SQL queries to specify either a time or a system change number (SCN) and then query against the committed data from the specified time. You can also use the `DBMS_FLASHBACK` PL/SQL package, which requires more code, but enables you to perform multiple operations, all of which refer to the same past time.

Flashback queries return data as it stood at the time specified in the query. However, if you use flashback query against a database object that is protected with VPD policies, then the current policies are applied to the old data. Applying the current VPD policies to flashback query data is more secure because it reflects the most current business policy.

> **See Also:**
>
> - *Oracle Database Application Developer's Guide - Fundamentals* for more information about the flashback query feature and how to write applications that use it.
> - *PL/SQL Packages and Types Reference* for more information about the `DBMS_FLASHBACK` PL/SQL package

15

# Preserving User Identity in Multitiered Environments

Enforcing security in multitiered environments can be challenging. This chapter discusses the risks associated with computing environments that span multiple tiers, and explains how to implement proxy authentication and use client identifiers for preserving user identity.

This chapter contains the following topics:

| Topic Category | Links to Topics |
|---|---|
| Security Challenges of Three-tier Computing | ■ Who Is the Real User?<br>■ Does the Middle Tier Have Too Much Privilege?<br>■ How to Audit? Whom to Audit?<br>■ What Are the Authentication Requirements for Three-tier Systems? |
| Oracle Database Solutions for Preserving User Identity | ■ Proxy Authentication<br>■ Client Identifiers |

## Security Challenges of Three-tier Computing

While three-tier computing provides many benefits, it raises a number of security issues. These issues are described in the following sections:

- Who Is the Real User?
- Does the Middle Tier Have Too Much Privilege?
- How to Audit? Whom to Audit?
- What Are the Authentication Requirements for Three-tier Systems?

### Who Is the Real User?

Most organizations want to know the identity of the actual user who is accessing the database, for reasons of access control, resource monitoring, or auditing. User accountability is diminished if the identity of the users cannot be traced through all tiers of the application.

Furthermore, if only the application server knows who the user is, then all security enforcement for each user must be done by the application itself. Application-based security is very expensive. If each application that accesses the data must enforce security, then security must be re-implemented in each and every application. It is often preferable to build security on the data itself, accountability for each user enforced within the database.

### Does the Middle Tier Have Too Much Privilege?

Some organizations are willing to accept three-tier systems within the enterprise, in which "all-privileged" middle tiers, such as transaction processing (TP) monitors, can perform all actions for all users. In this architecture, the middle tier connects to the database as the same user for all application users. It therefore needs to have *all* privileges that application users need to do their jobs.

This computing model can be undesirable in the Internet, where the middle tier resides outside, on, or just inside a firewall. More desirable, in this context, is a *limited trust model*, in which the identity of the real client is known to the data server, and the application server (or other middle tier) has a restricted privilege set.

Also useful is the ability to limit the users on whose behalf a middle tier can connect, and the roles the middle tier can assume for the user. For example, many organizations would prefer that users have different privileges depending on where they are connecting from. A user connecting to a Web server or application server on the firewall might only be able to use very minimal privileges to access data, whereas a user connecting to a Web server or application server within the enterprise might be able to exercise all privileges she is otherwise entitled to have.

## How to Audit? Whom to Audit?

Accountability through auditing is a basic principle of information security. Most organizations want to know on whose behalf a transaction was accomplished, not just that a particular application server performed a transaction. A system must therefore be able to differentiate between a user performing a transaction, and an application server performing a transaction on behalf of a user.

Auditing in three-tier systems should be tied to the issue of knowing the real user: if you cannot preserve the user's identity through the middle tier of a three-tier application, then you cannot audit actions on behalf of the user.

## What Are the Authentication Requirements for Three-tier Systems?

In client/server systems, authentication tends to be straightforward: the client authenticates to the server. In three-tier systems authentication is more difficult, because there are several potential types of authentication.

- Client to Middle Tier Authentication
- Middle Tier to Database Authentication
- Client Re-Authentication Through Middle Tier to Database

### Client to Middle Tier Authentication

Client authentication to the middle tier is clearly required if a system is to conform with basic security principles. The middle tier is typically the first gateway to useful information that the user can access. Users *must*, therefore, authenticate to the middle tier. Note that such authentication can be mutual; that is, the middle tier authenticates to the client just as the client authenticates to the middle tier.

### Middle Tier to Database Authentication

Since the middle tier must typically initiate a connection to a database to retrieve data (whether on its own behalf or on behalf of the user), this session clearly must be authenticated. In fact, the Oracle Database does not allow unauthenticated sessions. Again, middle tier to database authentication can also be mutual if using a protocol that supports this, such as SSL.

### Client Re-Authentication Through Middle Tier to Database

Client re-authentication from the middle tier to the database is problematic in three-tier systems. The username might not be the same on the middle tier and the database. In this case, users may need to reenter a username and credential, which the middle tier uses to connect on their behalf. Or, more commonly, the middle tier may need to map the username provided, to a database username. This mapping is often done in an LDAP-compliant directory service, such as Oracle Internet Directory.

For the client to re-authenticate himself to the database, the middle tier either needs to ask the user for a credential (which it then must be trusted to pass to the database), or the middle tier must retrieve a credential for the user and use that to authenticate the user. Both approaches involve security risks, because the middle tier is provided with the user's credentials.

Re-authenticating the client to the back-end database is not always beneficial. First, two sets of authentication handshakes for each user involves considerable network overhead. Second, you must trust the middle tier to have authenticated the user. (You clearly must trust the middle tier if it is privy to the user's credential.) It is therefore not unreasonable for the database to simply accept that the middle tier has performed proper authentication. In other words, the database accepts the identity of the real client without requiring the real client to authenticate herself.

For some authentication protocols, client re-authentication is just not possible. For example, many browsers and application servers support the Secure Sockets Layer (SSL) protocol. Both the Oracle Database (through Oracle Advanced Security) and Oracle Application Server support the use of SSL for client authentication. However, SSL is a point-to-point protocol, not an end-to-end protocol. It cannot be used to re-authenticate a browser client (through the middle tier) to the database.

In short, organizations deploying three-tier systems require flexibility as regards re-authentication of the client.

## Oracle Database Solutions for Preserving User Identity

Many organizations want to know who the user is through all tiers of an application, without sacrificing the benefits of a middle tier. The Oracle Database supports the following ways of preserving user identity through the middle tier of an application:

- Proxy Authentication

  Oracle Database provides proxy authentication in OCI or thick JDBC for database users or enterprise users. Enterprise users are those who are managed in Oracle Internet Directory and who access a shared schema in the database.

- Client Identifiers

  Oracle Database provides the `CLIENT_IDENTIFIER` attribute of the built-in `USERENV` application context namespace for application users. These users are known to an application but unknown to the database. The `CLIENT_IDENTIFIER` attribute can be used to capture any value the application uses for identification or access control and pass it to the database. `CLIENT_IDENTIFIER` is supported in OCI, thick JDBC, and thin JDBC.

