Oracle® Database

Security Guide

11*g* Release 1 (11.1)

B28531-01

July 2007

ORACLE®

Oracle Database Security Guide 11*g* Release 1 (11.1)

B28531-01

Primary Author: Patricia Huey

Contributors: Sumit Jeloka, Don Gosselin , Nina Lewis, Bryn Llewellyn, Narendra Manappa, Gopal Mulagund, Janaki Narasinghanallur, Paul Needham, Deb Owens, Robert Pang, Vipin Samar, Digvijay Sirmukaddam, Richard Smith, Sachin Sonawane, James Spiller, Ashwini Surpur, Srividya Tata, Kamal Tbeileh, Daniel Wong

Contents

3 Configuring Authentication

4 Configuring Privilege and Role Authorization

5 Managing Security for Application Developers

6 Configuring Auditing

7 Using Application Contexts to Retrieve User Information

8 Using Oracle Virtual Private Database to Control Data Access

9 Developing Applications Using the Data Encryption API

10 Keeping Your Oracle Database Secure

List of Examples

List of Figures

List of Tables

Preface

Welcome to *Oracle Database Security Guide*. This guide describes how you can configure security for Oracle Database by using the default database features.

This preface contains these topics:

- Audience
- Documentation Accessibility
- Related Documents
- Conventions

Audience

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 data of an organization, users, and applications from accidental, inappropriate, or unauthorized actions
- Creating and enforcing policies and practices of auditing and accountability for 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 ease of use

To use this document, you need a basic understanding of how and why a database is used, and basic familiarity with SQL.

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. Accessibility 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 more information, visit the Oracle Accessibility Program Web site at

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

Accessibility of Code Examples in Documentation

Screen readers 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, some screen readers 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.

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

Related Documents

For more security-related information, see these Oracle resources:

- *Oracle Database Administrator's Guide*
- *Oracle Database 2 Day DBA*
- *Oracle Database 2 Day + Security Guide*
- *Oracle Database Concepts*
- *Oracle Database Reference*
- *Oracle Database Vault Administrator's Guide*

Many of the examples in this guide use the sample schemas of the seed database, which you can create when you install Oracle Database. See *Oracle Database Sample Schemas* for information about how these schemas were created and how you can use them yourself.

Oracle Store

Printed documentation is available for sale in the Oracle Store at

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

Oracle Technology Network (OTN)

You can download free release notes, installation documentation, updated versions of this guide, white papers, or other collateral from the Oracle Technology Network (OTN). Visit

`http://www.oracle.com/technology/index.html`

If you are not already a member, you can register for free at

`http://www.oracle.com/technology/membership/`

For security-specific information on OTN, visit

http://www.oracle.com/technology/deploy/security/index.html

For the latest version of the Oracle documentation, including this guide, visit

http://www.oracle.com/technology/documentation/index.html

Oracle Documentation Search Engine

To access the database documentation search engine directly, visit

http://tahiti.oracle.com/

Oracle*MetaLink*

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

http://metalink.oracle.com/

Conventions

The following text conventions are used in this document:

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

What's New in Oracle Database Security?

The Oracle Database 11*g* Release 1 (11.1) security features and enhancements described in this section comprise the overall effort to provide superior access control, privacy, and accountability with this release of Oracle Database.

The following sections describe new security features of Oracle Database 11*g* Release 1 (11.1) and provide pointers to additional information:

- Automatic Secure Configuration
- New Password Protections
- SYSDBA and SYSOPER Strong Authentication
- SYSASM Privilege for Automatic Storage Management
- Encryption Enhancements
- Fine-Grained Access Control on Network Services on the Database
- Oracle XML DB Security Enhancements
- Directory Security Enhancements
- Oracle Call Interface Security Enhancements

Automatic Secure Configuration

When you create a new database, you can use Database Configuration Assistant (DBCA) to automatically create a more secure configuration than in previous releases of Oracle Database. You can enable the following secure configuration settings in one operation:

- **Password-specific settings in the default profile.** This feature enables you to enforce password expiration and other password policies. See "Configuring Password Settings in the Default Profile" on page 3-10 for more information.
- **Auditing.** This feature enables auditing for specific events such as database connections. See "Using Default Auditing for Security-Relevant SQL Statements and Privileges" on page 6-10 for more information.

To configure your database for greater security, follow the guidelines in Chapter 10, "Keeping Your Oracle Database Secure".

New Password Protections

Oracle Database now includes the following new password protections:

- **Password complexity verification.** Password complexity verification ensures that users set complex passwords when setting or resetting passwords. You can enforce password complexity by using the default settings provided by Oracle Database, or create custom requirements to further secure the password complexity requirements for your site.

  "Enforcing Password Complexity Verification" on page 3-7 describes built-in password verification.

- **Enforced case sensitivity.** See "Enabling or Disabling Password Case Sensitivity" on page 3-8 for more information.
- **Stronger password hashing algorithm.** See "What Are the Oracle Database Built-in Password Protections?" on page 3-2 for more information.

SYSDBA and SYSOPER Strong Authentication

You can now use the Secure Sockets Layer (SSL) and Kerberos strong authentication methods to authenticate users who have the `SYSDBA` and `SYSOPER` privileges.

See "Strong Authentication and Centralized Management for Database Administrators" on page 3-16 for more information.

SYSASM Privilege for Automatic Storage Management

The `SYSASM` system privilege has been added to Oracle Database 11*g* Release 1 (11.1), to be used exclusively to administer Automatic Storage Management (ASM). Use the `SYSASM` privilege instead of the `SYSDBA` privilege to connect to and administer ASM instances.

See *Oracle Database Storage Administrator's Guide* for more information about the `SYSASM` privilege.

Encryption Enhancements

This section describes the following enhancements in encryption:

- Intelligent LOB Compression, Deduplication, and Encryption with SecureFiles
- Compressed and Encrypted Dump File Sets
- Transparent Data Encryption with Hardware Security Module Integration
- Transparent Tablespace Encryption

Intelligent LOB Compression, Deduplication, and Encryption with SecureFiles

Oracle Database supports a new, faster, and scalable Large Object (LOB) storage paradigm called SecureFiles. SecureFiles, in addition to performance, supports efficient compression, deduplication (that is, coalescing duplicate data), and encryption. LOB data can now be encrypted with Oracle Database, and is available for random reads and writes.

For more information about SecureFiles, see *Oracle Database SecureFiles and Large Objects Developer's Guide*. See also *Oracle Database SQL Language Reference* for updates in the `CREATE TABLE` and `ALTER TABLE` statements to support this feature.

Compressed and Encrypted Dump File Sets

In this release, you can use Oracle Data Pump to compress and encrypt an entire dump file set. You can optionally compress and encrypt the data, metadata, or complete dump file set during an Oracle Data Pump export.

For more information, see *Oracle Database Utilities*.

Transparent Data Encryption with Hardware Security Module Integration

Transparent data encryption (TDE) stores the master key in an encrypted software wallet and uses this key to encrypt the column keys, which in turn encrypt column data. While this approach to key management is sufficient for many applications, it may not be sufficient for environments that require stronger security. Because the master key must reside in memory to perform cryptographic operations, an intruder could perform various types of logical attacks to dump the memory and then retrieve the key. To avoid the problem of insecure system memory, the transparent data encryption functionality is extended to use hardware security modules (HSMs). This enhancement offers far better physical and logical protection of the master keys.

This release focuses on storing the master key within the hardware security module at all times and limiting the hardware security module to the encryption and decryption of the column keys. The column keys are passed back to the database. Oracle recommends that you encrypt the traffic between HSM device and databases with Advanced Security Option Network Encryption. This new feature provides additional security for transparent data encryption, because the master key cannot leave the HSM device. Furthermore, it enables the sharing of the same key between multiple databases and instances in an Oracle Real Applications Clusters (RAC) environment.

To configure transparent data encryption with hardware security module integration, see *Oracle Database Advanced Security Administrator's Guide*.

Transparent Tablespace Encryption

Transparent tablespace encryption enables you to encrypt an entire tablespace. This encryption includes all the data within the tablespace. When an application accesses the tablespace, Oracle Database transparently decrypts the relevant data blocks for the application.

Tablespace encryption provides an alternative to transparent data encryption column encryption. This eliminates the need for granular analysis of applications to determine which columns to encrypt, especially for applications with a large number of columns containing personally identifiable information (PII) such as social security numbers or patient health care records. If your tables have small amounts of data to encrypt, you can continue to use the transparent data encryption column encryption solution.

For an introduction to transparent encryption, see *Oracle Database 2 Day + Security Guide*. For detailed information about transparent tablespace encryption, see *Oracle Database Advanced Security Administrator's Guide*.

Fine-Grained Access Control on Network Services on the Database

Oracle Database provides a set of PL/SQL utility packages, such as `UTL_TCP`, `UTL_SMTP`, `UTL_MAIL`, `UTL_HTTP`, and `UTL_INADDR`, that are designed to enable database users to access network services on the database. *Oracle Database PL/SQL Packages and Types Reference* describes the PL/SQL utility packages in detail.

In a default database installation, these packages are created with `EXECUTE` privileges granted to `PUBLIC` users. This release enhances the security of these packages by

providing database administrators the ability to control access to applications in the database that use these packages.

See "Managing Fine-Grained Access to External Network Services" on page 4-44 for more information.

Oracle XML DB Security Enhancements

This section describes the following Oracle XML DB security enhancements:

- XML Translation Support for Oracle Database XML
- Support for Web Services

XML Translation Support for Oracle Database XML

Security objects are now stored in the Oracle XML DB repository as XMLType objects. These security objects can contain strings that need to be translated to different languages so that they can be searched or displayed in those languages. Developers can store translated strings with the XMLType and retrieve and operate on these strings depending on the language settings of the user. The advantage of this feature is that it reduces the costs associated with developing applications that are independent of the target preferred language of the user.

To configure security for XMLType objects, see *Oracle XML DB Developer's Guide*.

Support for Web Services

You can now use the Oracle XML DB HTTP server for service-oriented architecture (SOA) operations. This allows the database to be treated as simply another service provider in an SOA environment. Security administrators can control user access to Oracle Database Web services and their associated database objects by using the `XDB_WEBSERVICES`, `XDB_WEBSERVICES_OVER_HTTP`, and `XDB_WEBSERVICES_WITH_PUBLIC` predefined roles.

To configure Oracle Database Web services, see *Oracle XML DB Developer's Guide*.For information on this feature's predefined roles, see Table 4–3, " Oracle Database Predefined Roles" on page 4-10.

Directory Security Enhancements

In this release, administrators can now disallow anonymous access to database service information in a directory and require clients to authenticate when performing LDAP directory-based name look-ups. If you are using Microsoft Active Directory-based name lookups, then Oracle Database uses the native operating system-based authentication. If you are using Oracle Internet Directory (OID)-based name lookups, then Oracle Database performs authentication by using wallets.

To configure directory security, see *Oracle Database Net Services Reference*.

Oracle Call Interface Security Enhancements

The following security enhancements are available for Oracle Call Interface (OCI):

- Reporting bad packets that may come from malicious users or intruders
- Terminating or resuming the client or server process on receiving a bad packet
- Configuring the maximum number of authentication attempts

- Controlling the display of the Oracle database version banner, to prevent intruders from finding information about the security vulnerabilities present in the database software based on the version
- Adding banner information, such as "Unauthorized Access" and "User Actions Audited," to server connections so that clients can display this information

Database administrators can manage these security enhancements for Oracle Call Interface developers by configuring a set of new initialization parameters. See "Parameters for Enhanced Security of Database Communication" on page 5-11 for more information. See also *Oracle Call Interface Programmer's Guide* for detailed information on Oracle Call Interface.

1

Introducing Oracle Database Security

Oracle Database Security Guide describes how you can use Oracle Database features to secure your database installation.

This chapter includes the following topics:

- About Oracle Database Security
- Additional Database Security Resources

About Oracle Database Security

You can use the default Oracle Database features to configure security in the following areas for your Oracle Database installation:

- **User accounts.** When you create user accounts, you can secure them in a variety of ways. You can also create password profiles to better secure password policies for your site. Chapter 2, "Managing Security for Oracle Database Users" describes how to manage user accounts.
- **Authentication methods.** Oracle Database provides several ways to configure authentication for users and database administrators. For example, you can authenticate users on the database level, from the operating system, and on the network. Chapter 3, "Configuring Authentication" describes how authentication in Oracle Database works.
- **Privileges and roles.** You can use privileges and roles to restrict user access to data. Chapter 4, "Configuring Privilege and Role Authorization" describes how to create and manage user privileges and roles.
- **Application security.** The first step to creating a database application is to ensure that it is properly secure. Chapter 5, "Managing Security for Application Developers" discusses how to incorporate application security into your application security policies.
- **Auditing database activities.** You can audit database activities in general terms, such as auditing all SQL statements, SQL privileges, schema objects, and network activity. Or, you can audit in a granular manner, such as when the IP addresses from outside the corporate network is being used. Chapter 6, "Configuring Auditing" describes how to configure database auditing.
- **User session information using application context.** An application context is a name-value pair that holds the session information. You can retrieve session information about a user, such as the user name or terminal, and restrict database and application access for that user based on this information. Chapter 7, "Using Application Contexts to Retrieve User Information" describes how to use application context.

- **Database access on the row and column level using Virtual Private Database.** A Virtual Private Database policy dynamically imbeds a `WHERE` predicate into SQL statements the user issues. Chapter 8, "Using Oracle Virtual Private Database to Control Data Access" describes how to create and manage Virtual Private Database policies.
- **Encryption.** You can disguise data on the network to prevent unauthorized access to that data. Chapter 9, "Developing Applications Using the Data Encryption API" explains how to use the `DBMS_CRYPTO` and `DBMS_SQLHASH` PL/SQL packages to encrypt data.

In addition, Chapter 10, "Keeping Your Oracle Database Secure" provides guidelines that you should follow when you secure your Oracle Database installation.

Additional Database Security Resources

In addition to the security resources described in this guide, Oracle Database provides the following database security products:

- **Advanced security features.** See *Oracle Database Advanced Security Administrator's Guide* for information about advanced features such as transparent data encryption, wallet management, network encryption, and the RADIUS, Kerberos, Secure Sockets Layer authentication.
- **Oracle Label Security.** Oracle Label Security secures database tables at the row level, allowing you to filter user access to row data based on privileges. See *Oracle Label Security Administrator's Guide* for detailed information about Oracle Label Security.
- **Oracle Database Vault.** Oracle Database Vault provides fine-grained access control to your sensitive data, including protecting data from superprivileged users. *Oracle Database Vault Administrator's Guide* describes how to use Oracle Database Vault.
- **Oracle Audit Vault.** Oracle Audit Vault audits database data from sources such as Oracle Database audit trail tables, database operating system audit files, and database redo logs. Using Oracle Audit Vault, you can create alerts on suspicious activities, and create reports on the history of privileged user changes, schema modifications, and even data-level access. *Oracle Audit Vault Administrator's Guide* explains how to administer Oracle Audit Vault.
- **Oracle Enterprise User Security.** Oracle Enterprise User Security enables you to manage user security at the enterprise level. *Oracle Database Enterprise User Security Administrator's Guide* explains how to configure Oracle Enterprise User Security.

In addition to these products, you can find the latest information about Oracle Database security, such as new products and important information about security patches and alerts, by visiting the Security Technology Center on Oracle Technology Network at

`http://www.oracle.com/technology/deploy/security/index.html`

2

Managing Security for Oracle Database Users

This chapter describes how to manage security for Oracle Database users. It discusses the following topics:

- About User Security
- Creating User Accounts
- Altering User Accounts
- Configuring User Resource Limits
- Deleting User Accounts
- Finding Information About Database Users and Profiles

About User Security

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. Oracle Database enables you to set up security for your users in a variety of ways. When you create user accounts, you can specify limits to the user account. You can also set limits on the amount of various system resources available to each user as part of the security domain of that user. Oracle Database provides a set of database views that you can query to find information such as resource and session information. This chapter also describes profiles. A profile is collection of attributes that apply to a user. It enables a single point of reference for any of multiple users that share those exact attributes.

Another way to manage user security is to assign users privileges and roles. Chapter 4, "Configuring Privilege and Role Authorization" provides detailed information.

Creating User Accounts

This section describes the following aspects of creating a user account:

- Creating a New User Account
- Specifying a User Name
- Assigning the User a Password
- Assigning a Default Tablespace for the User
- Assigning a Tablespace Quota for the User
- Assigning a Temporary Tablespace for the User

- Specifying a Profile for the User
- Setting a Default Role for the User

For guidelines about creating and managing user accounts and passwords, see the following sections:

- "Guidelines for Securing User Accounts and Privileges" on page 10-2
- "Guidelines for Securing Passwords" on page 10-6

Creating a New User Account

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 database administrator or security administrator is usually the only user who has the `CREATE USER` system privilege.

Example 2–1 creates a user and specifies the user password, default tablespace, temporary tablespace where temporary segments are created, tablespace quotas, and profile. It also grants the user the minimum privileges, `CONNECT` and `CREATE SESSION`, to log in to the database session.