### Proxy Authentication

The following sections explain how proxy authentication works and how to use it:

- Passing Through the Identity of the Real User by Using Proxy Authentication
- Limiting the Privilege of the Middle Tier
- Re-authenticating The User through the Middle Tier to the Database
- Auditing Actions Taken on Behalf of the Real User
- Advantages of Proxy Authentication

#### Passing Through the Identity of the Real User by Using Proxy Authentication

For enterprise users or database users, OCI or thick JDBC enables a middle tier to set up, within a single database connection, a number of user sessions, each of which uniquely identifies a connected user. This is commonly referred to as *connection pooling*. These sessions reduce the network overhead of creating separate network connections from the middle tier to the database.

**Authentication Process from Clients through Middle Tiers to the Database** The full authentication sequence from the client to the middle tier to the database occurs as follows:

1. The client authenticates to the middle tier, using whatever form of authentication the middle tier will accept. For example, the client could authenticate to the middle tier using a username/password, or an X.509 certificate by means of SSL.
2. The middle tier authenticates itself to the database, using whatever form of authentication the database accepts. This could be a password, or an authentication mechanism supported by Oracle Advanced Security, such as a **Kerberos ticket** or an X.509 certificate (SSL).
3. The middle tier then creates one or more sessions for users using OCI or thick JDBC.
   - If the user is a database user, the session must, as a minimum, include the database username. If the database requires it, the session may also include a password (which the database verifies against the password store in the database). The session may also include a list of database roles for the user.
   - If the user is an enterprise user, the session may provide different information depending on how the user is authenticated. For example:
     - If the user authenticated to the middle tier by way of SSL, then the middle tier can provide the DN from the user's X.509 certificate, or the certificate itself in the session. The database uses the DN to look up the user in Oracle Internet Directory.
     - If the user is a password-authenticated enterprise user, then the middle tier must provide, as a minimum, a globally unique name for the user. The database uses this name to look up the user in Oracle Internet Directory. If the session also provides a password for the user, the database will verify the password against Oracle Internet Directory. The user's roles are automatically retrieved from Oracle Internet Directory after the session is established.
   - The middle tier may optionally provide a list of database roles for the client. These roles are enabled if the proxy is authorized to exercise the roles on the client's behalf.
4. The database verifies that the middle tier has the privilege to create sessions on behalf of the user.

The `OCISessionBegin` call will fail if the application server is not allowed to proxy on behalf of the client by the administrator, or if the application server is not allowed to activate the specified roles.

### Limiting the Privilege of the Middle Tier

"Least privilege" is the principle that users should have the fewest privileges necessary to perform their duties, and no more. As applied to middle tier applications, this means that the middle tier should not have more privileges than it needs. The Oracle Database enables you to limit the middle tier such that it can connect only on behalf of certain database users, using only specific database roles. You can limit the *privilege* of the middle tier to connect on behalf of an enterprise user, stored in an LDAP directory, by granting to the middle tier the privilege to connect as the mapped database user. For instance, if the enterprise user is mapped to the APPUSER schema, you must at least grant to the middle tier the ability to connect on behalf of APPUSER. Otherwise, attempts to create a session for the enterprise user will fail.

However, you cannot limit the *ability* of the middle tier to connect on behalf of enterprise users. For example, suppose that user Sarah wants to connect to the database through a middle tier, `appsrv` (which is also a database user). Sarah has multiple roles, but it is desirable to restrict the middle tier to exercise only the `clerk` role on her behalf.

A DBA could effectively grant permission for `appsrv` to initiate connections on behalf of Sarah using her `clerk` role only, using the following syntax:

```
ALTER USER sarah GRANT CONNECT THROUGH appsrv WITH ROLE clerk;
```

By default, the middle tier cannot create connections for any client. The permission must be granted for each user.

To allow `appsrv` to use all of the roles granted to the client Sarah, the following statement would be used:

```
ALTER USER sarah GRANT CONNECT THROUGH appsrv;
```

Each time a middle tier initiates an OCI or thick JDBC session for another database user, the database verifies that the middle tier is authorized to connect for that user, using the role specified.

**Note:** Instead of using default roles, create your own and assign only necessary privileges to them. For example, if users only need CREATE SESSION, then granting them CONNECT, which includes several privileges (see Table 10–1, " Predefined Roles"), may result in assigning too many privileges for your database application. Creating your own roles enables you to control the privileges granted by them, and protects you if Oracle changes or removes default roles.

## Re-authenticating The User through the Middle Tier to the Database

Administrators can specify that authentication is required by using the AUTHENTICATION REQUIRED proxy clause with the ALTER USER SQL statement. In this case, the middle tier must provide user authentication credentials.

For example, suppose that user Sarah wants to connect to the database through a middle tier, appsrv. A DBA could require that appsrv provides authentication credentials for Sarah by using the following syntax:

```
ALTER USER sarah GRANT CONNECT THROUGH appsrv AUTHENTICATION REQUIRED;
```

The AUTHENTICATION REQUIRED clause ensures that authentication credentials for the user must be presented when the user is authenticated through the specified proxy.

**Note:** For backward compatibility, if a DBA uses the AUTHENTICATED USING PASSWORD proxy clause, then the system transforms it to AUTHENTICATION REQUIRED.

**Using Password-Based Proxy Authentication** Using password-based proxy authentication, the password of the client is passed to the middle-tier server. The middle-tier server then passes the password as an attribute to the data server for verification. The main advantage to this is that the client machine does not have to have Oracle software actually installed on it to perform database operations.

To pass over the password of the client, the middle-tier server calls OCIAttrSet() with the following pseudo-interface:

```
OCIAttrSet (OCISession *session_handle,
OCI_HTYPE_SESSION,
lxstp *password,
(ub4) 0,
OCI_ATTR_PASSWORD,
OCIError *error_handle);
```

**Using Proxy Authentication with Enterprise Users** If the middle tier is connecting to the database as a client who is an enterprise user, either the distinguished name, or the X.509 certificate containing the distinguished name is passed over instead of the database user name. If the user is a password-authenticated enterprise user, then the middle tier must provide, as a minimum, a globally unique name for the user. The database uses this name to look up the user in Oracle Internet Directory.

To pass over the distinguished name of the client, the application server would call `OCIAttrSet()` with the following pseudo-interface.

```
OCIAttrSet(OCISession *session_handle,
OCI_HTYPE_SESSION,
lxstp *distinguished_name,
(ub4) 0,
OCI_ATTR_DISTINGUISHED_NAME,
OCIError *error_handle);
```

To pass over the entire certificate, the middle tier would use the following pseudo-interface:

```
OCIAttrSet(OCISession *session_handle,
OCI_HTYPE_SESSION,
ub1 *certificate,
ub4 certificate_length,
OCI_ATTR_CERTIFICATE,
OCIError *error_handle);
```

If the type is not specified, then the server will use its default certificate type of X.509.

**Note:** `OCI_ATTR_CERTIFICATE` is `DER` encoded.

If using proxy authentication for password-authenticated enterprise users, then use the same OCI attributes as for database users authenticated by password (`OCI_ATTR_USERNAME`). The database first checks the username against the database; if no user is found, then the database checks the username in the directory. This username must be globally unique.

## Auditing Actions Taken on Behalf of the Real User

The proxy authentication features of the Oracle Database enable you to audit actions that a middle tier performs on behalf of a user. For example, suppose an application server `hrappserver` creates multiple sessions for users Ajit and Jane. A DBA could enable auditing for `SELECT`s on the `bonus` table that `hrappserver` initiates for Jane as follows:

```
AUDIT SELECT TABLE BY hrappserver ON BEHALF OF Jane;
```

Alternatively, the DBA could enable auditing on behalf of multiple users (in this case, both Jane and Ajit) connecting through a middle tier as follows:

```
AUDIT SELECT TABLE BY hrappserver ON BEHALF OF ANY;
```

This auditing option only audits `SELECT` statements being initiated by `hrappserver` on behalf of other users. A DBA can enable separate auditing options to capture `SELECT`s against the `bonus` table from clients connecting directly to the database:

```
AUDIT SELECT TABLE;
```

For audit actions taken on behalf of the real user, you cannot audit `CONNECT ON BEHALF OF` *`DN`*, since the user in the LDAP directory is not known to the database. However, if the user accesses a shared schema (for example, `APPUSER`), then you can audit `CONNECT ON BEHALF OF APPUSER`.

**See Also:** ▪ Chapter 11, "Configuring and Administering Auditing"

## Advantages of Proxy Authentication

In multitier environments, proxy authentication enables you to control the security of middle-tier applications by preserving client identities and privileges through all tiers, and by auditing actions taken on behalf of clients. For example, this feature allows the identity of a user using a Web application (which acts as a "proxy") to be passed through the application to the database server.

Three-tier systems provide many benefits to organizations.

- Application servers and Web servers enable users to access data stored in databases.
- Users like using a familiar, easy-to-use browser interface.
- Organizations can separate application logic from data storage, partitioning the former in application servers and the latter in databases.

- Organizations can also lower their cost of computing by replacing many "fat clients" with a number of "thin clients" and an application server.

In addition, Oracle proxy authentication delivers the following security benefits:

- A limited trust model, by controlling the users on whose behalf middle tiers can connect, and the roles the middle tiers can assume for the user
- Scalability, by supporting user sessions through OCI and thick JDBC, and eliminating the overhead of re-authenticating clients
- Accountability, by preserving the identity of the real user through to the database, and enabling auditing of actions taken on behalf of the real user
- Flexibility, by supporting environments in which users are known to the database, and in which users are merely "application users" of which the database has no awareness

**Note:** Oracle Database supports this proxy authentication functionality in three tiers only; it does not support it across multiple middle tiers.

## Client Identifiers

The following sections explain how using client identifiers works and how to use them:

- Support for Application User Models by Using Client Identifiers
- Using the CLIENT_IDENTIFIER Attribute to Preserve User Identity
- Using CLIENT_IDENTIFIER Independent of Global Application Context

### Support for Application User Models by Using Client Identifiers

Many applications use session pooling to set up a number of sessions to be reused by multiple application users. Users authenticate themselves to a middle-tier application, which uses a single identity to log in to the database and maintains all the user connections. In this model, "application users" are users who are authenticated to the middle tier of an application, but who are not known to the database. Oracle Database supports use of a `CLIENT_IDENTIFIER` attribute that acts like an application user proxy for these types of applications.

In this model, the middle tier passes a client identifier to the database upon the session establishment. (The client identifier could actually be anything that represents a client connecting to the middle tier, for example, a cookie or an IP address.) The client identifier, representing the application user, is available in user session information and can also be accessed with an application context (by using the USERENV naming context). In this way, applications can set up and reuse sessions, while still being able to keep track of the "application user" in the session. Applications can reset the client identifier and thus reuse the session for a different user, enabling high performance.

### Using the CLIENT_IDENTIFIER Attribute to Preserve User Identity

The CLIENT_IDENTIFIER, a predefined attribute of the built-in application context namespace, USERENV, can be used to capture the "application username" for use with global application context, or it can be used independently. When used independent of global application context, CLIENT_IDENTIFIER can be set with the DBMS_SESSION interface. The ability to pass a CLIENT_IDENTIFIER to the database is supported in OCI and thick JDBC.

When CLIENT_IDENTIFIER is used with global application context, it provides flexibility and high performance for building applications. For example, suppose a Web-based application that provides information to business partners has three types of users: gold partner, silver partner, and bronze partner, representing different levels of information available. Instead of each user having his own session set up with individual application contexts, the application could set up global applications contexts for gold partners, silver partners, and bronze partners. Then, use the CLIENT_IDENTIFIER to point the session at the correct context, in order to retrieve the appropriate type of data. The application need only initialize the three global contexts once, and use the CLIENT_IDENTIFIER to access the correct application context to limit data access. This provides performance benefits through session reuse, and through accessing global application contexts set up once, instead of having to initialize application contexts for each session individually.

> **See Also:** "How to Use Global Application Context" on page 14-24 for a discussion about implementing global application contexts and an example of using the CLIENT_IDENTIFIER attribute with it.

### Using CLIENT_IDENTIFIER Independent of Global Application Context

Using the CLIENT_IDENTIFIER attribute is especially useful for applications whose users are unknown to the database. In these situations, the application typically connects as a single database user, and all actions are taken as that user. Since all user sessions are created as the same user, this security model makes it very difficult to achieve data separation for each user. These applications can use the CLIENT_IDENTIFIER attribute to preserve the "real" application user's identity to the database.

With this approach, sessions can be reused by multiple users by changing the value of the CLIENT_IDENTIFIER attribute, which is used to capture the name of the real application user. This avoids the overhead of setting up a separate session and separate attributes for each user, and enables reuse of sessions by the application. When the CLIENT_IDENTIFIER attribute value changes, the change is piggybacked on the next OCI or thick JDBC call, for additional performance benefits.

For example, a user, Daniel, connects to a Web Expense application. Daniel is not a database user, he is a typical Web Expense application user. The application accesses the built-in application context namespace and sets DANIEL as the CLIENT_IDENTIFIER attribute value. Daniel completes his Web Expense form and exits the application. Then Ajit connects to the Web Expense application. Instead of setting up a new session for Ajit, the application reuses the session that currently exists for Daniel, by changing the CLIENT_IDENTIFIER to AJIT. This avoids the overhead of setting up a new connection to the database and the overhead of setting up a global application context. The CLIENT_IDENTIFIER attribute can be set to any value on which the application bases access control. It does not have to be the application username.

To use the DBMS_SESSION package to set and clear the CLIENT_IDENTIFIER on the middle tier, use the following interfaces:

- SET_IDENTIFIER
- CLEAR_IDENTIFIER

The middle tier uses SET_IDENTIFIER to associate the database session with a particular user or group. Then, the CLIENT_IDENTIFIER is an attribute of the session and can be viewed in session information.