Example 2–1 Creating a User Account with CONNECT and CREATE SESSION Privileges

```
CREATE USER jward
 IDENTIFIED BY fjk222kll
 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 you grant the user the `CREATE SESSION` system privileges. So, immediately after you create the user account, use the `GRANT` SQL statement to grant the user these privileges. If the user needs to access Oracle Enterprise Manager, you should also grant the user the `SELECT ANY DICTIONARY` privilege.

Note: As a security administrator, you should create your own roles and assign only those privileges that are needed. For example, many users formerly granted the `CONNECT` privilege did not need the additional privileges `CONNECT` used to provide. Instead, only `CREATE SESSION` was actually needed, and in fact, that is the only privilege `CONNECT` presently retains.

Creating organization-specific roles gives an organization detailed control of the privileges it assigns, and protects it in case Oracle Database changes the roles that it defines in future releases. For example, both `CONNECT` and `RESOURCE` roles will be deprecated in future Oracle Database releases. Chapter 4, "Configuring Privilege and Role Authorization" discusses how to create and manage roles.

Specifying a User 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.

```
CREATE USER jward
 ...
```

Assigning the User a Password

In Example 2–1 on page 2-2, 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. To specify a password for the user, use the `IDENTIFIED BY` clause in the `CREATE USER` statement.

```
CREATE USER jward
 IDENTIFIED BY dkjkj233l
 ...
```

Chapter 3, "Configuring Authentication" describes how to select and specify the method of user authentication.

Assigning a Default Tablespace for the User

Each user should have a default tablespace. When a user creates a schema object and specifies no tablespace to contain it, Oracle Database stores the object in the default user tablespace.

The default setting for the default tablespaces of all users is the `SYSTEM` tablespace. If a user does not create objects, and has no privileges to do so, then this default setting is fine. However, if a user is likely to create any type of object, then you should specifically assign the user a default tablespace, such as the `USERS` tablespace. Using a tablespace other than `SYSTEM` reduces contention between data dictionary objects and user objects for the same data files. In general, do not store user data in the `SYSTEM` tablespace.

You can use the `CREATE TABLESPACE` SQL statement to 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 nonfunctional. This default permanent tablespace is not used by system users, that is, `SYS`, `SYSTEM`, and `OUTLN`, whose default permanent tablespace is `SYSTEM`. A tablespace designated as the default permanent tablespace cannot be dropped. To accomplish this goal, you must first designate another tablespace as the default permanent tablespace. You can use the `ALTER TABLESPACE` SQL statement to alter the default permanent tablespace to another tablespace. Be aware that this will affect all users or objects created after the `ALTER` DDL statement commits.

You can also set a user default tablespace during user creation, and change it later with the `ALTER USER` statement. Changing the user default tablespace affects only objects created after the setting is changed.

When you specify the default tablespace for a user, also specify a quota on that tablespace.

In Example 2–1 on page 2-2, the default tablespace for user `jward` is `data_ts`, and his quota on that tablespace is `500K`:

```
CREATE USER jward
 DEFAULT TABLESPACE data_ts
 ...
 QUOTA 500K ON data_ts
 ...
```

Assigning a Tablespace Quota for the User

You can assign each user a tablespace quota for any tablespace (except a temporary tablespace). Assigning a quota accomplishes the following:

- Users with privileges to create certain types of objects can create those objects in the specified tablespace.
- Oracle Database 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, then you must assign a quota to allow the user to create objects. At a minimum, assign users a quota for the default tablespace, and additional quotas for other tablespaces in which they can create objects.

Example 2–1 on page 2-2 assigns the following quotas for the `test_ts` and `data_ts` tablespaces:

```
CREATE USER jward
 ...
 QUOTA 100M ON test_ts
 QUOTA 500K ON data_ts
...
```

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 using too much space in the database.

You can assign quotas to a user tablespace when you create the user, or add or change quotas later. (You can find existing user quotas by querying the `USER_TS_QUOTAS` view.) If a new quota is less than the old one, then the following conditions remain true:

- If a user has already exceeded a new tablespace quota, then the objects of a user in the tablespace cannot be allocated more space until the combined space of these objects is less than the new quota.
- If a user has not exceeded a new tablespace quota, or if the space used by the objects of the user in the tablespace falls under a new tablespace quota, then the user's objects can be allocated space up to the new quota.

Revoking the Ability for Users to Create Objects in a Tablespace

You can revoke the ability of a user to create objects in a tablespace by using the `ALTER USER` SQL statement to change the current quota of the user to zero. After a quota of zero is assigned, the objects of the user in the tablespace remain, but the user cannot create new objects, nor can existing objects be allocated any new space.

Granting Users the 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, then explicit quotas again take effect. You can grant this privilege only to users, not to roles.

Before granting the `UNLIMITED TABLESPACE` system privilege, you must 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 selective or restricted access only after revoking the privilege.

Assigning a Temporary Tablespace for the User

You should assign each user a temporary tablespace. When a user executes a SQL statement that requires a temporary segment, Oracle Database stores the segment in the temporary tablespace of the user. These temporary segments are created by the system when performing sort or join operations. Temporary segments are owned by `SYS`, which has resource privileges in all tablespaces.

In Example 2–1 on page 2-2, the temporary tablespace of `jward` is `temp_ts`, a tablespace created explicitly to contain only temporary segments.

```
CREATE USER jward
 ...
 TEMPORARY TABLESPACE temp_ts
 ...
```

To create a temporary tablespace, use the `CREATE TEMPORARY TABLESPACE` SQL statement.

If you do not explicitly assign the user a temporary tablespace, then Oracle Database assigns the user 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, then the default is the `SYSTEM` tablespace or another permanent default established by the system administrator. Do not store user data in the `SYSTEM` tablespace. 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 locally managed permanent tablespaces.

You can set the temporary tablespace for a user at user creation, and change it later using the `ALTER USER` statement. If you are logged in as user `SYS`, you can set a quota for the temporary tablespace, as well as other space allocations. (Only user `SYS` can do this, because all space in the temporary tablespace belongs to user `SYS`.) You can also establish tablespace groups instead of assigning individual temporary tablespaces.

See Also:

- "Temporary Tablespaces" in *Oracle Database Administrator's Guide*
- "Multiple Temporary Tablespaces: Using Tablespace Groups" in *Oracle Database Administrator's Guide*

Specifying a Profile for the User

You can 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 you do not specify a profile, then Oracle Database assigns the user a default profile.

Example 2–1 on page 2-2 demonstrates how to assign a user a profile.

```
CREATE USER jward
 ...
 PROFILE clerk;
...
```

See Also: "Managing Resources with Profiles" on page 2-11

Setting a Default Role for the User

A role is a named group of related privileges that you grant as a group to users or other roles. A default role is automatically enabled for a user when the user creates a session. You can assign a user zero or more default roles.

You cannot set default roles for a user in the `CREATE USER` statement. When you first create a user, the default role setting for the user is `ALL`, which causes all roles subsequently granted to the user to be default roles. Use the `ALTER USER` statement to change the default roles for the user. For example:

```
GRANT USER jward clerk_role;

ALTER USER jward DEFAULT ROLE clerk_role;
```

Before a role can be made the default role for a user, that user needs to have been already granted the role.

See Also: "Managing User Roles" on page 4-6

Altering User Accounts

Users can change their own passwords. However, to change any other option of a user security domain, you must have the `ALTER USER` system privilege. Security administrators are typically the only users that have this system privilege, as it allows a modification of *any* user 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 user security settings with the `ALTER USER` SQL statement. Changing user security settings affects the future user sessions, not current sessions.

Example 2–2 shows how to use the `ALTER USER` statement to alter the security settings for the user `avyrros`:

Example 2–2 Altering a User Account

```
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 the security settings for the user avyrros as follows:

- Authentication is changed to use the operating system account of the user avyrros.
- The default and temporary tablespaces are explicitly set for user AVYRROS.
- The user avyrros is given a 100M quota for the DATA_TS tablespace.
- The quota on the test_ts is revoked for the user avyrros.
- The user avyrros is assigned the clerk profile.

Changing the User Password

Most users can change their own passwords with the PASSWORD statement, as follows:

```
PASSWORD andy
Changing password for andy
New password: new_password
Retype new password: new_password
```

No special privileges (other than those to connect to the database and create a session) are required for a user to change passwords. Encourage users to change their passwords frequently. "Guidelines for Securing Passwords" on page 10-6 provides advice on the best ways to secure passwords. You can find existing users for the current database instance by querying the ALL_USERS view.

Users can also use the ALTER USER SQL statement change their passwords. For example:

```
ALTER USER andy
 IDENTIFIED BY quit_slackin2day
```

However, for better security, use the PASSWORD statement to change the account's password. The ALTER USER statement displays the new password on the screen, where it can be seen by any overly curious coworkers. The PASSWORD command does not display the new password, so it is only known to you, not to your co-workers. In both cases, the password is encrypted on the network.

Users must have the PASSWORD and ALTER USER privilege to switch between methods of authentication. Usually, only an administrator has this privilege.

> **See Also:** Chapter 3, "Configuring Authentication" for information about authentication methods that are available for Oracle Database users

Configuring User Resource Limits

This section explores the following topics:

- About User Resource Limits
- Types of System Resources and Limits
- Determining Values for Resource Limits
- Managing Resources with Profiles

About User Resource Limits

You can set limits on the amount of various system resources available to each user as part of the security domain of that user. By doing so, you can prevent the uncontrolled consumption of valuable system resources such as CPU time. To set resource limits, you use Database Resource Manager, which is described in *Oracle Database Administrator's Guide*.

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 user consumption of system resources is less likely to have a detrimental impact.

You manage user resource limits by using 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 Database allows the security administrator to enable or disable the enforcement of profile resource limits universally.

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

> **See Also:** *Oracle Database Administrator's Guide* for detailed information about managing resources

Types of System Resources and Limits

Oracle Database 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, call level, or both, as discussed in the following sections:

- Limiting the User Session Level
- Limiting Database Call Levels
- Limiting CPU Time
- Limiting Logical Reads
- Limiting Other Resources

Limiting the User Session Level

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

If a user exceeds a session-level resource limit, then Oracle Database terminates (rolls back) the current statement and returns a message indicating that 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 cannot accomplish any more work during the current session.

Limiting Database Call Levels

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

If a user exceeds a call-level resource limit, then Oracle Database 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 session remains connected.

Limiting CPU Time

When SQL statements and other types of calls are made to Oracle Database, a certain 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 use a large amount of CPU time, reducing CPU time available for other processing.

To prevent uncontrolled use of CPU time, you can set fixed or dynamic limits on the CPU time for each call and the total amount of CPU time used for Oracle Database 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.

Limiting Logical Reads

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, you can 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 Database 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 calls in a session reaches the idle time limit, then the current transaction is rolled back, the session is terminated, and the resources of the session are returned to the system. The next call receives an error that indicates that 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 terminated because it has exceeded an idle time limit, the process monitor (PMON) background process cleans up after the terminated session. Until PMON completes this process, the terminated session is still counted in any session or user resource limit.

- **You can limit the elapsed connect time for each session.** If the duration of a session exceeds the elapsed time limit, then 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 Database does not constantly monitor the elapsed idle time or elapsed connection time. Doing so reduces system performance. Instead, it checks every few minutes. Therefore, a session can exceed this limit slightly (for example, by 5 minutes) before Oracle Database enforces the limit and terminates the session.

- **You can limit the amount of private System Global Area (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 Program Global Area (PGA). This limit is set as a number of bytes of memory in the SGA of an instance. Use the characters **K** or **M** to specify kilobytes or megabytes.

  See Also: For instructions about enabling or disabling resource limits:

  - "Finding Information About Database Users and Profiles" on page 2-13
  - "Managing User Roles" on page 4-6
  - *Oracle Database Administrator's Guide* for detailed information about managing resources

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 usually perform a high number of logical data block reads, then use the `ALTER RESOURCE COST` SQL statement to set the `LOGICAL_READS_PER_SESSION` setting 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`.

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

See Also:

- "Using Data Dictionary Views to Find Information About Users and Profiles" on page 2-13
- Chapter 6, "Configuring Auditing"
- *Oracle Database 2 Day DBA* for more information about Database Control
- Enterprise Manager online Help for more information about the Monitor feature

Managing Resources with Profiles

A **profile** is a named set of resource limits and password parameters that restrict database usage and instance resources for a user. You can assign a profile to each user, and a default profile to all others. Each user can have only one profile, and creating a new one supersedes an earlier version.

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 those exact attributes. User profiles in Oracle Internet Directory contain 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. You can use Database Resource Manager to set these types of resource limits.

Profile resource limits are enforced only when you enable resource limitation for the associated database. Enabling this limitation can occur either before starting up the database (using the `RESOURCE_LIMIT` initialization parameter) or while it is open (using the `ALTER SYSTEM` statement).

Thought password parameters reside in profiles, they are unaffected by `RESOURCE_LIMIT` or `ALTER SYSTEM` and password management is always enabled. In Oracle Database, Database Resource Manager primarily handles resource allocations and restrictions.

See Also:

- *Oracle Database Administrator's Guide* for detailed information on managing resources
- "Finding Information About Database Users and Profiles" on page 2-13 for viewing resource information
- *Oracle Database SQL Language Reference* for information about `ALTER SYSTEM` or `RESOURCE_LIMIT`

Creating Profiles

Any authorized database user can create, assign to users, alter, and drop a profile at any time (using the `CREATE USER` or `ALTER USER` statement). Profiles can be assigned only to users and not to roles or other profiles. Profile assignments do not affect current sessions, instead, they take effect only in subsequent sessions. To find information about current profiles, query the `DBA_PROFILES` view.

See Also:

- *Oracle Database SQL Language Reference* for more information about the SQL statements used for managing profiles, such as `CREATE PROFILE`, and for information about how to calculate composite limits.
- *Oracle Database Administrator's Guide* for detailed information about managing resources
- "Creating User Accounts" on page 2-1
- "Altering User Accounts" on page 2-6

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 use the modified pro file assignments.

Deleting User Accounts

When you drop a user account, Oracle Database removes the user account and associated schema from the data dictionary. It also immediately drops all schema objects contained in the user schema, if any.

Notes:

- If a user schema and associated objects must remain but the user must be denied access to the database, then revoke the CREATE SESSION privilege from the user.
- Do not attempt to drop the SYS or SYSTEM user. Doing so corrupts 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 sessions using the SQL statement ALTER SYSTEM with the KILL SESSION clause. You can find the session ID (SID) by querying the V$SESSION view.

Example 2–3 shows how to query V$SESSION and displays the session ID, serial number, and user name for user ANDY.

Example 2–3 Querying V$SESSION for the Session ID of a User

```
SELECT SID, serial#, username FROM V$SESSION;

    SID         SERIAL#     USERNAME
------- --------------- ----------------------
    127          55234      ANDY
...
```

Example 2–4 shows how to kill the session for user ANDY.

Example 2–4 Killing a User Session

```
ALTER SYSTEM KILL SESSION '127, 55234';
```

You can drop a user from a database using the DROP USER statement. To drop a user and all the user schema objects (if any), you must have the DROP USER system privilege. Because the DROP USER system privilege is powerful, a security administrator is typically the only type of user that has this privilege.

If the schema of the user contains any dependent schema objects, then 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 schema contains dependent objects, then an error message is returned and the user is not dropped.

Before dropping a user whose schema contains objects, thoroughly investigate which objects the schema contains and the implications of dropping them. You can find the objects owned by a particular user by querying the DBA_OBJECTS view.

Example 2–5 shows how to find the objects owned by user andy.

Example 2–5 Finding Objects Owned by a User

```
SELECT OWNER, OBJECT_NAME FROM DBA_OBJECTS WHERE OWNER LIKE 'ANDY';
```

(Enter the user name in capital letters.) Pay attention to any unknown cascading effects. For example, if you intend to drop a user who owns a table, then check whether any views or procedures depend on that particular table.

Example 2–6 drops the user andy and all associated objects and foreign keys that depend on the tables owned by andy.

Example 2–6 Dropping a User Account

```
DROP USER andy CASCADE;
```

> **See Also:** *Oracle Database Administrator's Guide* for more information about terminating sessions

Finding Information About Database Users and Profiles

This section describes views that you can use to find information about database users and profiles:

- Using Data Dictionary Views to Find Information About Users and Profiles
- Listing All Users and Associated Information
- Listing All Tablespace Quotas
- Listing All Profiles and Assigned Limits
- Viewing Memory Use for Each User Session

Using Data Dictionary Views to Find Information About Users and Profiles

Table 2–1 lists data dictionary views that contain information about database users and profiles. For detailed information on these views, see *Oracle Database Reference*.