To set the CLIENT_IDENTIFIER attribute with OCI, use the OCI_ATTR_CLIENT_IDENTIFIER attribute in the call to OCIAttrSet(). Then, on the next request to the server, the information is propagated and stored in the server sessions. For example:

```
OCIAttrSet (session,
           OCI_HTYPE_SESSION,
           (dvoid *) "appuser1",
           (ub4)strlen("appuser1"),
           OCI_ATTR_CLIENT_IDENTIFIER,
           OCIError *error_handle);
```

For applications that use JDBC, in a connection pooling environment, the client identifier can be used to identify which light-weight user is currently using the database session. To set the `CLIENT_IDENTIFIER` for JDBC applications, use the following `oracle.jdbc.OracleConnection` interface methods:

- `setClientIdentifier()` : Sets the client identifier for a connection
- `clearClientIdentifier()` : Clears the client identifier for a connection

**See Also:**

- The chapter on the `DBMS_SESSION` interface in the *PL/SQL Packages and Types Reference*
- The section on OCI_ATTR_CLIENT_IDENTIFIER user session handle attribute in the *Oracle Call Interface Programmer's Guide*
- The section on the `oracle.jdbc.OracleConnection` interface in the *Oracle Database JDBC Developer's Guide and Reference* for information about the `setClientIdentifer` and the `clearClientIdentifier` methods

# 16
# Developing Applications Using Data Encryption

In addition to controlling access, you can also encrypt data to reduce your security risks. However, data encryption is not an infallible solution. This chapter discusses the appropriate uses of data encryption and provides examples of using data encryption in applications. It contains the following topics:

## Securing Sensitive Information

While the Internet poses new challenges in information security, many of them can be addressed by the traditional arsenal of security mechanisms:

- Strong user authentication to identify users
- Granular access control to limit what users can see and do
- Auditing for accountability
- Network encryption to protect the confidentiality of sensitive data in transmission

Encryption is an important component of several of these solutions. For example, Secure Sockets Layer (SSL), an Internet-standard network encryption and authentication protocol, uses encryption to strongly authenticate users by means of X.509 digital certificates. SSL also uses encryption to ensure data confidentiality, and cryptographic checksums to ensure data integrity. Many of these uses of encryption are relatively transparent to a user or application. For example, many browsers support SSL, and users generally do not need to do anything special to enable SSL encryption.

Oracle has provided network encryption between database clients and the Oracle database since version 7. Oracle Advanced Security, an option to the Oracle database server, provides encryption and cryptographic checksums for integrity checking with any protocol supported by the database, including Oracle Net, Java Database Connectivity (JDBC—both "thick" and "thin" JDBC), and the Internet Intra-Orb Protocol (IIOP). Oracle Advanced Security also supports SSL for Oracle Net, "thick" JDBC, and IIOP connections.

While encryption is not a security cure-all, it is an important tool in addressing specific security threats. In particular, the rapid growth of e-business has spurred increased encryption of stored data, such as credit card numbers. While SSL is typically used to protect these numbers in transit to a Web site, where data is not protected as it is in storage, the file system or database storing them often does so as clear text (un-encrypted). Information stored in the clear is then directly accessible to anyone who can break into the host and gain root access, or gain illicit access to the database. Databases can be made quite secure through proper configuration, but they can also be vulnerable to host break-ins if the host is misconfigured. In well-publicized break-ins, a hacker obtained a large list of credit card numbers by breaking into a database. Had they been encrypted, the stolen information would have been useless. Encryption of stored data can thus be an important tool in limiting information loss even in the normally rare occurrence that access controls are bypassed.

## Principles of Data Encryption

While there are many good reasons to encrypt data, there are many bad reasons to encrypt data. Encryption does not solve all security problems, and may even make some problems worse. The following sections describe some misconceptions about encryption of stored data:

- Principle 1: Encryption Does Not Solve Access Control Problems
- Principle 2: Encryption Does Not Protect Against a Malicious DBA
- Principle 3: Encrypting Everything Does Not Make Data Secure

### Principle 1: Encryption Does Not Solve Access Control Problems

Most organizations need to limit data access to those who have a "need to know." For example, a human resources system may limit employees to viewing only their own employment records, while allowing managers of employees to see the employment records of subordinates. Human resources specialists may also need to see employee records for multiple employees.

This type of security policy—limiting data access to those with a need to see it—is typically addressed by access control mechanisms. The Oracle database has provided strong, independently-evaluated access control mechanisms for many years. It enables access control enforcement to an extremely fine level of granularity, through its Virtual Private Database capability.

Because human resources records are considered sensitive information, it is tempting to think that all information should be encrypted "for better security." However, encryption cannot enforce granular access control, and it may actually hinder data access. For example, an employee, his manager, and a human resources clerk may all need to access the employee's record. If all employee data is encrypted, then all three must be able to access the data in un-encrypted form. Therefore, the employee, the manager and the HR clerk would have to share the same encryption key to decrypt the data. Encryption would therefore not provide any additional security in the sense of better access control, and the encryption might actually hinder the proper or efficient functioning of the application. An additional issue is that it is very difficult to securely transmit and share encryption keys among multiple users of a system.

A basic principle behind encrypting stored data is that it must not interfere with access control. For example, a user who has `SELECT` privilege on `EMP` should not be limited by the encryption mechanism from seeing all the data he is otherwise allowed to see. Similarly, there is little benefit to encrypting part of a table with one key and part of a table with another key if users need to see all encrypted data in the table; it merely adds to the overhead of decrypting the data before users can read it. If access controls are implemented well, encryption adds little additional security within the database itself. Any user who has privilege to access data within the database has no more nor any less privilege as a result of encryption. Therefore, encryption should never be used to solve access control problems.

## Principle 2: Encryption Does Not Protect Against a Malicious DBA

Some organizations, concerned that a malicious user might gain elevated (DBA) privilege by guessing a password, like the idea of encrypting stored data to protect against this threat. However, the correct solution to this problem is to protect the DBA account, and to change default passwords for other privileged accounts. The easiest way to break into a database is by using a default password for a privileged account that an administrator has allowed to remain unchanged. One example is `SYS/CHANGE_ON_INSTALL`.

While there are many destructive things a malicious user can do to a database after gaining DBA privilege, encryption will not protect against many of them. Examples include corrupting or deleting data, exporting user data to the file system to mail the data back to himself so as to run a password cracker on it, and so on.

Some organizations are concerned that DBAs, typically having all privileges, are able to see all data in the database. These organizations feel that the DBAs should merely administer the database, but should not be able to see the data that the database contains. Some organizations are also concerned about concentrating so much privilege in one person, and would prefer to partition the DBA function, or enforce two-person access rules.

It is tempting to think that encrypting all data (or significant amounts of data) will solve these problems, but there are better ways to protect against these threats. For example, Oracle does support limited partitioning of DBA privileges. Oracle provides native support for SYSDBA and SYSOPER users. SYSDBA has all privileges, but SYSOPER has a limited privilege set (such as startup and shutdown of the database).

Furthermore, an organization can create smaller roles encompassing a number of system privileges. A `JR_DBA` role might not include all system privileges, but only those appropriate to a junior DBA (such as `CREATE TABLE`, `CREATE USER`, and so on).

Oracle also enables auditing the actions taken by SYS (or SYS-privileged users) and storing that audit trail in a secure operating system location. Using this model, a separate auditor who has root privileges on the operating system can audit all actions by SYS, enabling the auditor to hold all DBAs accountable for their actions.

> **See Also:** "Auditing Administrative Users" on page 11-4 for information about using the `AUDIT_SYS_OPERATIONS` parameter.

The DBA function by its nature is a trusted position. Even organizations with the most sensitive data—such as intelligence agencies—do not typically partition the DBA function. Instead, they vet their DBAs strongly, because it is a position of trust. Periodic auditing can help to uncover inappropriate activities.

Encryption of stored data must not interfere with the administration of the database; otherwise, larger security issues can result. For example, if by encrypting data you corrupt the data, you've created a security problem: the data itself has become uninterpretable, and it may not be recoverable.

Encryption can be used to limit the ability of a DBA—or other privileged user—to see data in the database. However, it is not a substitute for vetting a DBA properly, or for controlling the use of powerful system privileges. If untrustworthy users have significant privileges, they can pose multiple threats to an organization, some of them far more significant than viewing un-encrypted credit card numbers.

## Principle 3: Encrypting Everything Does Not Make Data Secure

A common error is to think that if encrypting some data strengthens security, then encrypting everything makes all data secure.

As the discussion of the prior two principles illustrates, encryption does not address access control issues well, and it is important that encryption not interfere with normal access controls. Furthermore, encrypting an entire production database means that all data must be decrypted to be read, updated, or deleted. Encryption is inherently a performance-intensive operation; encrypting all data will significantly affect performance.

Availability is a key aspect of security. If encrypting data makes data unavailable, or adversely affects availability by reducing performance, then encrypting everything will have created a new security problem. Availability is also adversely affected by the database being inaccessible when encryption keys are changed, as good security practices require on a regular basis. When the keys are to be changed, the database is inaccessible while data is decrypted and re-encrypted with a new key or keys.

However, there may be advantages to encrypting data stored off-line. For example, an organization may store backups for a period of six months to a year off-line, in a remote location. Of course, the first line of protection is to secure the facility storing the data, by establishing physical access controls. Encrypting this data before it is stored may provide additional benefits. Since it is not being accessed on-line, performance need not be a consideration. While an Oracle database server does not provide this capability, there are vendors who can provide such encryption services. Before embarking on large-scale encryption of backup data, organizations considering this approach should thoroughly test the process. It is essential to verify that data encrypted before off-line storage can be decrypted and re-imported successfully.

## Solutions For Stored Data Encryption in Oracle Database

The `DBMS_CRYPTO` package provides several means for addressing the security issues that have been discussed. (For backward compatibility, `DBMS_OBFUSCATION_TOOLKIT` is also provided.) This section includes these topics:

- Oracle Database Data Encryption Capabilities
- Data Encryption Challenges

### Oracle Database Data Encryption Capabilities

While encryption is not the ideal solution for addressing a number of security threats, it is clear that selectively encrypting sensitive data before storage in the database does improve security. Examples of such data could include:

- Credit card numbers
- National identity numbers

To address these needs, Oracle Database provides the PL/SQL package `DBMS_CRYPTO` to encrypt and decrypt stored data. This package supports several industry-standard encryption and hashing algorithms, including the Advanced Encryption Standard (AES) encryption algorithm. AES has been approved by the National Institute of Standards and Technology (NIST) to replace the Data Encryption Standard (DES).

The `DBMS_CRYPTO` package enables encryption and decryption for common Oracle datatypes, including `RAW` and large objects (`LOB`s), such as images and sound. Specifically, it supports `BLOB`s and `CLOB`s. In addition, it provides Globalization Support for encrypting data across different database character sets.

The following cryptographic algorithms are supported:

- Data Encryption Standard (DES), Triple DES (3DES, 2-key)

- Advanced Encryption Standard (AES)
- MD5, MD4, and SHA-1 cryptographic hashes
- MD5 and SHA-1 Message Authentication Code (MAC)

Block cipher modifiers are also provided with DBMS_CRYPTO. You can choose from several padding options, including PKCS (Public Key Cryptographic Standard) #5, and from four block cipher chaining modes, including Cipher Block Chaining (CBC). (Padding must be done in multiples of eight bytes.)

Table 16–1 compares the DBMS_CRYPTO package features to the other PL/SQL encryption package, the DBMS_OBFUSCATION_TOOLKIT.

***Table 16–1 DBMS_CRYPTO and DBMS_OBFUSCATION_TOOLKIT Feature Comparison***

| Package Feature | DBMS_CRYPTO | DBMS_OBFUSCATION_TOOLKIT |
|---|---|---|
| Cryptographic algorithms | DES, 3DES, AES, RC4, 3DES_2KEY | DES, 3DES |
| Padding forms | PKCS5, zeroes | none supported |
| Block cipher chaining modes | CBC, CFB, ECB, OFB | CBC |
| Cryptographic hash algorithms | MD5, SHA-1, MD4 | MD5 |
| Keyed hash (MAC) algorithms | HMAC_MD5, HMAC_SH1 | none supported |
| Cryptographic pseudo-random number generator | RAW, NUMBER, BINARY_INTEGER | RAW, VARCHAR2 |
| Database types | RAW, CLOB, BLOB | RAW, VARCHAR2 |

DBMS_CRYPTO is intended to replace the obfuscation toolkit, since it is easier to use and supports a range of algorithms accommodating both new and existing systems. Although 3DES_2KEY and MD4 are provided for backward compatibility, you achieve better security using 3DES, AES, MD5, or SHA-1. Thus 3DES_2KEY and MD4 are not recommended.

The DBMS_CRYPTO package includes cryptographic checksumming capabilities (MD5), which are useful for compares, and the ability to generate a secure random number (the RANDOMBYTES function). Secure random number generation is an important part of cryptography; predictable keys are easily-guessed keys, and easily-guessed keys may lead to easy decryption of data. Most cryptanalysis is done by finding weak keys or poorly-stored keys, rather than through brute force analysis (cycling through all possible keys).

**Note:**

- Do not use DBMS_RANDOM as it is unsuitable for cryptographic key generation.
- For more detailed descriptions of both DBMS_CRYPTO and DBMS_OBFUSCATION_TOOLKIT, see also the *PL/SQL Packages and Types Reference.*

Key management is programmatic. That is, the application (or caller of the function) must supply the encryption key; and this means that the application developer must find a way of storing and retrieving keys securely. The relative strengths and weaknesses of various key management techniques are discussed in the sections that follow. The DBMS_OBFUSCATION_TOOLKIT package, which can handle both string and raw data, requires the submission of a 64-bit key. The DES algorithm itself has an effective key length of 56-bits.