Table 2–1 Data Dictionary Views That Contain User and Profile Information

View	Description
ALL_OBJECTS	Describes all objects accessible to the current user
ALL_USERS	Lists users visible to the current user, but does not describe them
DBA_PROFILES	Displays all profiles and their limits
DBA_TS_QUOTAS	Describes tablespace quotas for users
DBA_OBJECTS	Describes all objects in the database

Table 2–1 (Cont.) Data Dictionary Views That Contain User and Profile Information

View	Description
DBA_USERS	Describes all users of the database
DBA_USERS_WITH_DEFPWD	Lists all user accounts that have default passwords
PROXY_USERS	Describes users who can assume the identity of other users
RESOURCE_COST	Lists the cost for each resource in terms of CPUs for each session, reads for each session, connection times, and SGA
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
USER_TS_QUOTAS	Describes tablespace quotas for users
USER_OBJECTS	Describes all objects owned by the current user
USER_USERS	Describes only the current user
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

The following sections present examples of using these views. These examples assume that the following the following statements have been run:

```
CREATE PROFILE clerk LIMIT
    SESSIONS_PER_USER 1
    IDLE_TIME 30
    CONNECT_TIME 600;

CREATE USER jfee
    IDENTIFIED BY wildcat2564
    DEFAULT TABLESPACE users
    TEMPORARY TABLESPACE temp_ts
    QUOTA 500K ON users
    PROFILE clerk;

CREATE USER dcranney
    IDENTIFIED BY bedrock_3444
    DEFAULT TABLESPACE users
    TEMPORARY TABLESPACE temp_ts
    QUOTA unlimited ON users;

CREATE USER userscott
     IDENTIFIED BY tigris992;
```

Listing All Users and Associated Information

To find all users and their associated information as defined in the database, query the DBA_USERS view. For detailed information on the DBA_USERS view, see *Oracle Database Reference*.

For example:

```
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
```

Oracle Database encrypts all passwords to preserve security. If a user queries the PASSWORD column, then that user is not able to determine the password of another user. For example:

```
SELECT USERNAME, PASSWORD FROM DBA_USERS;

USERNAME        PASSWORD
--------------- ---------------
SYS             7C9BA35745000U8
SYSTEM          SKJ2O4IJI4HHN4O
USERSCOTT       AIUOHNU4I290AM3
JFEE            VXLJDOIU2343OWO
DCRANNEY        389NWKNKEO35NJO
```

Listing All Tablespace Quotas

Use the DBA_TS_QUOTAS view to list all tablespace quotas specifically assigned to each user. (For detailed information on this view, see *Oracle Database Reference*.) For example:

```
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. 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, then it is rounded up accordingly. Unlimited quotas are indicated by -1.

Listing All Profiles and Assigned Limits

The DBA_PROFILE view lists all profiles in the database and associated settings for each limit in each profile. (For detailed information on this view, see *Oracle Database Reference*.) For example:

```
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   10
DEFAULT             PASSWORD_LIFE_TIME         PASSWORD   180
DEFAULT             PASSWORD_REUSE_MAX         PASSWORD   UNLIMITED
DEFAULT             PASSWORD_LOCK_TIME         PASSWORD   1
DEFAULT             PASSWORD_GRACE_TIME        PASSWORD   7
DEFAULT             PASSWORD_VERIFY_FUNCTION   PASSWORD   UNLIMITED
DEFAULT             PASSWORD_REUSE_TIME        PASSWORD   UNLIMITED
32 rows selected.
```

Viewing Memory Use for Each User Session

To find the memory use for each user session, query the `V$SESSION` view. (For detailed information on this view, see *Oracle Database Reference*. The following query lists all current sessions, showing the Oracle Database user and current User Global Area (UGA) 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 allocated to each session since the instance started, replace `'session uga memory'` in the preceding query with `'session uga memory max'`.

3

Configuring 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.

This chapter discusses the following topics:

- About Authentication
- Configuring Password Protection
- Authenticating Database Administrators
- Using the Database to Authenticate Users
- Using the Operating System to Authenticate Users
- Using the Network to Authenticate Users
- Configuring Global User Authentication and Authorization
- Configuring an External Service to Authenticate Users and Passwords
- Using Multitier Authentication and Authorization
- Preserving User Identity in Multitiered Environments

About Authentication

Oracle Database can verify the identity of someone (a user, a device, or an entity) who wants to access data, resources, or applications. Validating this identity establishes a trust relationship for further interactions. Authentication 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. Authorization is described in Chapter 4, "Configuring Privilege and Role Authorization".

Configuring Password Protection

This section describes the following password-related topics:

- What Are the Oracle Database Built-in Password Protections?
- Using a Password Management Policy
- Configuring Password Settings in the Default Profile
- Managing the Secure External Password Store for Password Credentials

See also "Guidelines for Securing Passwords" on page 10-6 for advice on securing passwords. If you want to configure Oracle XML DB to authenticate users by encrypting their passwords but you do not need to encrypt other data (for example, an Intranet e-mail), see *Oracle XML DB Developer's Guide* for more information.

What Are the Oracle Database Built-in Password Protections?

Oracle Database provides a set of built-in password protections designed to protect your users' passwords. These password protections are as follows:

- **Password encryption.** Oracle Database automatically and transparently encrypts passwords during network (client-to-server and server-to-server) connections, using Advanced Encryption Standard (AES) before sending them across the network. For more information about encryption, see Chapter 9, "Developing Applications Using the Data Encryption API". See also *Oracle Database Advanced Security Administrator's Guide* for information about transparent data encryption.
- **Password complexity checking.** In a default installation, Oracle Database checks that new or changed passwords are sufficiently complex to prevent intruders who try to break into the system by guessing passwords. You can further customize the complexity of your users' passwords. See "Enforcing Password Complexity Verification" on page 3-7 for more information.
- **Preventing passwords from being broken.** If a user tries to log in to Oracle Database multiple times using an incorrect password, Oracle Database delays each login after the third try. This protection applies for attempts made from different IP addresses or multiple client connections. For the first three attempts, there is no delay. Afterwards, it gradually increases the time before the user can try another password, up to a maximum of about 10 seconds. If the user enters the correct password, he or she is able to log in successfully without any delay.

  This feature significantly decreases the number of passwords that an intruder would be able to try when attempting to log in. It is designed to prevent repeated attacks on password checking.
- **Enforced case sensitivity for passwords.** Passwords are case sensitive. For example, the password `hPP5620qr` fails if it is entered as `hpp5620QR` or `hPp5620Qr`. In previous releases, passwords were not case sensitive. See "Enabling or Disabling Password Case Sensitivity" on page 3-8 for information about how case sensitivity works, and how it affects password files and database links.
- **Passwords hashed using the Secure Hash Algorithm (SHA) cryptographic hash function SHA-1.** Oracle Database uses the SHA-1 verifier is to authenticate the user password and establish the session of the user. In addition, it enforces case sensitivity and restricts passwords to 160 bits. The advantage of using the SHA-1 verifier is that it is commonly used by Oracle Database customers and provides much better security without forcing a network upgrade. It also adheres to compliance regulations that mandate the use of strong passwords being protected by a suitably strong password hashing algorithm.

Using a Password Management Policy

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

- About Managing Passwords
- Finding User Accounts That Have Default Passwords
- Account Locking
- Password Aging and Expiration
- Controlling User Ability to Reuse Old Passwords
- Enforcing Password Complexity Verification
- Enabling or Disabling Password Case Sensitivity

See Also:

- "Managing Resources with Profiles" on page 2-11
- *Oracle Database SQL Language Reference* for syntax and specific information about SQL statements discussed in this section

About Managing Passwords

Database security systems that depend on passwords require that passwords be kept secret at all times. Because passwords are vulnerable to theft, forgery, and misuse, Oracle Database uses a password management policy. Database administrators and security officers control this policy through user profiles, enabling greater control of database security.

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.

Finding User Accounts That Have Default Passwords

When you create a database in Oracle Database 11*g* Release 1 (11.1), its default accounts are locked with the passwords expired. If you have upgraded from an earlier release of Oracle Database, you may have user accounts that have default passwords. These are default accounts that are created when you create a database, such as the `HR`, `OE`, and `SCOTT` accounts.

For greater security, change the passwords for these accounts. Using a default password that is commonly known can make your database vulnerable to attacks by intruders. To find both locked and unlocked accounts that use default passwords, log onto SQL*Plus using the `SYSDBA` privilege and then query the `DBA_USERS_WITH_DEFPWD` data dictionary view.

For example:

```
CONNECT / AS SYSDBA
Enter password: password
Connected.

SELECT * FROM DBA_USERS_WITH_DEFPWD;

USERNAME
---------
```

```
SCOTT
```

Then change the passwords for any accounts that the DBA_USERS_WITH_DEFPWD view lists:

```
ALTER USER scott IDENTIFIED BY password;
```

Replace *password* with a password that is secure. "How Oracle Database Checks the Complexity of Passwords" on page 3-7 describes the minimum requirements for passwords.

Account Locking

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

Example 3–1 sets the maximum number of failed login attempts for the user johndoe to 10 (the default), and the amount of time the account locked to 30 days. The account will unlock automatically after 30 days.

Example 3–1 Locking an Account with the CREATE PROFILE Statement

```
CREATE PROFILE prof LIMIT
 FAILED_LOGIN_ATTEMPTS 10
 PASSWORD_LOCK_TIME 30;
ALTER USER johndoe PROFILE prof;
```

Each time the user unsuccessfully logs in, Oracle Database increases the delay exponentially with each login failure.

If you do not specify a time interval for unlocking the account, then PASSWORD_LOCK_TIME assumes the value specified in a default profile. (The recommended value is 1 day.) If you specify PASSWORD_LOCK_TIME as UNLIMITED, then you must explicitly unlock the account by using an ALTER USER statement. For example, assuming that PASSWORD_LOCK_TIME UNLIMITED is specified for johndoe, then you use the following statement to unlock the johndoe account:

```
ALTER USER johndoe ACCOUNT UNLOCK;
```

After a user successfully logs into an account, Oracle Database resets the unsuccessful login attempt count for the user, if it exists, 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 explicitly lock or unlock user accounts. For example, the following statement locks the user account, susan:

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

Password Aging and Expiration

You can specify a password lifetime, after which the password expires and must be changed before logging into the account is permitted again. In addition, you can set a grace period, during which each attempt to log in to the database account receives a warning message to change the password. If the user does not change it by the end of that period, then Oracle Database expires the account. No further logins to that account are allowed without assistance by the database administrator.

You can also manually set the password state to expired, which sets the user account status to expired. The user or the database administrator must then change the password, using either the `PASSWORD` or `ALTER USER` statement, before the user can log in to the database.

Use the `CREATE PROFILE` or `ALTER 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.

Example 3–2 demonstrates how to create and assign a profile to user `johndoe`, and the `PASSWORD_LIFE_TIME` clause specifies that `johndoe` can use the same password for 180 days before it expires.

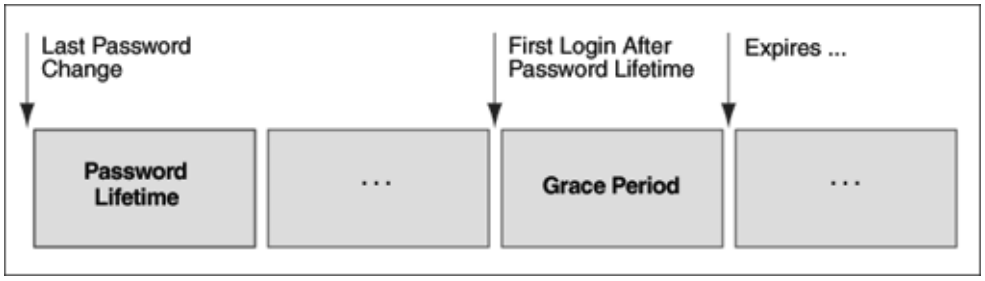
Example 3–2 Setting Password Aging and Expiration with the CREATE PROFILE Statement

```
CREATE PROFILE prof LIMIT
 FAILED_LOGIN_ATTEMPTS 4
 PASSWORD_LOCK_TIME 30
 PASSWORD_LIFE_TIME 180;
ALTER USER johndoe 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 they do not change the password within the grace period, then they are prompted for a new password each time they try to access their accounts. Access to an account is denied until a new password is supplied.

Figure 3–1 shows the chronology of the password lifetime and grace period.

Figure 3–1 Chronology of Password Lifetime and Grace Period



In the following example, the profile assigned to `johndoe` includes the specification of a grace period: `PASSWORD_GRACE_TIME = 3` (the recommended value). The first time `johndoe` tries to log in to the database after 90 days (this can be *any* day after the 90th day, that is, the 91st day, 100th day, or another day), he receives a warning message that his password will expire in 3 days. If 3 days pass, and if he does not change his password, then the password expires. After this, he receives a prompt to change his password on any attempt to log in, and cannot log in until he does so.

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

You can explicitly expire a password by using the CREATE USER and ALTER USER statements. 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 z8990sghj
 ...
 PASSWORD EXPIRE;
```

Controlling User Ability to Reuse Old Passwords

You can ensure that users do not reuse their old passwords for a specified amount of time or for a specified number of password changes. To do so, configure the rules for password reuse with CREATE PROFILE or ALTER PROFILE statements. For the complete syntax of these statements, see the *Oracle Database SQL Language Reference*.

Table 3–1 lists the CREATE PROFILE and ALTER PROFILE parameters that control ability of a user to reuse an old password.

Table 3–1 Parameters Controlling Reuse of an Old Password

Parameter Name	Description and Use
PASSWORD_REUSE_TIME	Requires either of the following: ■ A number specifying how many days (or a fraction of a day) between the earlier use of a password and its next use ■ The word UNLIMITED
PASSWORD_REUSE_MAX	Requires either of the following: ■ An integer to specify the number of password changes required before a password can be reused ■ The word UNLIMITED

If you do not specify a parameter, then the user can reuse passwords at any time, which is not a good security practice.

If neither parameter is UNLIMITED, then 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 that the profile of user A had PASSWORD_REUSE_MAX set to 10 and PASSWORD_REUSE_TIME set to 30. User A cannot reuse a password until he or she has reset the password 10 times, and until 30 days had passed since the password was last used.

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

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

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

Enforcing Password Complexity Verification

Password complexity refers to the creation of strong, secure passwords for database user accounts. You need to ensure that the passwords for your users are complex enough to provide reasonable protection against intruders who try to break into the system by guessing passwords.

How Oracle Database Checks the Complexity of Passwords

Oracle Database provides a sample password verification function in the PL/SQL script `UTLPWDMG.SQL` (located in *ORACLE_BASE*/*ORACLE_HOME*/`RDBMS/ADMIN`) that, when enabled, checks whether users are correctly creating or modifying their passwords. The `UTLPWDMG.SQL` script provides two password verification functions: one for previous releases of Oracle Database (which is commented out) and an updated version for Oracle Database Release 11*g*.

The `UTLPWDMG.SQL` script checks for the following when users create or modify passwords:

- The password contains no fewer than eight characters.
- The password is not the same as the user name, nor is it the user name spelled backward or with numeric characters appended.
- The password is not the same as the server name or the server name with the numbers 1–100 appended.
- The password is not too simple, for example, `welcome1`, `database1`, `account1`, `user1234`, `password1`, `oracle`, `oracle123`, `computer1`, `abcdefg1`, or `change_on_install`.
- The password includes at least 1 numeric and 1 alphabetic character.
- The password differs from the previous password by at least 3 letters.

> **See Also:** "Guidelines for Securing Passwords" on page 10-6 for guidelines on how you can further secure passwords

Customizing Password Complexity Verification

You can create your own password complexity verification function by backing up and customizing the `PASSWORD_VERIFY` function in the `UTLPWDMG.SQL` script. In fact, Oracle recommends that you do so to further secure your site's passwords. See also Guideline 1 in "Guidelines for Securing Passwords" on page 10-6 for general advice on creating passwords.

By default, password complexity verification is not enabled. To enable the password complexity verification:

1. Log in to SQL*Plus with administrative privileges and then run the `UTLPWDMG.SQL` script (or your modified version of this script) to create the password complexity function in the `SYS` schema.

   ```
   CONNECT SYS/AS SYSDBA
   Enter password: password
   Connected.

   @$ORACLE_HOME/RDBMS/ADMIN/utlpwdmg.sql
   ```

2. In the default profile or the user profile, set the `PASSWORD_VERIFY_FUNCTION` setting to either the sample password complexity function in the `UTLPWDMG.SQL` script, or to your customized function. Use one of the following methods:

- Log in to SQL*Plus with administrator privileges and use the CREATE PROFILE or ALTER PROFILE statement to enable the function. For example, to update the default profile to use the verify_function_11G function:

```
ALTER PROFILE default
 PASSWORD_VERIFY_FUNCTION verify_function_11G;
```

- In Oracle Enterprise Manager, go to the Edit Profiles page and then under Complexity, select the name of the password complexity function from the Complexity function list.

After you have enabled password complexity verification, it takes effect immediately.

Note: The ALTER USER statement has a REPLACE clause. With this clause, users can change their own unexpired passwords by supplying the old password to authenticate themselves.

If the password has expired, then 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 database administrator with ALTER ANY USER privilege can change any user password (force a new password) without supplying the old one.

Enabling or Disabling Password Case Sensitivity

When you create or modify user accounts, by default, passwords are case sensitive. To control the use of case sensitivity in passwords, set the SEC_CASE_SENSITIVE_LOGON initialization parameter. Only users who have the ALTER SYSTEM privilege can set the SEC_CASE_SENSITIVE_LOGIN parameter. Set it to TRUE to enable case sensitivity or FALSE to disable case sensitivity.

For greater security, Oracle recommends that you enable case sensitivity in passwords. However, if you have compatibility issues with your applications, you can use this parameter to disable password case sensitivity. Examples of application compatibility issues are passwords for your applications being hard-coded to be case insensitive, or different application modules being inconsistent about case sensitivity when sending credentials to start a database session.

Example 3–3 shows how to enable case sensitivity in passwords.

Example 3–3 Enabling Case Sensitivity in Passwords

```
CONNECT SYS/AS SYSDBA
Enter password: password
Connected.