The DBMS_OBFUSCATION_TOOLKIT is granted to PUBLIC by default. Oracle Corporation strongly recommends that you revoke this grant.

**Note:** While the DBMS_OBFUSCATION_TOOLKIT package can take either VARCHAR2 or RAW datatypes, it is preferable to use the RAW datatype for keys and encrypted data. Storing encrypted data as VARCHAR2 can cause problems if it passes through Globalization Support routines. For example, when transferring database to a database that uses another character set.

To convert between VARCHAR2 and RAW datatypes, use the CAST_TO_RAW and CAST_TO_VARCHAR2 funtions of the UTL_RAW package.

**See Also:** *PL/SQL Packages and Types Reference* for detailed documentation of the DBMS_CRYPTO, DBMS_OBFUSCATION_TOOLKIT and UTL_RAW packages

## Data Encryption Challenges

Even in cases where encryption can provide additional security, it is not without its technical challenges, as described in the following sections:

- Encrypting Indexed Data
- Key Management
- Key Transmission
- Key Storage
- Changing Encryption Keys
- Binary Large Objects (BLOBS)

## Encrypting Indexed Data

Special difficulties arise in handling encrypted data that is indexed. For example, suppose a company uses a national identity number—such as the U.S. Social Security number (SSN)—as the employee number for its employees. The company considers employee numbers to be very sensitive data and therefore wants to encrypt data in the `EMPLOYEE_NUMBER` column of the `EMPLOYEES` table. Because `EMPLOYEE_NUMBER` contains unique values, the database designers want to have an index on it for better performance.

However, if `DBMS_CRYPTO` or the `DBMS_OBFUSCATION_TOOLKIT` (or another mechanism) is used to encrypt data in a column, then an index on that column will also contain encrypted values. Although such an index can be used for equality checking (for example, `'SELECT * FROM emp WHERE employee_number = '123245'`), if the index on that column contains encrypted values, then the index is essentially unusable for any other purpose. Oracle therefore recommends that developers not encrypt indexed data.

Given the privacy issues associated with overuse of national identity numbers (for example, identity theft), the fact that some allegedly unique national identity numbers have duplicates (as with U.S. Social Security numbers), and the ease with which a sequence can generate a unique number, there are many good reasons to avoid using national identity numbers as unique IDs.

## Key Management

To address the issue of secure cryptographic key generation, Oracle Database provides support for secure random number generation, the `RANDOMBYTES` function of `DBMS_CRYPTO`. (This function replaces the capabilities provided by the `GetKey` procedure of the earlier `DBMS_OBFUSCATION_TOOLKIT`.) `DBMS_CRYPTO` calls the secure random number generator (RNG) previously certified by RSA. Developers should not, under any circumstances use the `DBMS_RANDOM` package. The `DBMS_RANDOM` package generates pseudo-random numbers, which, as

RFC-1750 states, "The use of pseudo-random processes to generate secret quantities can result in pseudo-security."

## Key Transmission

If the key is to be passed by the application to the database, then it must be encrypted. Otherwise, a snooper could grab the key as it is being transmitted. Use of network encryption, such as that provided by Oracle Advanced Security, will protect all data in transit from modification or interception, including cryptographic keys.

## Key Storage

Key storage is one of the most important, yet difficult, aspects of encryption. To recover data encrypted with a symmetric key, the key must be accessible to the application or user seeking to decrypt the data. The key needs to be easy enough to retrieve that users can access encrypted data, without significant performance degradation. The key needs to be secure enough not to be easily recoverable by someone who is maliciously trying to access encrypted data which he is not supposed to see.

The three basic options available to a developer are:

- Storing the Keys in the Database
- Storing the Keys in the Operating System
- Users Managing Their Own Keys

### Storing the Keys in the Database

Storing the keys in the database cannot always provide infallible security if you are trying to protect against the DBA accessing encrypted data. An all-privileged DBA could still access tables containing encryption keys. However, it can often provide quite good security against the casual snooper or against someone compromising the database file on the operating system.

As a trivial example, suppose you create a table (`EMP`) that contains employee data. You want to encrypt each employee's Social Security number (SSN) stored in one of the columns. You could encrypt each employee's SSN using a key which is stored in a separate column. However, anyone with `SELECT` access on the entire table could retrieve the encryption key and decrypt the matching SSN.

While this encryption scheme seems easily defeated, with a little more effort you can create a solution that is much harder to break. For example, you could encrypt the SSN using a technique that performs some additional data transformation on the `employee_number` before using it to encrypt the SSN. This technique might be something as simple, for example, as XORing the `employee_number` with the birthdate of the employee.

As additional protection, a PL/SQL package body performing encryption can be "wrapped," (using the `WRAP` utility) which obfuscates the code. A developer could wrap a package body called `KEYMANAGE` as follows:

```
wrap iname=/mydir/keymanage.sql
```

A developer can subsequently have a function in the package call the `DBMS_OBFUSCATION_TOOLKIT` with the key contained in the wrapped package.

While wrapping is not unbreakable, it makes it harder for a snooper to get the key. Because literals are still readable within the package file, the key could be split up in the package and then have the procedure reassemble it prior to use. Even in cases where a different key is supplied for each encrypted data value, not embedding the key value within a package, wrapping the package that performs key management (that is, data transformation or padding) is recommended.

**See Also:** *PL/SQL Packages and Types Reference* for additional information about the `WRAP` utility

An alternative would be to have a separate table in which to store the encryption key and to envelope the call to the keys table with a procedure. The key table can be joined to the data table using a primary key to foreign key relationship. For example, `EMPLOYEE_NUMBER` is the primary key in the `EMPLOYEES` table which stores employee information and the encrypted SSN. `EMPLOYEE_NUMBER` is a foreign key to the `SSN_KEYS` table which stores the encryption keys for each employee's SSN. The key stored in the `SSN_KEYS` table can also be transformed before use (by using XORing), so the key itself is not stored unencrypted. The procedure itself should be wrapped, to hide the way in which keys are transformed before use.

The strengths of this approach are:

- Users who have direct table access cannot see the sensitive data unencrypted, nor can they retrieve the keys to decrypt the data.
- Access to decrypted data can be controlled through a procedure that selects the encrypted data, retrieves the decryption key from the key table, and transforms it before it can be used to decrypt the data.

- The data transformation algorithm is hidden from casual snooping by wrapping the procedure, which obfuscates the procedure code.
- SELECT access to both the data table and the keys table does not guarantee that the user with this access can decrypt the data, because the key is transformed before use.

The weakness in this approach is that a user who has SELECT access to both the key table and the data table, who can derive the key transformation algorithm, can break the encryption scheme.