ALTER SYSTEM SET SEC_CASE_SENSITIVE_LOGON = TRUE
```

In previous releases of Oracle Database, passwords were not case sensitive. If you import user accounts from a previous release, for example, Release 10*g*, into the current database release, the case-insensitive passwords in these accounts remain case insensitive until the user changes his or her password. If the account was granted SYSDBA or SYSOPER privileges, it is imported to the password file. (See "How Case Sensitivity Affects Password Files" on page 3-9 for more information.) When a password from a user account from the previous release is changed, it then becomes case sensitive.

You can find users who have case sensitive or case insensitive passwords by querying the DBA_USERS view. The PASSWORD_VERSIONS column in this view indicates the release in which the password was created. For example:

```
SELECT USERNAME,PASSWORD_VERSIONS FROM DBA_USERS;

USERNAME                       PASSWORD_VERSIONS
------------------------------ -----------------
JONES                          10G 11G
ADAMS                          10G 11G
CLARK                          10G 11G
PRESTON                        11G
BLAKE                          10G
```

The passwords for accounts jones, adams, and clark were originally created in Release 10*g* and then reset in Release 11*g*. Their passwords, assuming case sensitivity has been enabled, are now case sensitive, as is the password for preston. However, the account for blake is still using the Release 10*g* standard, so it is case insensitive. Ask him to reset his password so that it will be case sensitive, and therefore more secure.

See *Oracle Database Reference* for more information about the DBA_USERS view.

How Case Sensitivity Affects Password Files

You can enable or disable case sensitivity for password files by using the ignorecase argument in the ORAPWD command line utility. The default value for ignorecase is n (no), which enforces case sensitivity.

Example 3–4 shows how to enable case sensitivity in password files.

Example 3–4 Enabling Password Case Sensitivity

```
orapwd file=orapw password=manag23?er entries=100 ignorecase=n
```

This creates a password file called orapw, whose password for SYS is manag23?er and will be case sensitive. Afterwards, this connection succeeds:

```
CONNECT SYS/AS SYSDBA
Enter password: manag23?er
```

> **Note:** Passwords do not display at the Enter password prompt, but are shown here to illustrate the case sensitivity of the password.

But the following connection, which uses incorrect case for the password, fails:

```
CONNECT SYS/AS SYSDBA
Enter password: Manag23?er
```

If you set ignorecase to y, the passwords in the password file are case insensitive. In other words, entering the password manag23?er as manag23?er, Manag23?er, or MANAG23?ER succeeds.

If you imported user accounts from a previous release and these accounts were created with SYSDBA or SYSOPER privileges, they will be included in the password file. The passwords for these accounts are case insensitive. The next time these users change their passwords, and assuming case sensitivity is enabled, the passwords become case sensitive. For greater security, have these users change their passwords.

See *Oracle Database Administrator's Guide* for more information about password files.

How Case Sensitivity Affects Accounts Created for Database Link Connections

When you create a database link connection, you need to define a user name and password for the connection. When you create the database link connection, the password is case sensitive. How this user enters his or her password for connections depends on the release in which the database link was created:

- Before a user can connect from a pre-Release 11*g* database to a Release 11*g* database, and assuming that case sensitivity is enabled, you must re-create the password for this database link using all uppercase letters, for example, `SEBASTIAN2GO`.

  The reason you need to re-create the password using all uppercase letters is so that it will match how Oracle Database stores database link passwords. Oracle Database always stores this type of password in uppercase letters, even if the password had originally been created using lower or mixed case letters (for example, `sebastian2go` or `Sebastian2Go`). If case sensitivity is disabled, the user can enter the password using the case the password was created in, for example, `Sebastian2Go` if it was created as `Sebastian2Go`.

- If the user is connecting from a Release 11*g* database to another Release 11*g* database, he or she must enter the password using the case in which it was created, assuming that case sensitivity is enabled.

- If the user connecting from a Release 11*g* database to a pre-Release 11*g* database, he or she can enter his or her password using any case, because the password is still case insensitive.

In other words, any time a user connects to a Release 11*g* database from a database link, he or she needs to enter the password in its exact case.

You can find the user accounts for existing database links by running the `V$DBLINK` view. For example:

```
SELECT DB_LINK, OWNER_ID FROM V$DBLINK;
```

See *Oracle Database Reference* for more information about the `V$DBLINK` view.

Configuring Password Settings in the Default Profile

A profile is a collection of parameters that sets limits on database resources. If you assign the profile to a user, then that user cannot exceed these limits. You can use profiles to configure database settings such as sessions per user, logging and tracing features, and so on. Profiles can also control user passwords.

Table 3–2 lists the password-specific parameter settings in the default profile.

Table 3–2 Password-Specific Settings in the Default Profile

Parameter	Default Setting	Description
`FAILED_LOGIN_ATTEMPTS`	`10`	Sets the maximum times a user try to log in and to fail before locking the account. **Note:** You also can set limits on the number of times an unauthorized user (possibly an intruder) attempts to log in to Oracle Call Interface applications by using the `SEC_MAX_FAILED_LOGIN_ATTEMPTS` initialization parameter. See "Configuring the Maximum Number of Authentication Attempts" on page 5-13 for more information.

Table 3–2 (Cont.) Password-Specific Settings in the Default Profile

Parameter	Default Setting	Description
PASSWORD_GRACE_TIME	7	Sets the number of days that a user has to change his or her password before it expires.
PASSWORD_LIFE_TIME	180	Sets the number of days the user can use his or her current password.
PASSWORD_LOCK_TIME	1	Sets the number of days an account will be locked after the specified number of consecutive failed login attempts.
PASSWORD_REUSE_MAX	UNLIMITED	Sets the number of days before which a password cannot be reused.
PASSWORD_REUSE_TIME	UNLIMITED	Sets the number of password changes required before the current password can be reused.

For greater security, use the default settings described in Table 3–2, based on your needs. You can create or modify the password settings in the profile by using one of the following methods:

- **Database Configuration Assistant (DBCA).** When you create a new database or modify an existing database, you can use the Security Settings window to enable or disable its default security settings. The password-specific settings in Table 3–2 are part of these default settings. The default security settings also include the auditing settings described in "Using Default Auditing for Security-Relevant SQL Statements and Privileges" on page 6-10. Oracle recommends that you enable the default security settings.
- **CREATE PROFILE or ALTER PROFILE statement.** You can create or modify the password-specific parameters individually by using the `CREATE PROFILE` or `ALTER PROFILE` statement. For example:

```
ALTER PROFILE prof
 FAILED_LOGIN_ATTEMPTS 10
 PASSWORD_LOCK_TIME 1;
```

  See *Oracle Database SQL Language Reference* for more information about `CREATE PROFILE`, `ALTER PROFILE`, and the password-related parameters described in this section.

Managing the Secure External Password Store for Password Credentials

This section describes how to use the secure external password store to manage password credentials.

- About the Secure External Password Store
- How Does the External Password Store Work?
- Configuring Clients to Use the External Password Store
- Managing External Password Store Credentials

About the Secure External Password Store

You can store password credentials for connecting to databases by using a client-side Oracle wallet. An Oracle wallet is a secure software container that stores authentication and signing credentials.

This wallet usage can simplify large-scale deployments that rely on password credentials for connecting to databases. When this feature is configured, application code, batch jobs, and scripts no longer need embedded user names and passwords. This reduces risk because the passwords are no longer exposed, and password management policies are more easily enforced without changing application code whenever user names or passwords change.

> **See Also:** *Oracle Database Advanced Security Administrator's Guide* for general information about Oracle wallets

---

Note: The external password store of the wallet is separate from the area where public key infrastructure (PKI) credentials are stored. Consequently, you cannot use Oracle Wallet Manager to manage credentials in the external password store of the wallet. Instead, the command-line utility, `mkstore`, is provided to manage these credentials.

How Does the External Password Store Work?

Typically, users (and as applications, batch jobs, and scripts) connect to databases by using a standard `CONNECT` statement that specifies a database connection string. This string can include a user name and password, and an Oracle Net service name identifying the database on an Oracle Database network. For example, the service name could be the URL that identifies that database, or a TNS alias you entered in the `tnsnames.ora` file in the database. Another possibility is a *`host:port:sid`* string.

The following examples are standard `CONNECT` statements that could be used for a client that is not configured to use the external password store:

```
CONNECT SALESAPP/2Ip6Cg8@sales_db.us.acme.com

CONNECT SALESAPP/2Ip6Cg8@ORASALES

CONNECT SALESAPP/2Ip6Cg8@ourhost37:1527:DB17
```

In these examples, `salesapp` is the user name and `2Ip6Cg8` is the password, with the unique connect string for the database shown as specified in three different ways. You could use its URL `sales_db.us.acme.com`, or its TNS alias `ORASALES` from the `tnsnames.ora` file, or its *`host:port:sid`* string.

However, when clients are configured to use the secure external password store, applications can connect to a database with the following `CONNECT` statement syntax, without specifying database login credentials:

```
CONNECT /@db_connect_string
```

In this specification, *`db_connect_string`* is a valid connection string to access the intended database, such as the service name, URL, or alias as shown in the earlier examples.

In this case, the database credentials, user name and password, are securely stored in an Oracle wallet created for this purpose. The autologin feature of this wallet is turned on, so the system does not need a password to open the wallet. From the wallet, it gets the credentials to access the database for the user they represent.

> **See Also:** *Oracle Database Advanced Security Administrator's Guide* for information about autologin wallets

Configuring Clients to Use the External Password Store

If your client is already configured to use external authentication, such as Windows native authentication or Secure Sockets Layer (SSL), then Oracle Database uses that authentication method. The same credentials used for such authentication are typically also used to log in to the database.

For clients not using such authentication methods or wanting to override them for database authentication, you can set the `SQLNET.WALLET_OVERRIDE` parameter in `sqlnet.ora` to `TRUE`. The default value for `SQLNET.WALLET_OVERRIDE` is `FALSE`, allowing standard use of authentication credentials as before.

If you want a client to use the secure external password store feature, then perform the following configuration tasks.

To enable clients to use the external password store:

1. Create a wallet on the client by using the following syntax at the command line:

   ```
   mkstore -wrl wallet_location -create
   ```

   For example:

   ```
   mkstore -wrl c:\oracle\product\11.1.0\db_1\wallets -create
   ```

   wallet_location is the path to the directory where you want to create and store the wallet. This command creates an Oracle wallet with the autologin feature enabled at the location you specify. The autologin feature enables the client to access the wallet contents without supplying a password. See *Oracle Database Advanced Security Administrator's Guide* for information about autologin wallets.

2. Create database connection credentials in the wallet by using the following syntax at the command line:

   For example:

   ```
   mkstore -wrl c:\oracle\product\11.1.0\db_1\wallets -createCredential orcl
   system
   Enter passwword: password
   ```

   In this specification:

   - *wallet_location* is the path to the directory where you created the wallet in Step 1.
   - *db_connect_string* is the TNS alias you use to specify the database in the `tnsnames.ora` file or any service name you use to identify the database on an Oracle network. By default, `tnsnames.ora` is located in the `$ORACLE_HOME/network/admin` directory on UNIX systems and in *ORACLE_HOME*`\network\admin` on Windows.
   - *username* is the database login credential. When prompted, enter the password for this user.

   Repeat this step for each database you want accessible using the `CONNECT /@`*db_connect_string* syntax.

   Note: The *db_connect_string* used in the `CONNECT /@`*db_connect_string* statement must be identical to the *db_connect_string* specified in the `-createCredential` command.

3. In the client `sqlnet.ora` file, enter the `WALLET_LOCATION` parameter and set it to the directory location of the wallet you created in Step 1.

   For example, if you created the wallet in `$ORACLE_HOME/network/admin` and your Oracle home is set to `/private/ora11`, then you need to enter the following into your client `sqlnet.ora` file:

   ```
   WALLET_LOCATION =
     (SOURCE =
       (METHOD = FILE)
       (METHOD_DATA =
     (DIRECTORY = /private/ora11/network/admin)
     )
    )
   ```

4. In the client `sqlnet.ora` file, enter the `SQLNET.WALLET_OVERRIDE` parameter and set it to `TRUE` as follows:

   ```
   SQLNET.WALLET_OVERRIDE = TRUE
   ```

   This setting causes all `CONNECT /@db_connect_string` statements to use the information in the wallet at the specified location to authenticate to databases.

   When external authentication is in use, an authenticated user with such a wallet can use the `CONNECT /@db_connect_string` syntax to access the previously specified databases without providing a user name and password. However, if a user fails that external authentication, then these connect statements also fail.

   Note: If an application uses SSL for encryption, then the `sqlnet.ora` parameter, `SQLNET.AUTHENTICATION_SERVICES`, specifies SSL and an SSL wallet is created. If this application wants to use secret store credentials to authenticate to databases (instead of the SSL certificate), then those credentials must be stored in the SSL wallet. After SSL authentication, if `SQLNET.WALLET_OVERRIDE = TRUE`, then the user names and passwords from the wallet are used to authenticate to databases. If `SQLNET.WALLET_OVERRIDE = FALSE`, then the SSL certificate is used.

Example 3–5 shows a sample `sqlnet.ora` file with the `WALLET_LOCATION` and the `SQLNET.WALLET_OVERRIDE` parameters set as described in Steps 3 and 4.

Example 3–5 Sample SQLNET.ORA File with Wallet Parameters Set

```
WALLET_LOCATION =
   (SOURCE =
     (METHOD = FILE)
     (METHOD_DATA =
       (DIRECTORY = /private/ora11/network/admin)
     )
    )

SQLNET.WALLET_OVERRIDE = TRUE
SSL_CLIENT_AUTHENTICATION = FALSE
SSL_VERSION = 0
```

Managing External Password Store Credentials

This section summarizes the following tasks you can perform to manage credentials in the external password store by using the `mkstore` command-line utility:

- Listing External Password Store Contents
- Adding Credentials to an External Password Store
- Modifying Credentials in an External Password Store
- Deleting Credentials from an External Password Store

Listing External Password Store Contents Periodically, you may want to view all contents of a client wallet external password store, or you may need to check specific credentials by viewing them. Listing the external password store contents provides information you can use to decide whether to add or delete credentials from the store.

To list the contents of the external password store, enter the following command at the command line:

```
mkstore -wrl wallet_location -listCredential
```

For example:

```
mkstore -wrl c:\oracle\product\11.1.0\db_1\wallets -listCredential
```

wallet_location specifies the path to the directory where the wallet, whose external password store contents you want to view, is located. This command lists all of the credential database service names (aliases) and the corresponding user name (schema) for that database. Passwords are not listed.

Adding Credentials to an External Password Store You can store multiple credentials in one client wallet. For example, if a client batch job connects to `hr_database` and a script connects to `sales_database`, then you can store the login credentials in the same client wallet. You cannot, however, store multiple credentials (for logging in to multiple schemas) for the same database in the same wallet. If you have multiple login credentials for the same database, then they must be stored in separate wallets.

To add database login credentials to an existing client wallet, enter the following command at the command line:

```
mkstore -wrl wallet_location -createCredential db_alias username
```

For example:

```
mkstore -wrl c:\oracle\product\11.1.0\db_1\wallets -createCredential orcl system
Enter password: password
```

In this specification:

- *wallet_location* is the path to the directory where the client wallet to which you want to add credentials is stored.
- *db_alias* can be the TNS alias you use to specify the database in the `tnsnames.ora` file or any service name you use to identify the database on an Oracle network.
- *username* is the database login credential for the schema to which your application connects. When prompted, enter the password for this user.

Modifying Credentials in an External Password Store If the database connection strings change, then you can modify the database login credentials that are stored in the wallet.

To modify database login credentials in a wallet, enter the following command at the command line:

```
mkstore -wrl wallet_location -modifyCredential dbase_alias username
```

For example:

```
mkstore -wrl c:\oracle\product\11.1.0\db_1\wallets -modifyCredential sales_db
Enter password: password
```

In this specification:

- *wallet_location* is the path to the directory where the wallet is located.
- *db_alias* is a new or different alias you want to use to identify the database. It can be a TNS alias you use to specify the database in the `tnsnames.ora` file or any service name you use to identify the database on an Oracle network.
- *username* is the new or different database login credential. When prompted, enter the password for this user.

Deleting Credentials from an External Password Store If a database no longer exists or if you want to disable connections to a specific database, then you can delete all login credentials for that database from the wallet.

To delete database login credentials from a wallet, enter the following command at the command line:

```
mkstore -wrl wallet_location -deleteCredential db_alias
```

For example:

```
mkstore -wrl c:\oracle\product\11.1.0\db_1\wallets -deleteCredential orcl
```

In this specification:

- *wallet_location* is the path to the directory where the wallet is located.
- *db_alias* is the TNS alias you use to specify the database in the `tnsnames.ora` file, or any service name you use to identify the database on an Oracle Database network.

Authenticating Database Administrators

Database administrators perform special operations, such as shutting down or starting up a database, that should not be performed by non-administrative database users. Oracle Database provides the following methods to secure the authentication of database administrators who have either `SYSDBA` or `SYSOPER` privileges:

- Strong Authentication and Centralized Management for Database Administrators
- Authenticating Database Administrators by Using the Operating System
- Authenticating Database Administrators by Using Their Passwords

Strong Authentication and Centralized Management for Database Administrators

You can centrally control `SYSDBA` and `SYSOPER` access to multiple databases. Consider using this type of authentication for database administration for the following situations:

- You have concerns about password file vulnerability.
- Your site has very strict security requirements.

- You want to separate the identity management from your database. By using a directory server such as Oracle Internet Directory (OID), for example, you can maintain, secure, and administer that server separately.

To enable the Oracle Internet Directory server to authorize SYSDBA and SYSOPER connections, use one of the following methods, depending on your environment:

- Configuring Directory Authentication for Administrative Users
- Configuring Kerberos Authentication for Administrative Users
- Configuring Secure Sockets Layer Authentication for Administrative Users

Configuring Directory Authentication for Administrative Users

To configure directory authentication for administrative users:

1. Configure the administrative user by using the same procedures you would use to configure a typical user.
2. In Oracle Internet Directory, grant the SYSDBA or SYSOPER privilege to the user for the database that this user will administer.

   Grant SYSDBA or SYSOPER only to trusted users. See "Guidelines for Securing User Accounts and Privileges" on page 10-2 for advice on this topic.
3. Set the LDAP_DIRECTORY_SYSAUTH initialization parameter to YES:

   ```
   ALTER SYSTEM SET LDAP_DIRECTORY_SYSAUTH = YES;
   ```

   When set to YES, the LDAP_DIRECTORY_SYSAUTH parameter enables SYSDBA and SYSOPER users to authenticate to the database by using a strong authentication method.

   See *Oracle Database Reference* for more information about LDAP_DIRECTORY_SYSAUTH.
4. Set the LDAP_DIRECTORY_ACCESS parameter to either PASSWORD or SSL. For example:

   ```
   ALTER SYSTEM SET LDAP_DIRECTORY_ACCESS = PASSWORD;
   ```

   Ensure that the LDAP_DIRECTORY_ACCESS initialization parameter is not set to NONE. Setting this parameter to PASSWORD or SSL ensures that users can be authenticated using SYSDBA or SYSOPER through Oracle Internet Directory. See *Oracle Database Reference* for more information about LDAP_DIRECTORY_ACCESS.

Afterward, this user can log in by including the net service name in the CONNECT statement in SQL*Plus. For example, to log on as SYSDBA if the net service name is orcl:

```
CONNECT user/password@orcl AS SYSDBA
```

If the database is configured to use a password file for remote authentication, Oracle Database checks the password file first.

Configuring Kerberos Authentication for Administrative Users

To configure Kerberos authentication for administrative users:

1. Configure the administrative user by using the same procedures you would use to configure a typical user.

   See *Oracle Database Advanced Security Administrator's Guide* for more information.

2. Configure Oracle Internet Directory for Kerberos authentication.

   See *Oracle Database Enterprise User Security Administrator's Guide* for more information.

3. In Oracle Internet Directory, grant the SYSDBA or SYSOPER privilege to the user for the database that this user will administer.

   Grant SYSDBA or SYSOPER only to trusted users. See "Guidelines for Securing User Accounts and Privileges" on page 10-2 for advice on this topic.

4. Set the LDAP_DIRECTORY_SYSAUTH initialization parameter to YES:

   ```
   ALTER SYSTEM SET LDAP_DIRECTORY_SYSAUTH = YES;
   ```

   When set to YES, the LDAP_DIRECTORY_SYSAUTH parameter enables SYSDBA and SYSOPER users to authenticate to the database by using strong authentication methods. See *Oracle Database Reference* for more information about LDAP_DIRECTORY_SYSAUTH.

5. Set the LDAP_DIRECTORY_ACCESS parameter to either PASSWORD or SSL. For example:

   ```
   ALTER SYSTEM SET LDAP_DIRECTORY_ACCESS = SSL;
   ```

   Ensure that the LDAP_DIRECTORY_ACCESS initialization parameter is not set to NONE. Setting this parameter to PASSWORD or SSL ensures that users can be authenticated using SYSDBA or SYSOPER through Oracle Internet Directory. See *Oracle Database Reference* for more information about LDAP_DIRECTORY_ACCESS.

Afterward, this user can log in by including the net service name in the CONNECT statement in SQL*Plus. For example, to log on as SYSDBA if the net service name is orcl:

```
CONNECT /@orcl AS SYSDBA
```

Configuring Secure Sockets Layer Authentication for Administrative Users

To configure Secure Sockets Layer (SSL) authentication for administrative users:

1. Configure the client to use SSL:
   a. Configure the client wallet and user certificate. Update the wallet location in the sqlnet.ora configuration file.

      You can use Wallet Manager to configure the client wallet and user certificate. See *Oracle Database Advanced Security Administrator's Guide* for more information.

   b. Configure the Oracle net service name to include server DNs and use TCP/IP with SSL in tnsnames.ora.
   c. Configure TCP/IP with SSL in listener.ora.
   d. Set the client SSL cipher suites and the required SSL version, and then set SSL as an authentication service in sqlnet.ora.
2. Configure the server to use SSL:
   a. Enable SSL for your database listener on TCPS and provide a corresponding TNS name. You can use Net Configuration Assistant to configure the TNS name.
   b. Store the database PKI credentials in the database wallet. You can use Wallet Manager do this.

c. Set the LDAP_DIRECTORY_ACCESS initialization parameter to SSL:

```
ALTER SYSTEM SET LDAP_DIRECTORY_ACCESS = SSL;
```

See *Oracle Database Reference* for more information about LDAP_DIRECTORY_ACCESS.

3. Configure Oracle Internet Directory for SSL user authentications.

See *Oracle Database Enterprise User Security Administrator's Guide* for information on configuring enterprise user security SSL authentication.

4. In Oracle Internet Directory, grant the SYSDBA or SYSOPER privilege to the user for the database that the user will administer.

5. On the server computer, set the LDAP_DIRECTORY_SYSAUTH initialization parameter to YES.

```
ALTER SYSTEM SET LDAP_DIRECTORY_SYSAUTH = YES;
```

When set to YES, the LDAP_DIRECTORY_SYSAUTH parameter enables SYSDBA and SYSOPER users to authenticate to the database by using a strong authentication method. See *Oracle Database Reference* for more information about LDAP_DIRECTORY_SYSAUTH.

Afterward, this user can log in by including the net service name in the CONNECT statement in SQL*Plus. For example, to log on as SYSDBA if the net service name is orcl:

```
CONNECT /@orcl AS SYSDBA
```

Authenticating Database Administrators by Using the Operating System

Operating system authentication for a database administrator typically involves establishing a group on the operating system, granting DBA privileges to that group, and then adding the names of persons who should have those privileges to that group.

Note: On UNIX systems, the special group is called the **dba** group.

See Also: Your Oracle Database operating system-specific documentation for information about configuring operating system authentication of database administrators

Authenticating Database Administrators by Using Their Passwords

Oracle Database uses database-specific password files to keep track of database user names that have been granted the SYSDBA and SYSOPER privileges. These privileges enable the following activities:

- The SYSOPER system privilege lets database administrators perform STARTUP, SHUTDOWN, ALTER DATABASE OPEN/MOUNT, ALTER DATABASE BACKUP, ARCHIVE LOG, and RECOVER operations. SYSOPER also includes the RESTRICTED SESSION privilege.
- The SYSDBA system privilege has all system privileges with ADMIN OPTION, including the SYSOPER system privilege, and permits CREATE DATABASE and time-based recovery.
- A password file containing users with SYSDBA or SYSOPER privileges can be shared between different databases. You can have a shared password file that

contains users in addition to the SYS user. To share a password file among different databases, set the REMOTE_LOGIN_PASSWORDFILE parameter in the init.ora file to SHARED.

- Password file-based authentication is enabled by default. This means that the database is ready to use a password file for authenticating users that have SYSDBA or SYSOPER system privileges. Password file based authentication is activated as soon as you create a password file using the ORAPWD utility.

  Anyone who has EXECUTE privileges and write privileges to the $ORACLE_HOME/dbs directory can run the ORAPWD utility.

However, be aware that using password files may pose security risks. For this reason, consider using the authentication methods described in "Strong Authentication and Centralized Management for Database Administrators" on page 3-16. Examples of password security risks are as follows:

- An intruder could steal or attack the password file.
- Many users do not change the default password.
- The password could be easily guessed.
- The password is vulnerable if it can be found in a dictionary.
- Passwords that are too short, chosen perhaps for ease of typing, are vulnerable if an intruder obtains the cryptographic hash of the password.

Note: Connections requested AS SYSDBA or AS SYSOPER must use these phrases; without them, the connection fails. The Oracle Database parameter O7_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: *Oracle Database Administrator's Guide* for information about creating and maintaining password files

Using the Database to Authenticate Users

This section explains how you can use the database itself to authenticate users. It explains the following topics:

- About Database Authentication
- Advantages of Database Authentication
- Creating a User Who Is Authenticated by the Database

About Database Authentication

Oracle Database can authenticate users attempting to connect to a database by using information stored in that database itself. To configure Oracle Database to use database authentication, you must create each user with an associated password. User names can be multibyte, but each password must be composed of single-byte characters, even if your database uses a multibyte character set. The user must provide this user name and password when attempting to establish a connection. Oracle Database stores user passwords in the data dictionary in an encrypted format.

To identify the authentication protocols that are allowed by a client or a database, a DBA can explicitly set the SQLNET.ALLOWED_LOGON_VERSION parameter in the server sqlnet.ora file. 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 value is 8. These values represent database server versions. Oracle recommends the value 10.

To enhance security when using database authentication, Oracle recommends that you use password management, including account locking, password aging and expiration, password history, and password complexity verification. See "Using a Password Management Policy" on page 3-3 for more information about password management.

Advantages of Database Authentication

The advantages of database authentication are as follows:

- User accounts and all authentication are controlled by the database. There is no reliance on anything outside of the database.
- Oracle Database provides strong password management features to enhance security when using database authentication.
- It is easier to administer when there are small user communities.

Creating a User Who Is Authenticated by the Database

The following SQL statement creates a user who is identified and authenticated by Oracle Database. User sebastian must specify the password p34stoN whenever he connects to Oracle Database.

```
CREATE USER sebastian IDENTIFIED BY p34stoN;
```

Using the Operating System to Authenticate Users

Some operating systems permit Oracle Database to use information they maintain to authenticate users. This has the following benefits:

- Once authenticated by the operating system, users can connect to Oracle Database more conveniently, without specifying a user name or password. For example, an operating system-authenticated user can invoke SQL*Plus and omit the user name and password prompts by entering the following command at the command line:

  ```
  SQLPLUS /
  ```

  Within SQL*Plus, you enter:

  ```
  CONNECT /
  ```

- With control over user authentication centralized in the operating system, Oracle Database need not store or manage user passwords, although it still maintains user names in the database.
- Audit trails in the database and operating system can use the same user names.
- You can authenticate both operating system and non-operating system users in the same system. For example:
  - **Authenticate users by the operating system.** You create the user account using the IDENTIFIED EXTERNALLY clause of the CREATE USER statement, and then you set the OS_AUTHENT_PREFIX initialization parameter to specify

a prefix that Oracle Database uses to authenticate users attempting to connect to the server.

- **Authenticate non-operating system users.** These are users who are assigned passwords and authenticated by the database.
- **Authenticate Oracle Database Enterprise User Security users.** These user accounts where created using the `IDENTIFIED GLOBALLY` clause of the `CREATE USER` statement, and then authenticated by Oracle Internet Directory (OID) currently in the same database.

However, you should be aware of the following drawbacks to using the operating system to authenticate users:

- A user must have an operating system account on the computer that needs to be accessed. Not all users have operating system accounts, particularly non-administrative users.
- If a user has logged in using this method and steps away from the terminal, another user could easily log in because this user does not need any passwords or credentials. This could pose a serious security problem.
- 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* for more information about authentication, operating systems, distributed database concepts, and distributed data management
- Operating system-specific documentation by Oracle Database for more information about authenticating by using your operating system

Using the Network to Authenticate Users

You can authenticate users over a network by using Secure Sockets Layer with third-party services.

- Authentication Using Secure Sockets Layer
- Authentication Using Third-Party Services

Authentication Using Secure Sockets Layer

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

See *Oracle Database Advanced Security Administrator's Guide* for instructions about configuring SSL.

Authentication Using Third-Party Services

You need to use third-party network authentication services if you want to authenticate Oracle Database users over a network. Prominent examples include Kerberos, PKI (public key infrastructure), the RADIUS (Remote Authentication Dial-In User Service), and directory-based services, as described in the following sections.

If network authentication services are available to you, then Oracle Database can accept authentication from the network service. If you use a network authentication service, then some special considerations arise for network roles and database links.

Note: To use a network authentication service with Oracle Database, you need Oracle Database Enterprise Edition with the Oracle Database Advanced Security option.

See Also: *Oracle Database Advanced Security Administrator's Guide* for information about Oracle Enterprise Edition with the Oracle Database Advanced Security option

Authenticating Using Kerberos

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 Database Advanced Security Administrator's Guide* for more information about Kerberos

Authenticating Using RADIUS

Oracle Database 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. For information about how to configure RADIUS, see *Oracle Database Advanced Security Administrator's Guide*.

Authenticating Using Directory-Based Services

Using a central directory can make authentication and its administration efficient. Directory-based services include the following:

- **Oracle Internet Directory**, which uses the Lightweight Directory Access Protocol (LDAP), uses a central repository to store and manage information about users (called enterprise users) whose accounts were created in a distributed environment. Although database users must be created (with passwords) in each database that they need to access, enterprise user information is accessible centrally in the Oracle Internet Directory. You can also integrate this directory with Microsoft Active Directory and SunOne.

  For more information about Oracle Internet Directory, see *Oracle Internet Directory Administrator's Guide*.

- **Oracle Enterprise Security Manager** lets you store and retrieve roles from Oracle Internet Directory, which provides centralized privilege management to make administration easier and increase security levels. For more information about Oracle Enterprise Security Manager, see *Oracle Enterprise Manager Advanced Configuration*.

Authenticating Using Public Key Infrastructure

Authentication systems based on public key infrastructure (PKI) issue digital certificates to user clients, which use them to authenticate directly to servers in the enterprise without directly involving an authentication server. Oracle Database

provides a PKI for using public keys and certificates, consisting of the following components:

- **Authentication and secure session key management using SSL**

  See "Authentication Using Secure Sockets Layer" on page 3-22 for more information.

- **Trusted certificates**

  These are used to identify third-party entities that are trusted as signers of user certificates when an identity is being validated. When the user certificate is being validated, the signer is checked by 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 this chain, then a trusted certificate at a lower level is simply trusted without needing to have all its higher-level certificates reverified. For more information about trusted certificates, see *Oracle Database Advanced Security Administrator's Guide*.

- **OracleAS Certificate Authority**

  This is a component of the Oracle Identity Management infrastructure, which provides an integrated solution for provisioning X.509 version 3 certificates for individuals, applications, and servers that require certificates for PKI-based operations such as authentication, SSL, S/MIME, and so on. For more information about OracleAS Certificate Authority, see *Oracle Application Server Certificate Authority Administrator's Guide*.

- **Oracle Wallet Manager**

  An Oracle wallet is a data structure that contains the private key of a user, a user certificate, and the set of trust points of a user (trusted certificate authorities). See *Oracle Database Advanced Security Administrator's Guide* for information about managing Oracle wallets.

  You can use Oracle Wallet Manager to manage Oracle wallets. This is a standalone Java application used to manage and edit the security credentials in Oracle wallets. It performs the following operations:

  – Generates a public-private key pair and creates a certificate request for submission to a certificate authority, and creates wallets

  – Installs a certificate for the entity

  – Manages X.509 version 3 certificates on Oracle Database clients and servers

  – Configures trusted certificates for the entity

  – 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**

  Because the certificate authority is trusted, these certificates verify 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.

Configuring Global User Authentication and Authorization

You can use Oracle Advanced Security to centralize the management of user-related information, including authorizations, in an LDAP-based directory service. This allows users and administrators to be identified in the database as global users, meaning that they are authenticated by SSL and that the management of these users is handled outside of the database by the centralized directory service. Global roles are defined in a database and are known only to that database, but the directory service handles authorizations for global roles.

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 *Oracle Database 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.

> **See Also:** "Strong Authentication and Centralized Management for Database Administrators" on page 3-16 if you want to centralize the management of `SYSDBA` or `SYSOPER` access

Creating a User Who Is Authorized by a Directory Service

You have the following options to specify users who are authorized by a directory service:

- Creating a Global User Who Has a Private Schema
- Creating Multiple Enterprise Users Who Share Schemas

Creating a Global User Who Has a Private Schema

The following statement shows the creation of a global user with a private schema, 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 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 Multiple Enterprise Users Who Share Schemas

Multiple enterprise users can share a single schema in the database. These users are authorized by the enterprise directory service but do not own individual private schemas in the database. These users are not individually created in the database. They connect to a shared schema in the database.

To create a schema-independent user:

1. Create a shared schema in the database as follows.

   ```
   CREATE USER appschema IDENTIFIED GLOBALLY AS '';
   ```

2. In the directory, create multiple enterprise users and a mapping object.

   The mapping object tells the database how you want to map the DNs for the users to the shared schema. You can either create a full DN mapping (one directory entry for each unique DN), or you can map, for each user, multiple DN components to one schema. For example:

   ```
   OU=division,O=Oracle,C=US
   ```

   See Also: *Oracle Database Enterprise User Security Administrator's Guide* for an explanation of these mappings

Most users do not need their own schemas, and implementing schema-independent users separates users from databases. You create multiple users who share the same schema in a database, and as enterprise users, they can also access shared schemas in other databases.

Advantages of Global Authentication and Global Authorization

Some advantages of global user authentication and authorization are as follows:

- Provides strong authentication using SSL, Kerberos, or Windows native authentication.
- Enables centralized management of users and privileges across the enterprise.
- Is easy to administer: You do not have to create a schema for every user in every database in the enterprise.
- Facilitates single sign-on: Users need to sign on once to only access multiple databases and services. Further, users using passwords can have a single password to access multiple databases accepting password-authenticated enterprise users.
- Because global user authentication and authorization provide password-based access, you can migrate previously defined password-authenticated database users to the directory (using the User Migration Utility) to be centrally administered. This makes global authentication and authorization available for earlier Oracle Database 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, that is, without storing the global user password in a link definition.

  See Also: The following manuals for additional information about global authentication and authorization and enterprise users and roles:

  - *Oracle Database Advanced Security Administrator's Guide*
  - *Oracle Database Enterprise User Security Administrator's Guide*

Configuring an External Service to Authenticate Users and Passwords

This section explores the following topics on external authentication:

- About External Authentication
- Advantages of External Authentication
- Creating a User Who Is Authenticated Externally
- Authenticating User Logins Using the Operating System
- Authentication User Logins Using Network Authentication

About External Authentication

When you use external authentication for user accounts, Oracle Database maintains the user account, but an external service performs the password administration and user authentication. 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, then it can authenticate users before they can log in to the database. To enable this feature, set the initialization parameter `OS_AUTHENT_PREFIX`, and use this prefix in Oracle Database user names. The `OS_AUTHENT_PREFIX` parameter defines a prefix that Oracle Database adds to the beginning of the operating system account name of every user. Oracle Database compares the prefixed user name with the Oracle Database user names in the database when a user attempts to connect.

You should set `OS_AUTHENT_PREFIX` to a null string (an empty set of double quotation marks: `""`). Using a null string eliminates the addition of any prefix to operating system account names, so that Oracle Database user names exactly match operating system user names.