The preceding approach is not infallible, but it is good enough to protect against easy retrieval of sensitive information stored in clear text.

### Storing the Keys in the Operating System

Storing keys in a flat file in the operating system is another option. Oracle Database enables you to make callouts from PL/SQL, which you could use to retrieve encryption keys. However, if you store keys in the operating system and make callouts to it, then your data is only as secure as the protection on the operating system. If your primary security concern driving you to encrypt data stored in the database is that the database can be broken into from the operating system, then storing the keys in the operating system arguably makes it easier for a hacker to retrieve encrypted data than storing the keys in the database itself.

### Users Managing Their Own Keys

Having the user supply the key assumes the user will be responsible with the key. Considering that 40% of help desk calls are from users who have forgotten their passwords, you can see the risks of having users manage encryption keys. In all likelihood, users will either forget an encryption key, or write the key down, which then creates a security weakness. If a user forgets an encryption key or leaves the company, then your data is unrecoverable.

If you do elect to have user-supplied or user-managed keys, then you need to make sure you are using network encryption so the key is not passed from client to server in the clear. You also must develop key archive mechanisms, which is also a difficult security problem. Key archives or "backdoors" create the security weaknesses that encryption is attempting to address in the first place.

## Changing Encryption Keys

Prudent security practice dictates that you periodically change encryption keys. For stored data, this requires periodically unencrypting the data, and reencrypting it with another well-chosen key. This would likely have to be done while the data is not being accessed, which creates another challenge. This is especially true for a Web-based application encrypting credit card numbers, since you do not want to shut down the entire application while you switch encryption keys.

### Binary Large Objects (BLOBS)

Certain datatypes require more work to encrypt. For example, Oracle supports storage of binary large objects (BLOBs), which let users store very large objects (for example, multiple gigabytes) in the database. A BLOB can be either stored internally as a column, or stored in an external file.

For an example of using DBMS_CRYPTO on BLOB data, see the section entitled Example of Encrypt / Decrypt Procedures for BLOB Data on page 16-15.

## Example of a Data Encryption PL/SQL Program

The following sample PL / SQL program (dbms_crypto.sql) illustrates encrypting data. This example code does the following:

- DES-encrypts a string (VARCHAR2 type) after first converting it into RAW type.

  This step is necessary because encrypt and decrypt functions and procedures in dbms_crypto package work on RAW data type only, unlike functions / packages in dbms_obfuscation_toolkit package.

- Shows how to create 160-bit hash using SHA-1 algorithm.
- Demonstrates how MAC, a key-dependent one-way hash, can be computed using MD5 algorithm.

The dbms_crypto.sql procedure follows:

```
DECLARE
    input_string     VARCHAR2(16) := 'tigertigertigert';
    raw_input        RAW(128) :=
UTL_RAW.CAST_TO_RAW(CONVERT(input_string,'AL32UTF8','US7ASCII'));
    key_string       VARCHAR2(8)  := 'scottsco';
    raw_key          RAW(128) :=
UTL_RAW.CAST_TO_RAW(CONVERT(key_string,'AL32UTF8','US7ASCII'));
    encrypted_raw    RAW(2048);
    encrypted_string VARCHAR2(2048);
    decrypted_raw    RAW(2048);
    decrypted_string VARCHAR2(2048);
```

```
-- 1. Begin testing Encryption
BEGIN
    dbms_output.put_line('> Input String                     : ' ||
    CONVERT(UTL_RAW.CAST_TO_VARCHAR2(raw_input),'US7ASCII','AL32UTF8'));
    dbms_output.put_line('> ========= BEGIN TEST Encrypt =========');
    encrypted_raw := dbms_crypto.Encrypt(
        src => raw_input,
        typ => DBMS_CRYPTO.DES_CBC_PKCS5,
        key => raw_key);
       dbms_output.put_line('> Encrypted hex value              : ' ||
        rawtohex(UTL_RAW.CAST_TO_RAW(encrypted_raw)));
decrypted_raw := dbms_crypto.Decrypt(
        src => encrypted_raw,
        typ => DBMS_CRYPTO.DES_CBC_PKCS5,
        key => raw_key);
    decrypted_string :=
    CONVERT(UTL_RAW.CAST_TO_VARCHAR2(decrypted_raw),'US7ASCII','AL32UTF8');
dbms_output.put_line('> Decrypted string output          : ' ||
        decrypted_string);
if input_string = decrypted_string THEN
    dbms_output.put_line('> String DES Encyption and Decryption successful');
END if;
dbms_output.put_line('');
dbms_output.put_line('> ========= BEGIN TEST Hash =========');
    encrypted_raw := dbms_crypto.Hash(
        src => raw_input,
        typ => DBMS_CRYPTO.HASH_SH1);
dbms_output.put_line('> Hash value of input string       : ' ||
        rawtohex(UTL_RAW.CAST_TO_RAW(encrypted_raw)));
dbms_output.put_line('> ========= BEGIN TEST Mac =========');
    encrypted_raw := dbms_crypto.Mac(
        src => raw_input,
        typ => DBMS_CRYPTO.HMAC_MD5,
        key => raw_key);
dbms_output.put_line('> Message Authentication Code      : ' ||
        rawtohex(UTL_RAW.CAST_TO_RAW(encrypted_raw)));
dbms_output.put_line('');
dbms_output.put_line('> End of DBMS_CRYPTO tests  ');
END;
/
```

**See Also:** *PL/SQL User's Guide and Reference*

## Example of Encrypt/Decrypt Procedures for BLOB Data

The following sample PL/SQL program (blob_test.sql) illustrates encrypting and decrypting BLOB data. This example code does the following, printing out its progress (or problems) at each step:

- Creates a table for the BLOB column.
- Inserts the raw values into that table.
- Encrypts the raw data.
- Decrypts the encrypted data.

The blob_test.sql procedure follows:

```
-- Create a table for BLOB column.
create table table_lob (id number, loc blob);

-- insert 3 empty lobs for src/enc/dec
insert into table_lob values (1, EMPTY_BLOB());
insert into table_lob values (2, EMPTY_BLOB());
insert into table_lob values (3, EMPTY_BLOB());

set echo on
set serveroutput on

declare
    srcdata    RAW(1000);
    srcblob    BLOB;
    encrypblob BLOB;
    encrypraw  RAW(1000);
    encrawlen  BINARY_INTEGER;
    decrypblob BLOB;
    decrypraw  RAW(1000);
    decrawlen  BINARY_INTEGER;

    leng       INTEGER;

begin

    -- RAW input data 16 bytes
    srcdata := hextoraw('6D6D6D6D6D6D6D6D6D6D6D6D6D6D6D6D');

    dbms_output.put_line('---');
    dbms_output.put_line('input is ' || srcdata);
    dbms_output.put_line('---');
```

```
-- select empty lob locators for src/enc/dec
select loc into srcblob from table_lob where id = 1;
select loc into encrypblob from table_lob where id = 2;
select loc into decrypblob from table_lob where id = 3;

dbms_output.put_line('Created Empty LOBS');
dbms_output.put_line('---');

leng := DBMS_LOB.GETLENGTH(srcblob);
IF leng IS NULL THEN
    dbms_output.put_line('Source BLOB Len NULL ');
ELSE
    dbms_output.put_line('Source BLOB Len ' || leng);
END IF;

leng := DBMS_LOB.GETLENGTH(encrypblob);
IF leng IS NULL THEN
    dbms_output.put_line('Encrypt BLOB Len NULL ');
ELSE
    dbms_output.put_line('Encrypt BLOB Len ' || leng);
END IF;

leng := DBMS_LOB.GETLENGTH(decrypblob);
IF leng IS NULL THEN
    dbms_output.put_line('Decrypt  BLOB Len NULL ');
ELSE
    dbms_output.put_line('Decrypt BLOB Len ' || leng);
END IF;

-- write source raw data into blob
DBMS_LOB.OPEN (srcblob, DBMS_LOB.lob_readwrite);
DBMS_LOB.WRITEAPPEND (srcblob, 16, srcdata);
DBMS_LOB.CLOSE (srcblob);

dbms_output.put_line('Source raw data written to source blob');
dbms_output.put_line('---');

leng := DBMS_LOB.GETLENGTH(srcblob);
IF leng IS NULL THEN
    dbms_output.put_line('source BLOB Len NULL ');
ELSE
    dbms_output.put_line('Source BLOB Len ' || leng);
END IF;
```

```
/*
* Procedure Encrypt
* Arguments: srcblob -> Source BLOB
*            encrypblob -> Output BLOB for encrypted data
*            DBMS_CRYPTO.AES_CBC_PKCS5 -> Algo : AES
*                                         Chaining : CBC
*                                         Padding : PKCS5
*            256 bit key for AES passed as RAW
*                ->
hextoraw('000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F')
*            IV (Initialization Vector) for AES algo passed as RAW
*                -> hextoraw('00000000000000000000000000000000')
*/

DBMS_CRYPTO.Encrypt(encrypblob,
                srcblob,
                DBMS_CRYPTO.AES_CBC_PKCS5,
                hextoraw
    ('000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F'),
                hextoraw('00000000000000000000000000000000'));


dbms_output.put_line('Encryption Done');
dbms_output.put_line('---');

leng := DBMS_LOB.GETLENGTH(encrypblob);
IF leng IS NULL THEN
    dbms_output.put_line('Encrypt BLOB Len NULL');
ELSE
    dbms_output.put_line('Encrypt BLOB Len ' || leng);
END IF;

-- Read encrypblob to a raw
encrawlen := 999;

DBMS_LOB.OPEN (encrypblob, DBMS_LOB.lob_readwrite);
DBMS_LOB.READ (encrypblob, encrawlen, 1, encrypraw);
DBMS_LOB.CLOSE (encrypblob);

dbms_output.put_line('Read encrypt blob to a raw');
dbms_output.put_line('---');

dbms_output.put_line('Encrypted data is (256 bit key) ' || encrypraw);
dbms_output.put_line('---');
```

```
/*
* Procedure Decrypt
* Arguments: encrypblob -> Encrypted BLOB to decrypt
*            decrypblob -> Output BLOB for decrypted data in RAW
*            DBMS_CRYPTO.AES_CBC_PKCS5 -> Algo : AES
*                                         Chaining : CBC
*                                         Padding : PKCS5
*            256 bit key for AES passed as RAW (same as used during Encrypt)
*                ->
hextoraw('000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F')
*            IV (Initialization Vector) for AES algo passed as RAW (same as
used during Encrypt)
*                -> hextoraw('00000000000000000000000000000000')
*/

DBMS_CRYPTO.Decrypt(decrypblob,
                encrypblob,
                DBMS_CRYPTO.AES_CBC_PKCS5,
                hextoraw
    ('000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F'),
                hextoraw('00000000000000000000000000000000'));

leng := DBMS_LOB.GETLENGTH(decrypblob);
IF leng IS NULL THEN
    dbms_output.put_line('Decrypt BLOB Len NULL');
ELSE
    dbms_output.put_line('Decrypt BLOB Len ' || leng);
END IF;

-- Read decrypblob to a raw
decrawlen := 999;

DBMS_LOB.OPEN (decrypblob, DBMS_LOB.lob_readwrite);
DBMS_LOB.READ (decrypblob, decrawlen, 1, decrypraw);
DBMS_LOB.CLOSE (decrypblob);

dbms_output.put_line('Decrypted data is (256 bit key) ' || decrypraw);
dbms_output.put_line('---');

DBMS_LOB.OPEN (srcblob, DBMS_LOB.lob_readwrite);
DBMS_LOB.TRIM (srcblob, 0);
DBMS_LOB.CLOSE (srcblob);

DBMS_LOB.OPEN (encrypblob, DBMS_LOB.lob_readwrite);
DBMS_LOB.TRIM (encrypblob, 0);
```

```
    DBMS_LOB.CLOSE (encrypblob);

    DBMS_LOB.OPEN (decrypblob, DBMS_LOB.lob_readwrite);
    DBMS_LOB.TRIM (decrypblob, 0);
    DBMS_LOB.CLOSE (decrypblob);

end;
/

truncate table table_lob;
drop table table_lob;
```

# Glossary

### application roles

Database roles that are granted to application users and that are secured by embedding passwords inside the application. See also **secure application roles**

### definer's rights procedures

Definer's rights procedures execute with the privileges of their owner, not their current user. Such definer's rights sub-programs are bound to the schema in which they reside. For example, assume that user `blake` and user `scott` each have a table called `dept` in their respective user schemas. If user `blake` calls a definer's rights procedure, which is owned by user `scott`, to update the `dept` table, then this procedure will update the `dept` table in the `scott` schema because this procedure executes with the privileges of the user who owns (defined) the procedure.

### Forwardable Ticket Granting Ticket

A special Kerberos ticket that can be forwarded to proxies permits the proxy to obtain additional Kerberos tickets on behalf of the client for proxy authentication. See also **Kerberos ticket**

### invoker's rights procedures

Invoker's rights procedures execute with the privileges of the current user, that is, the user who invokes the procedure. Such procedures are not bound to a particular schema. They can be run by a variety of users and allow multiple users to manage their own data by using centralized application logic. Invoker's rights procedures are created with the `AUTHID` clause in the declaration section of the procedure code.

### KDC

See **Key Distribution Center**

## Kerberos ticket

A temporary set of electronic credentials that verify the identity of a client for a particular service. Also referred to as a service ticket.

## Key Distribution Center

(KDC) A machine that issues Kerberos tickets. See also **Kerberos ticket**

## secure application roles

Like an application roles, a secure application role is a database role that is granted to application users, but it is secured by using an Invoker's Right stored procedure to retrieve the role password from a database table. A secure application role password is not embedded in the application. See also **application roles**

## service ticket

See **Kerberos ticket**

# Index