```
OS_AUTHENT_PREFIX=" "
```

After you set `OS_AUTHENT_PREFIX`, it should remain the same for the life of a database. If you change the prefix, then 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.

The default value of this parameter is `OPS$` for backward compatibility with previous versions of Oracle Database. For example, assume that you set `OS_AUTHENT_PREFIX` 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 Database 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 installation and be authenticated by the operating system, then Oracle Database checks that there is a corresponding database user `OPS$tsmith` and, if so, lets the user connect. All references to a user authenticated by the operating system must include the prefix, `OPS$`, as seen in `OPS$tsmith`.

Advantages of External Authentication

The advantages of external authentication are as follows:

- More choices of authentication mechanisms are available, such as smart cards, fingerprints, Kerberos, or the operating system.
- Many network authentication services, such as Kerberos support single sign-on, enabling users to have fewer passwords to remember.
- If you are already using an external mechanism for authentication, such as one of those listed earlier, then there may be less administrative overhead to use that mechanism with the database.

Creating a User Who Is Authenticated Externally

The following statement creates a user who is identified by Oracle Database and authenticated by the operating system or a network service. This example assumes that the `OS_AUTHENT_PREFIX` parameter has been set to a blank space (`" "`).

```
CREATE USER scott IDENTIFIED EXTERNALLY;
```

Using the `CREATE USER ... IDENTIFIED EXTERNALLY` statement, you create database accounts that must be authenticated by the operating system or network service. Oracle Database then relies on this external login authentication when it provides that specific operating system user with access to the database resources of a specific user.

> **See Also:** *Oracle Database Advanced Security Administrator's Guide* for more information about external authentication

Authenticating User Logins Using the Operating System

By default, Oracle Database 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 `REMOTE_OS_AUTHENT` parameter to `TRUE` in the database initialization parameter file forces the database to accept the client operating system user name received over an unsecure connection and use it for account access. Because 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 Database without specifying a separate database user name or password. Also, user entries correspond in the database and operating system audit trails.

Authentication User Logins Using Network Authentication

Oracle Advanced Security performs network authentication, which you can configure to use a third-party service such as Kerberos. If you are using Oracle Advanced Security as your only external authentication service, then the `REMOTE_OS_AUTHENT`

parameter setting is irrelevant, because Oracle Advanced Security allows only secure connections.

Using Multitier Authentication and Authorization

In a multitier environment, Oracle Database 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 very busy middle tier, such as a transaction processing monitor, the identity of the clients connecting to the middle tier must be preserved. One advantage of using 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, you can use the Oracle Call Interface to create **lightweight sessions**, which enable 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 of or on a firewall, then security is better when each lightweight session has its own password. For an internal application server, lightweight sessions without passwords might be appropriate.

Administration and Security in 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.

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

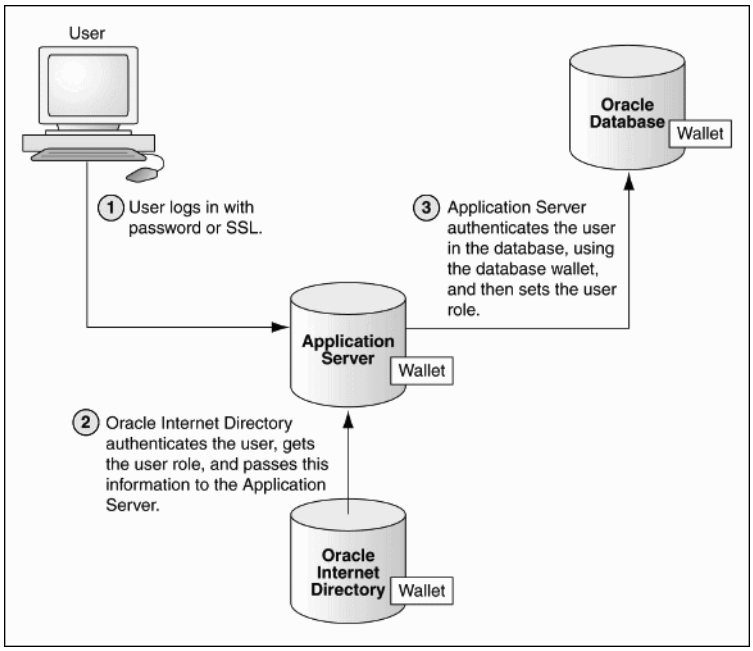
- The end user provides proof of authenticity to the application server, typically, by using a password or an X.509 certificate.
- The application server authenticates the end user and then authenticates itself to the database server.
- The database server authenticates the application server, verifies that the end user exists, and verifies that the application server has the privilege to connect for the end user.

Application servers can also enable roles for an end user on whose behalf they connect. The application server can obtain these roles from a directory, which 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 3–2 shows an example of multitier authentication.

Figure 3–2 Multitier Authentication



The following actions take place:

1. The user logs on using a password or Secure Sockets Layer. The authentication information is passed through Oracle Application Server.
2. Oracle Internet Directory authenticates the user, gets the roles associated with that user from the wallet, and then passes this information back to Oracle Application Server.
3. Oracle Application Server checks the identity of the user in Oracle Database, which contains a wallet that stores this information, and then sets the role for that user.

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

- **Accountability**. The database server must be able to distinguish between the actions of the application and the actions an application takes on behalf of a client. It must be possible to audit both kinds of actions.
- **Least privilege**. Users and middle tiers should be given the fewest privileges necessary to perform their actions, to reduce the danger of inadvertent or malicious unauthorized activities.

Preserving User Identity in Multitiered Environments

Many organizations would like to know who the user is through all tiers of an application without sacrificing the benefits of a middle tier. Oracle Database supports the following ways to preserve user identity through the middle tier of an application:

- Using a Middle Tier Server for Proxy Authentication
- Using Client Identifiers to Identify Application Users Not Known to the Database

Using a Middle Tier Server for Proxy Authentication

Oracle Database provides proxy authentication in Oracle Call Interface (OCI), thick JDBC, or thin 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.

The following sections explain how to use proxy authentication:

- About Proxy Authentication
- Advantages of Proxy Authentication
- Altering a User Account to Connect Through a Proxy
- Passing Through the Identity of the Real User by Using Proxy Authentication
- Limiting the Privilege of the Middle Tier
- Authorizing a Middle Tier to Proxy and Authenticate a User
- Authorizing a Middle Tier to Proxy a User Authenticated by Other Means
- Reauthenticating the User Through the Middle Tier to the Database
- Auditing Actions Taken on Behalf of the Real User

About Proxy Authentication

You can design a middle-tier server to authenticate clients in a secure fashion by using the following 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

> **Note:** The use of certificates for proxy authentication may not be supported in future Oracle Database releases.

In all cases, an administrator must authorize the middle-tier server to act on behalf of the client.

> **See Also:** *Oracle Call Interface Programmer's Guide* and *Oracle Database Advanced Application Developer's Guide* or details about designing a middle-tier server to proxy users

Advantages of Proxy Authentication

In multitier environments, proxy authentication controls 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 the following benefits to organizations:

- Organizations can separate application logic from data storage, partitioning the former in application servers and the latter in databases.
- 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 also lower their cost of computing by replacing many *thick clients* with a number of *thin clients* and an application server.

In addition, Oracle Database proxy authentication provides the following security benefits:

- A limited trust model, by controlling the users on whose behalf middle tiers can connect and the roles that the middle tiers can assume for the user
- Scalability, by supporting user sessions through OCI, thick JDBC, or thin JDBC, and eliminating the overhead of reauthenticating 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.

Altering a User Account to Connect Through a Proxy

To authorize a user account to connect using a proxy account, use the `GRANT CONNECT THROUGH` clause of the `ALTER USER` statement.

Example 3–6 shows how to alter user `preston` to connect through the proxy user `jward`.

Example 3–6 Altering a User Account to Connect Through a Proxy User Account

```
ALTER USER preston GRANT CONNECT THROUGH jward;
```

Afterward, user `jward` can connect using the `preston` proxy user as follows:

```
CONNECT jward[preston]
Enter password: password
```

Note the following:

- **Using roles with middle-tier clients.** You can also specify roles that the middle tier is permitted to activate when connecting as the client. Operations performed on behalf of a client by a middle-tier server can be audited.
- **Finding proxy users.** To find the users who are currently authorized to connect through a middle tier, query the `PROXY_USERS` data dictionary view, for example:

```
SELECT * FROM PROXY_USERS;
```

- **Removing proxy connections.** Use the `REVOKE CONNECT THROUGH` clause of `ALTER USER` to disallow a proxy connection. For example, to revoke user `preston` from connecting through the proxy user `jward`, enter the following statement:

  ```
  ALTER USER preston REVOKE CONNECT THROUGH jward
  ```

- **Password expiration and proxy connections.** Middle-tier use of password expiration does not apply to accounts that are authenticated through a proxy. Instead, lock the account rather than expire the password.

  See Also:

  - *Oracle Database SQL Language Reference* for a description and syntax of the proxy clause for `ALTER USER`
  - "Auditing SQL Statements and Privileges in a Multitier Environment" on page 6-26 for details about auditing operations done on behalf of a user by a middle tier

Passing Through the Identity of the Real User by Using Proxy Authentication

For enterprise users or database users, Oracle Call Interface, thick JDBC, or thin JDBC enables a middle tier to set up a number of user sessions within a single database connection, each of which uniquely identifies a connected user (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 by using a user name and password or an X.509 certificate by means of SSL.
2. The middle tier authenticates itself to the database by 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, thick JDBC, or thin JDBC.

   - If the user is a database user, then the session must, as a minimum, include the database user name. If the database requires it, then the session can include a password (which the database verifies against the password store in the database). The session can also include a list of database roles for the user.
   - If the user is an enterprise user, then the session may provide different information depending on how the user is authenticated.

     Example 1: If the user authenticates to the middle tier using SSL, then the middle tier can provide the DN from the X.509 certificate of the user, or the certificate itself in the session. The database uses the DN to look up the user in Oracle Internet Directory.

Example 2: 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, then the database will verify the password against Oracle Internet Directory. User 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 use the roles on behalf of the client.

4. The database verifies that the middle tier has the privilege to create sessions on behalf of the user.

   The `OCISessionBegin` call fails if the application server cannot perform a proxy authentication on behalf of the client by the administrator, or if the application server is not allowed to activate the specified roles.

The Authentication Process for Kerberos Tickets When a middle tier authenticates itself to the database as a proxy for the client in a Kerberos environment the following process occurs:

1. The client gets a **Forwardable Ticket Granting Ticket** (FTGT) from the Kerberos **Key Distribution Center (KDC)** (KDC).
2. The client submits the FTGT to the middle-tier application.
3. The middle-tier application initiates an OCI session by calling `OCISessionBegin` to create a new proxy session, passing the FTGT as an attribute associated with the request. The client responds as follows:
   a. The client-side OCI call submits the FTGT to the KDC and requests a Kerberos service ticket for the database server.
   b. The client-side OCI call submits the service ticket to the database server. Depending on network configuration, the FTGT may also be submitted to the database server.
4. The database server authenticates the proxy request based on the service ticket it receives from the middle-tier application. If the service ticket is valid, then the client-database session begins.

Note:

- Oracle Database Kerberos-based proxy authentication must use the Oracle Advanced Security Kerberos adapter.
- If a client is authenticated by using Kerberos, then the middle-tier proxy must also be authenticated by using the Oracle Advanced Security Kerberos adapter.

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. 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, then 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 use only the clerk role on her behalf.

An administrator 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, thick JDBC, or thin JDBC session for another database user, the database verifies that the middle tier is authorized to connect for that user by using the role specified.

Note: Instead of using default roles, create your own roles and assign only necessary privileges to them. Creating your own roles enables you to control the privileges granted by them and protects you if Oracle Database changes or removes default roles. For example, the CONNECT role now has only the CREATE SESSION privilege, the one most directly needed when connecting to a database.

However, CONNECT formerly provided several additional privileges, often not needed or appropriate for most users. Extra privileges can endanger the security of your database and applications. These have now been removed from CONNECT, and both CONNECT and RESOURCE roles will be deprecated in future releases of Oracle Database.

See Chapter 4, "Configuring Privilege and Role Authorization" for more information about roles.

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 (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 user mary, and midtier must also pass the user password to the database server for authorization.

Reauthenticating 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. An administrator 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.

The type of credentials can be either a password or a **Kerberos ticket**. When Kerberos tickets are passed to the database during proxy authentication, they are tested for validity. If the Kerberos tickets are invalid, the authentication attempt is rejected by the database server.

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 When you use password-based proxy authentication, Oracle Database passes the password of the client 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 computer does not have to have Oracle software installed on it to perform database operations.

To pass the password of the client, the middle-tier server calls the OCIAttrSet() function 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 connects to the database as a client who is an enterprise user, then 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 the Oracle Call Interface method `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 database uses its default certificate type of X.509.

Note:

- `OCI_ATTR_CERTIFICATE` is `DER encoded.`
- Certificate based proxy authentication using `OCI_ATTR_CERTIFICATE` will not be supported in future Oracle Database releases. Use the `OCI_ATTR_DISTINGUISHED_NAME` or `OCI_ATTR_USERNAME` attribute instead

If you are using proxy authentication for password-authenticated enterprise users, then use the same OCI attributes as for database users authenticated by password (`OCI_ATTR_USERNAME`). Oracle Database first checks the user name against the database. If it finds no user, then the database checks the user name in the directory. This user name must be globally unique.

Auditing Actions Taken on Behalf of the Real User

The proxy authentication features of 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 database administrator could enable auditing for `SELECT` statements performed on the `bonus` table that `hrappserver` initiates for Jane as follows:

```
AUDIT SELECT TABLE BY hrappserver ON BEHALF OF Jane;
```

Alternatively, you 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. You can enable separate auditing options to capture `SELECT` statements 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`, because 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`.

Using Client Identifiers to Identify Application Users Not Known to the Database

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 capture any value that the application uses for identification or access control, and passes it to the database. The `CLIENT_IDENTIFIER` attribute is supported in OCI, thick JDBC, and thin JDBC.

The following sections explain how to use client identifiers:

- How Client Identifiers Work in Middle Tier Systems
- Using the CLIENT_IDENTIFIER Attribute to Preserve User Identity
- Using CLIENT_IDENTIFIER Independent of Global Application Context

How Client Identifiers Work in Middle Tier Systems

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. You can use a `CLIENT_IDENTIFIER` attribute, which 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

You can use the `CLIENT_IDENTIFIER` predefined attribute of the built-in application context namespace, `USERENV`, to capture the application user name for use with global application context. You also can use the `CLIENT_IDENTIFIER` attribute independently. When you use the `CLIENT_IDENTIFIER` attribute independently from a global application context, you can set `CLIENT_IDENTIFIER` with the `DBMS_SESSION` interface. The ability to pass a `CLIENT_IDENTIFIER` to the database is supported in Oracle Call Interface (OCI), thick JDBC, and thin JDBC.

When you use the `CLIENT_IDENTIFIER` attribute 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 or her own session set up with individual application contexts, the application could set up global application contexts for gold partners, silver partners, and bronze partners. Then, use

the CLIENT_IDENTIFIER to point the session at the correct context 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:

- "Using Global Application Contexts" on page 7-20 for how to implement global application contexts
- "Example of Creating a Global Application Context That Uses a Client Session ID" on page 7-31

Using CLIENT_IDENTIFIER Independent of Global Application Context

Using the CLIENT_IDENTIFIER attribute is especially useful for those applications in which the 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. Because all user sessions are created as the same user, this security model makes it difficult to achieve data separation for each user. These applications can use the CLIENT_IDENTIFIER attribute to preserve the real application user identity through to the database.

With this approach, sessions can be reused by multiple users by changing the value of the CLIENT_IDENTIFIER attribute, which captures 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 added to the next OCI, thick JDBC, or thin 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 user name.

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, an application developer can use the client identifier to identify which lightweight user is currently using the database session. To set the `CLIENT_IDENTIFIER` attribute 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:

- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_SESSION` package
- *Oracle Call Interface Programmer's Guide* about how the `OCI_ATTR_CLIENT_IDENTIFIER` user session handle attribute is used in middle-tier applications
- The section about the `oracle.jdbc.OracleConnection` interface in the *Oracle Database JDBC Developer's Guide and Reference* for information about the `setClientIdentifer()` and the `clearClientIdentifier()` methods

4

Configuring Privilege and Role Authorization

Authorization includes primarily two processes:

- Permitting only certain users to access, process, or alter data.
- Applying varying limitations on user 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 discusses the following topics:

- About Privileges and Roles
- Who Should Be Granted Privileges?
- Managing System Privileges
- Managing User Roles
- Managing Object Privileges
- Granting User Privileges and Roles
- Revoking User Privileges and Roles
- Granting to and Revoking from the PUBLIC User Group
- Granting Roles Using the Operating System or Network
- When Do Grants and Revokes Take Effect?
- Managing Fine-Grained Access to External Network Services
- Finding Information About User Privileges and Roles

About Privileges and Roles

A user **privilege** is the right to run a particular type of SQL statement, or the right to access an object that belongs to another user, run a PL/SQL package, and so on. The types of privileges are defined by Oracle Database.

Roles are created by users (usually administrators) to group together privileges or other roles. They are a way to facilitate the granting of multiple privileges or roles to users.

This section describes the following general categories:

- **System privileges.** These privileges allow the grantee to perform standard administrator tasks in the database. Restrict them only to trusted users. "Managing System Privileges" on page 4-2 describes system privileges in detail.

- **User roles.** A **role** groups several privileges and roles, so that they can be granted to and revoked from users simultaneously. You must enable the role for a user before the user can use it. See "Managing User Roles" on page 4-6 for more information.
- **Object privileges.** Each type of object has privileges associated with it. "Managing Object Privileges" on page 4-20 describes how to manage privileges for different types of objects.

Who Should Be Granted Privileges?

You grant privileges to users so they can accomplish tasks required for their jobs. You should grant a privilege only to a user who requires that privilege to accomplish the necessary work. Excessive granting of unnecessary privileges can compromise security. For example, you never should grant `SYS` or `SYSDBA` privileges to users who do not perform administrative tasks.

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 also can 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 `ROBERT`.

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

See Also:

- "Guidelines for Securing User Accounts and Privileges" on page 10-2 for best practices to follow when granting privileges
- *Oracle Database SQL Language Reference* for the complete list of system privileges and their descriptions

Managing System Privileges

This section covers the following topics about system privileges:

- About System Privileges
- Why It Is Important to Restrict System Privileges
- Granting and Revoking System Privileges
- Who Can Grant or Revoke System Privileges?
- About ANY and PUBLIC Privileges

About 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. Each system privilege allows a user to perform a particular database operation or class of database operations. *Remember that system privileges are very powerful.* Only grant them when necessary to roles and trusted

users of the database. You can find a complete list of system privileges and their descriptions in *Oracle Database SQL Language Reference.*

Why It Is Important to Restrict System Privileges

Because system privileges are so powerful, you should configure your database to prevent typical (non-administrative) users from exercising the ANY system privileges (such as UPDATE ANY TABLE) on the data dictionary. See "Guidelines for Securing User Accounts and Privileges" on page 10-2 for additional guidelines about restricting system privileges.

- Restricting System Privileges by Securing the Data Dictionary
- Securing Scheduler Jobs That Run in the Schema of a Grantee
- Allowing Access to Objects in the SYS Schema

Restricting System Privileges by Securing the Data Dictionary

To secure the data dictionary, set the O7_DICTIONARY_ACCESSIBILITY initialization parameter to FALSE, which is the default value. This feature is called the dictionary protection mechanism.

The O7_DICTIONARY_ACCESSIBILITY initialization parameter controls restrictions on system privileges when you upgrade from Oracle Database release 7 to Oracle8*i* and later releases. If the parameter is set to TRUE, then access to objects in the SYS schema is allowed (Oracle Database release 7 behavior). Because the ANY privilege applies to the data dictionary, a malicious user with ANY privilege could access or alter data dictionary tables.

To set the O7_DICTIONARY_ACCESSIBILTY initialization parameter, modify it in the init*SID*.ora file. Alternatively, you can log on to SQL*Plus as SYS /AS SYSDBA and then enter an ALTER SYSTEM statement, assuming you have started the database using a server parameter file (SPFILE).

Example 4–1 shows how to set the O7_DICTIONARY_ACCESSIBILTY initialization parameter to FALSE by issuing an ALTER SYSTEM statement in SQL*Plus.

Example 4–1 Setting O7_DICTIONARY_ACCESSIBILITY to FALSE

```
ALTER SYSTEM SET O7_DICTIONARY_ACCESSIBILITY=FALSE SCOPE=SPFILE;
```

When you set O7_DICTIONARY_ACCESSIBILITY to FALSE, system privileges that enable access to objects in any schema (for example, users who have ANY privileges, such as CREATE ANY PROCEDURE) do not allow access to objects in the SYS schema. This means that access to the objects in the SYS schema (data dictionary objects) is restricted to users who connect as SYS or connect using the SYSDBA privilege.

System privileges that provide 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, regular views, packages, and synonyms). You can, however, grant these users explicit object privileges to access objects in the SYS schema.

See *Oracle Database Reference* for more information about the O7_DICTIONARY_ACCESSIBILITY initialization parameter.

Securing Scheduler Jobs That Run in the Schema of a Grantee

The CREATE EXTERNAL JOB privilege is automatically created in the schema of the grantee user so that operating system jobs can run outside the database. To support backward compatibility, by default, this privilege is granted to all existing users who have the CREATE JOB privilege. For greater security, grant the CREATE EXTERNAL JOB privilege only to SYS, database administrators, and those users who need it.

Allowing Access to 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.

Table 4–1 lists roles that you can grant to users who need access to objects in the SYS schema.

Table 4–1 Roles to Allow Access to SYS Schema Objects

Role	Description
SELECT_CATALOG_ROLE	Grant this role to allow users SELECT privileges on data dictionary views.
EXECUTE_CATALOG_ROLE	Grant this role to allow users EXECUTE privileges for packages and procedures in the data dictionary.
DELETE_CATALOG_ROLE	Grant this role to allow users to delete records from the system audit table (AUD$).

Additionally, you can grant the SELECT ANY DICTIONARY system privilege to users who require access to tables created in the SYS schema. 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, but it can be granted through a role.

> **Caution:** You should grant these roles and the SELECT ANY DICTIONARY system privilege with extreme care, because the integrity of your system can be compromised by their misuse.

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 exercise system privileges. For example, roles permit privileges to be made selectively available.

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

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

- GRANT and REVOKE SQL statements
- Oracle Enterprise Manager Database Control

See Also:

- "Granting User Privileges and Roles" on page 4-32
- "Revoking User Privileges and Roles" on page 4-36
- "When Do Grants and Revokes Take Effect?" on page 4-42
- "Finding Information About User Privileges and Roles" on page 4-57
- *Oracle Database 2 Day DBA* for more information about Database Control

Who Can Grant or Revoke System Privileges?

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

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

For this reason, only grant these privileges to trusted users.

About ANY and PUBLIC Privileges

System privileges that use the `ANY` keyword enable you to set privileges for an entire category of objects in the database. For example, the `CREATE ANY PROCEDURE` system privilege allows a user to create a procedure anywhere in the database. The behavior of an object created by users with the `ANY` privilege is not restricted to the schema in which it was created. For example, if user `MALCOEUR` has the `CREATE ANY PROCEDURE` privilege and creates a procedure in the schema `JONES`, then the procedure will run as `JONES`. However, `JONES` may not be aware that the procedure `MALCOEUR` created is running as him (`JONES`). If `JONES` has `DBA` privileges, letting `MALCOEUR` run a procedure as `JONES` could pose a security violation.

`PUBLIC` privileges are granted to every user in an Oracle database, and can be granted to roles and as users. Because objects created with the `PUBLIC` privilege are accessible to everyone, they can pose a security risk similar to `ANY` objects. For example, if `MALCOEUR` has the `CREATE PUBLIC SYNONYM` privilege, he could redefine an interface that he knows everyone else uses, and then point to it with the `PUBLIC SYNONYM` that he created. Instead of accessing the correct interface, users would access the interface of `MALCOEUR`, which could possibly perform illegal activities such as stealing the login credentials of users.

As you can see, these types of privileges are very powerful and could pose a security risk if given to the wrong person. Be careful about granting privileges using `ANY` or `PUBLIC`. As with all privileges, you should follow the principles of "least privilege" when granting these privileges to users.

To protect the data dictionary for `SYS` against users who have `ANY` privileges, set the `O7_DICTIONARY_ACCESSIBILITY` initialization parameter is set to `FALSE`. You can set this parameter by using an ALTER SYSTEM statement (see Example 4–1, "Setting O7_DICTIONARY_ACCESSIBILITY to FALSE" on page 4-3) or by modifying the `init`*`SID`*`.ora` file. See "Guidelines for Securing a Database Installation and Configuration" on page 10-9 for additional guidelines.

Managing User Roles

This section describes how to manage user roles:

- About User Roles
- Predefined Roles in an Oracle Database Installation
- Creating a Role
- Specifying the Type of Role Authorization
- Dropping Roles
- Restricting SQL*Plus Users from Using Database Roles
- Further Securing Role Privileges by Using Secure Application Roles

About User Roles

Managing and controlling privileges is easier when you use **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 user names 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.

This section explores the following topics:

- Properties of Roles and Why They Are Advantageous
- Common Uses of Roles
- How Roles Affect the Scope of a User's Privileges
- How Roles Work in PL/SQL Blocks
- How Roles Aid or Restrict DDL Usage
- How Operating Systems Can Aid Roles
- How Roles Work in a Distributed Environment

Properties of Roles and Why They Are Advantageous

Table 4–2 describes the properties of roles that enable easier privilege management within a database.

Table 4–2 Properties of Roles and Their Description

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, then 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.
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.

Table 4–2 (Cont.) Properties of Roles and Their Description

Property	Description
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 user name.
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. You should grant a secure application role all privileges necessary to run the application. You then can grant 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 role.

> **See Also:** "How Roles Aid or Restrict DDL Usage" on page 4-8 for information about restrictions for procedures

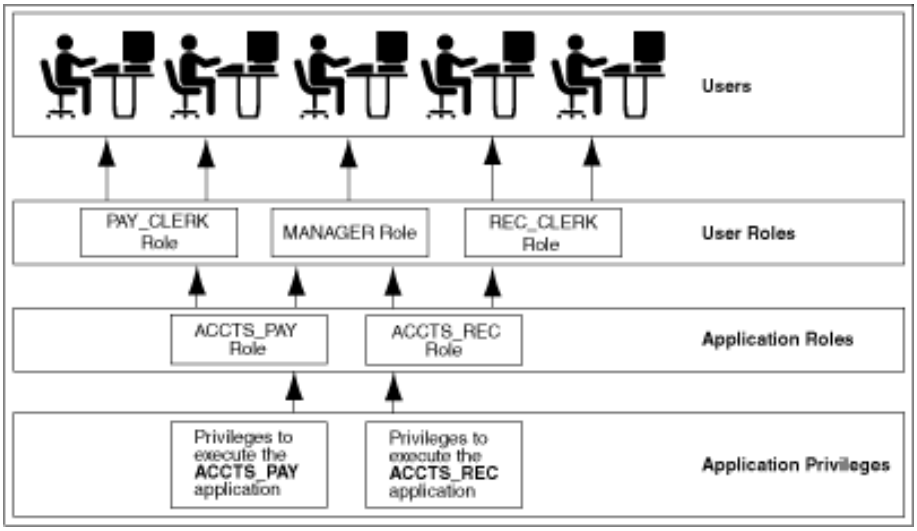
Common Uses of Roles

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

- To manage the privileges for a database application (see "Common Uses of Application Roles" on page 4-8)
- To manage the privileges for a user group (see "Common Uses of User Roles" on page 4-8)

Figure 4–1 and the sections that follow describe the two uses of roles.

Figure 4–1 Common Uses for Roles



Common Uses of Application Roles Grant an application role all privileges necessary to run a given database application. Then, 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.

Common Uses of User Roles Create a user role for a group of database users with common privilege requirements. You can manage user privileges by granting secure application roles and privileges to the user role and then granting the user role to appropriate users.

How Roles Affect the Scope of a User's Privileges

Each role and user has its own unique security domain. The security domain of a role includes the privileges granted to the role plus those privileges granted to any roles that are granted to the role.

The security domain of a user 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.) This domain also includes the privileges and roles granted to the user group `PUBLIC`. The `PUBLIC` user group represents all users in the database.

How Roles Work in PL/SQL Blocks

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.

Roles Used in 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`, then the query does not return any rows.

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

Roles Used in Named Blocks with Invoker's Rights and Anonymous PL/SQL blocks 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:**
>
> - *Oracle Database PL/SQL Language Reference* for an explanation of invoker's and definer's rights
> - *Oracle Database PL/SQL Language Reference* for information about dynamic SQL in PL/SQL

How Roles Aid or Restrict DDL Usage

A user requires one or more privileges to successfully execute a 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 a table that belongs to another user, 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 Database 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**: CREATE TABLE, CREATE VIEW, and CREATE PROCEDURE privileges
  - **Schema object privileges**: ALTER and INDEX privileges for a table

  Note: The REFERENCES object privilege for a table cannot be used to define the foreign key of a table 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 a table that belongs to another user.

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 also 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.

How Operating Systems Can Aid Roles

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

See Also: Your operating system-specific Oracle Database documentation for details about managing roles through the operating system

How Roles Work in a Distributed Environment

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

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

Predefined Roles in an Oracle Database Installation

Oracle Database provides a set of predefined roles to help in database administration. These roles, listed in Table 4–3, are automatically defined for Oracle databases when you run the standard scripts that are part of database creation. If you install other options or products, then other predefined roles may be created. 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.

Table 4–3 Oracle Database Predefined Roles

Predefined Role	Created by Script	Description
AQ_ADMINISTRATOR_ROLE	CATQUEUE.SQL	Provides privileges to administer Advanced Queuing. Includes ENQUEUE ANY QUEUE, DEQUEUE ANY QUEUE, and MANAGE ANY QUEUE, SELECT privileges on Advanced Queuing tables and EXECUTE privileges on Advanced Queuing packages.
AQ_USER_ROLE	CATQUEUE.SQL	Obsolete, but kept mainly for release 8.0 compatibility. Provides EXECUTE privileges on the DBMS_AQ and DBMS_AQIN packages.
CONNECT	SQL.BSQ	Provides the CREATE SESSION system privilege. This role is provided for compatibility with previous releases of Oracle Database. You can determine the privileges encompassed by this role by querying the DBA_SYS_PRIVS data dictionary view. **Note:** Oracle recommends that you design your own roles for database security rather than relying on this role. This role may not be created automatically by future releases of Oracle Database. **See Also:** *Oracle Database Reference* for a description of the DBA_SYS_PRIVS view

Table 4–3 (Cont.) Oracle Database Predefined Roles

Predefined Role	Created by Script	Description
DBA	SQL.BSQ	Provides all system privileges WITH ADMIN OPTION. This role is provided for compatibility with previous releases of Oracle Database. You can determine the privileges encompassed by this role by querying the DBA_SYS_PRIVS data dictionary view. **Note:** Oracle recommends that you design your own roles for database security rather than relying on this role. This role may not be created automatically by future releases of Oracle Database. **See Also:** *Oracle Database Reference* for a description of the DBA_SYS_PRIVS view
DELETE_CATALOG_ROLE	SQL.BSQ	Provides the DELETE privilege on the system audit table (AUD$)
EXECUTE_CATALOG_ROLE	SQL.BSQ	Provides EXECUTE privileges on objects in the data dictionary. Also provides the HS_ADMIN_ROLE privilege.
EXP_FULL_DATABASE	CATEXP.SQL	Provides the privileges required to perform full and incremental database exports, and 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. This role is provided for convenience in using the export and import utilities. **See Also:** *Oracle Database Utilities* for more information about these roles
HS_ADMIN_ROLE	CATHS.SQL	Provides privileges for DBAs who need to use the DBA role using Oracle Database Heterogeneous Services to access appropriate tables in the data dictionary. Used to protect access to the Heterogeneous Services (HS) 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. **See Also:** *Oracle Database Heterogeneous Connectivity Administrator's Guide* for more information
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. This role is provided for convenience in using the export and import utilities. **See Also:** *Oracle Database Utilities* for more information about these roles
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

Table 4–3 (Cont.) Oracle Database Predefined Roles

Predefined Role	Created by Script	Description
RESOURCE	SQL.BSQ	Provides the following system privileges: CREATE CLUSTER, CREATE INDEXTYPE, CREATE OPERATOR, CREATE PROCEDURE, CREATE SEQUENCE, CREATE TABLE, CREATE TRIGGER, CREATE TYPE. This role is provided for compatibility with previous releases of Oracle Database. You can determine the privileges encompassed by this role by querying the DBA_SYS_PRIVS data dictionary view. **Note:** Oracle recommends that you design your own roles for database security rather than relying on this role. This role may not be created automatically by future releases of Oracle Database. **See Also:** *Oracle Database Reference* for a description of the DBA_SYS_PRIVS view
SCHEDULER_ADMIN	EXECSCH.SQL	Allows the grantee to execute the procedures of the DBMS_SCHEDULER package. It includes all of the job scheduler system privileges and is included in the DBA role. **See Also:** *Oracle Database Administrator's Guide* for more information about the DBMS_SCHEDULER package
SELECT_CATALOG_ROLE	SQL.BSQ	Provides SELECT privilege on objects in the data dictionary. Also provides the HS_ADMIN_ROLE privilege.
XDBADMIN	CATQM.SQL	Allows the grantee to register an XML schema globally, as opposed to registering it for use or access only by its owner. It also lets the grantee bypass access control list (ACL) checks when accessing Oracle XML DB Repository. **See Also:** *Oracle XML DB Developer's Guide* for information about XML schemas and the XML DB Repository
XDB_SET_INVOKER	CATXEV.SQL	Allows the grantee to define invoker's rights handlers and to create or update the resource configuration for XML repository triggers. By default, Oracle Database grants this role to the DBA role but not to the XDBADMIN role. **See Also:** *Oracle XML DB Developer's Guide* for information about Oracle Database XML repository triggers

Table 4–3 (Cont.) Oracle Database Predefined Roles

Predefined Role	Created by Script	Description
XDB_WEBSERVICES	CATXDBC1.SQL	Allows the grantee to access Oracle Database Web services over HTTPS. However, it does not provide the user access to objects in the database that are public. To allow public access, you need to grant the user the XDB_WEBSERVICES_WITH_PUBLIC role. For a user to use these Web services, SYS must enable the Web service servlets. **See Also:** *Oracle XML DB Developer's Guide* for information about Oracle Database Web services
XDB_WEBSERVICES_OVER_HTTP	CATXDBC1.SQL	Allows the grantee to access Oracle Database Web services over HTTP. However, it does not provide the user access to objects in the database that are public. To allow public access, you need to grant the user the XDB_WEBSERVICES_WITH_PUBLIC role. **See Also:** *Oracle XML DB Developer's Guide* for information about Oracle Database Web services
XDB_WEBSERVICES_WITH_PUBLIC	CATXDBC1.SQL	Allows the grantee access to public objects through Oracle Database Web services. **See Also:** *Oracle XML DB Developer's Guide* for information about Oracle Database Web services

Note: Each installation should create its own roles and assign only those privileges that are needed, thus retaining detailed control of the privileges in use. This process also removes any need to adjust existing roles, privileges, or procedures whenever Oracle Database changes or removes roles that Oracle Database defines. For example, the CONNECT role now has only one privilege: CREATE SESSION. Both CONNECT and RESOURCE roles will be deprecated in future Oracle Database releases.

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 user names 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, then the encrypted role name and password combination is considerably less secure. See Guideline 1 in "Guidelines for Securing Passwords" on page 10-6 for password guidelines.

Example 4–2 creates the clerk role, which is authorized by the database using the password morework2do.

Example 4–2 Creating a User Role Authorized by a Password

```
CREATE ROLE clerk IDENTIFIED BY morework2do;
```

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 the following sections.

You can set or change the authorization method for a role using the `ALTER ROLE` statement.

Example 4–3 shows how to alter the `clerk` role to specify that the user must have been authorized by an external source before enabling the role.

Example 4–3 Altering a Role to be Authorized by an External Source

```
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 Language Reference* for syntax, restrictions, and authorization information about the SQL statements used to manage roles and privileges

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.

This section describes the following ways you can authorize roles:

- Authorizing a Roles by Using the Database
- Authorizing a Role by Using an Application
- Authorizing a Role by Using an External Source

> **See Also:** "When Do Grants and Revokes Take Effect?" on page 4-42 for a discussion about enabling roles

Authorizing a Roles by Using the Database

You can protect a role authorized by the database by assigning the role a password. If a user is granted a role protected by a password, then 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 connection time, then the user is not required to enter a password.

Example 4–2, "Creating a User Role Authorized by a Password" on page 4-14shows a `CREATE ROLE` statement that creates a role called `clerk`. When it is enabled, the password `morework2do` must be supplied.

Note: In a database that uses a multibyte character set, passwords for roles must include only single-byte characters. Multibyte characters are not accepted in passwords. See Guideline 1 in "Guidelines for Securing Passwords" on page 10-6 for password guidelines.

Authorizing a Role by Using an Application

An application role (secure application role) can be enabled only by applications using an authorized PL/SQL 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.

To create a role enabled by an authorized PL/SQL package, use the `IDENTIFIED USING` *`package_name`* clause in the `CREATE ROLE` SQL statement.

Example 4–4 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`.

Example 4–4 Creating a Role Authorized by a PL/SQL Package for an Application

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

When you enable the default roles of the user at login as specified in the user profile, Oracle Database performs no checking for application roles.

See the following for more information about secure application roles:

- "Further Securing Role Privileges by Using Secure Application Roles" on page 4-19
- "Creating a Secure Application Role to Control Access to Applications" on page 5-4
- *Oracle Database 2 Day + Security Guide*

Authorizing a Role by Using an External Source

You can create roles that are authorized by the operating system or network clients.

Example 4–5 creates a role named `accts_rec` and requires that the user is authorized by an external source before it can be enabled:

Example 4–5 Creating a Role Authorized by an External Source

```
CREATE ROLE accts_rec IDENTIFIED EXTERNALLY;
```

Authorizing a Role by Using 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 operating system account of the user.

If a role is authorized by the operating system, then 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, then you do not need to have the operating system authorize them also.

> **See Also:** "Granting Roles Using the Operating System or Network" on page 4-40 for more information about roles granted by the operating system

Authorizing a Role by Using a Network Clients If users connect to the database over Oracle Net, then 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. Oracle recommends that you set `REMOTE_OS_ROLES` to `FALSE`, which is the default.

If you are not concerned with this security risk and want to use operating system role authentication for network clients, then set the initialization parameter `REMOTE_OS_ROLES` in the database initialization parameter file to `TRUE`. The change will take effect the next time you start the instance and mount the database.

Global Role Authorization by an Enterprise Directory Service

A role can be defined as a global role, where 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.

Example 4–6 creates a global role.

Example 4–6 Creating 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 you can grant it to an enterprise role defined in the enterprise directory. An enterprise role is a directory structure that contains global roles on multiple databases and can be granted to enterprise users.

See "Configuring Global User Authentication and Authorization" on page 3-25 for a general discussion of global authentication and authorization of users, and its role in enterprise user management.

> **See Also:** *Oracle Database Enterprise User Security Administrator's Guide* for information about implementing enterprise user management

Granting and Revoking Roles

You can grant system or schema object privileges 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, 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 Database permits applications and users to enable and disable roles. Each role granted to a user is, at any given time, either enabled or disabled. The security domain of a user 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. You can explicitly enable or disable it 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:

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

Privileges are granted to and revoked from roles using the same options.

See Also:

- "Granting User Privileges and Roles" on page 4-32
- "Revoking User Privileges and Roles" on page 4-36
- "When Do Grants and Revokes Take Effect?" on page 4-42
- "Finding Information About User Privileges and Roles" on page 4-57
- *Oracle Database 2 Day DBA* for more information about Database Control

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. (A global role is managed in a directory, such as Oracle Internet Directory, but its privileges are contained within a single database.) By default, the `SYS` or `SYSTEM` user has this privilege. 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 Enterprise User Security Administrator's Guide* for information about global roles

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 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 user 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;
```

Restricting SQL*Plus Users from Using Database Roles

This section describes features that you can use to restrict SQL*Plus users from using database roles and thus, prevent serious security problems.

- Potential Security Problems of Using Ad Hoc Tools
- Limiting Roles Through the PRODUCT_USER_PROFILE Table
- Using Stored Procedures to Encapsulate Business Logic

Potential Security Problems of Using Ad Hoc Tools

Prebuilt database applications explicitly control the potential actions of a user, including the enabling and disabling of user roles while using the application. By contrast, ad hoc query tools such as SQL*Plus, permit a user to submit any SQL statement (which may or may not succeed), including enabling and disabling a 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.

Limiting Roles Through the PRODUCT_USER_PROFILE Table

You can use the `PRODUCT_USER_PROFILE` table, which is in the `SYSTEM` schema, 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. You can even restrict access to the `GRANT`, `REVOKE`, and `SET ROLE` commands to control user ability to change their database privileges.

The `PRODUCT_USER_PROFILE` table enables you to list roles that you do not want users to activate with an application. You can also explicitly disable the use of various commands, such as `SET ROLE`.

For example, you could create an entry in the `PRODUCT_USER_PROFILE` table to:

- Disallow the use of the `clerk` and `manager` roles with SQL*Plus
- Disallow the use of `SET ROLE` with SQL*Plus

Suppose user Marla connects to the database using SQL*Plus. Marla has the `clerk`, `manager`, and `analyst` roles. As a result of the preceding entry in `PRODUCT_USER_PROFILE`, Marla is only able to exercise her `analyst` role with SQL*Plus. Also, when Ginny 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`.

Be aware that 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 Marla had other privileges granted to her directly, then 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

Using Stored Procedures to Encapsulate Business Logic

Stored procedures encapsulate the 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 can create a procedure to update the 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 security 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.

Further Securing Role Privileges by Using Secure Application Roles

A secure application role is a role that can be enabled only by an authorized PL/SQL package. The PL/SQL package itself reflects the security policies needed to control access to the application.

This method of role creation restricts the enabling of such roles to the invoking application. For example, the application can perform authentication and customized authorization, such as checking whether the user has connected through a proxy.

This type of role strengthens security because passwords are not embedded in application source code or stored in a table. This way, the actions the database performs are based on the implementation of your security policies, and these definitions are stored in one place, the database, rather than in your applications. If you need to modify the policy, you do so in one place without having to modify your applications. No matter how many users connect to the database, the result is always the same, because the policy is bound to the role.

When you enable the secure application role, Oracle Database verifies that the authorized PL/SQL package is on the calling stack, that is, it verifies that the authorized PL/SQL package is issuing the command to enable the role. Also, when you enable the default user roles, Oracle Database performs no checking for application roles. So, it is important that you do not make the secure application role the default role of the user, for the role to be checked by the security policy before being granted.

You can use secure application roles 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.

> **See Also:**
>
> - "Creating a Secure Application Role to Control Access to Applications" on page 5-4
> - *Oracle Database 2 Day + Security Guide*

Managing Object Privileges

This section describes how to manage object privileges:

- About Object Privileges
- Granting or Revoking Object Privileges
- Managing Schema Object Privileges
- Managing Table Privileges
- Managing View Privileges
- Managing Procedure Privileges
- Managing Type Privileges

About Object Privileges

An **object 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 a stored procedure of another user

Granting or Revoking 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 `GRANT` and `REVOKE` statement. If all object privileges are granted using the `ALL` shortcut, then individual privileges can still be revoked.

Similarly, you can revoke all individually granted privileges 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), then you must include the `CASCADE CONSTRAINTS` option in the `REVOKE` statement.

Example 4–7 revokes all privileges on the orders table in the `HR` schema using `CASCADE CONSTRAINTS`.

Example 4–7 Revoking All Object Privileges Using CASCADE CONSTRAINTS

```
REVOKE ALL
 ON orders FROM hr
 CASCADE CONSTRAINTS;
```

> **See Also:** *Oracle Database SQL Language Reference* for the complete list of `GRANT` object privileges

Managing Schema Object Privileges

A **schema object privilege** is the permission 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 sections discuss granting and revoking such privileges:

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

The following sections discuss object privileges that apply to specific schema objects:

- Managing Table Privileges
- Managing View Privileges
- Sequences (see *Oracle Database Administrator's Guide* for information on managing sequences)
- Managing Procedure Privileges
- Functions and Packages(*Oracle Database Administrator's Guide* for information on managing object dependencies)
- Managing 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, then you can make the privileges selectively available.

You can grant or revoke object privileges for users and roles using the following:

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

> **See Also:** *Oracle Database 2 Day DBA* for more information about Database Control

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 the user 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.

> **See Also:** *Oracle Database SQL Language Reference* for information about `GRANT` and `GRANT ANY OBJECT PRIVILEGE`

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 by using a synonym.

For example, assume there is a table `jward.emp` with a synonym named `jward.employee`. 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 by 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, then 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`, then `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, then all grants for the underlying schema object remain in effect, even if the privileges were granted by specifying the dropped synonym.

Managing Table Privileges

Schema object privileges for tables enable table security at the DML (data manipulation language) or DDL (data definition language) level of operation.

The following sections discuss table privileges and DML and DDL operations:

- How Table Privileges Affect Data Manipulation Language Operations
- How Table Privileges Affect Data Definition Language Operations

How Table Privileges Affect Data Manipulation Language 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 data in a table.

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

For example, if you do not want data entry users to alter the `salary` column of the `employees` table, then 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 Language Reference* for more information about DML operations

How Table Privileges Affect Data Definition Language 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 these 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, you can grant the REFERENCES privilege 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

Managing View Privileges

This section discusses how to manage view privileges. It explores the following topics:

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

About 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. Its selected data can be thought of as the result of a stored query. A view contains no actual data but rather derives what it shows from the tables and views on which it is based. You can query a view, and change the data it represents. 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.

You can apply DML object privileges to views, similar to tables. Schema object privileges for a view allow various DML operations, which as noted affect the base tables from which the view is derived.

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 the schema of another user)
- 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

- In addition, 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, then grantees cannot access your view.

  See Also: *Oracle Database SQL Language 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.

Managing Procedure Privileges

This section discusses how to manage procedure privileges. It explores the following topics:

- Using the EXECUTE Privilege for Procedure Privileges
- Procedure Execution and Security Domains
- System Privileges Needed to Create or Alter a Procedure
- How Procedure Privileges Affect Packages and Package Objects

Using the EXECUTE Privilege for Procedure Privileges

EXECUTE is the only **schema object privilege** for procedures, including standalone procedures and functions and as packages. Grant this privilege only to users who need to run a procedure or to compile another procedure that calls a desired procedure.

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. Oracle Database does not perform a run-time privilege check when the procedure is called. A user with the EXECUTE ANY PROCEDURE system privilege can execute any procedure in the database. Privileges to run procedures can be granted to a user through roles.

How Procedure Privileges Affect Definer's Rights

The owner of a procedure, 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, then the owner of the 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 called the *invoker*. Additional privileges on referenced objects are required for invoker's rights procedures, but not for definer's rights procedures.

See Also: "How Roles Work in PL/SQL Blocks" on page 4-8

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. This is 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 owner of the procedure must have all the necessary object privileges for referenced objects. Fewer privileges have to be granted to users of a definer's rights procedure. This results in stronger 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 run time, Oracle Database checks the privileges of the owner of a definer's rights stored procedure before the procedure is executed. If a necessary privilege on a referenced object was revoked from the owner of a definer's rights procedure, then the procedure cannot be run by the owner or any other user.

Note: Trigger processing follows the same patterns as definer's rights procedures. The user runs a SQL statement, which that user is privileged to run. 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*

How Procedure Privileges Affect Invoker's Rights

An invoker's rights procedure executes with all of the invoker's privileges. Oracle Database enables roles unless a definer's rights procedure calls the invoker's rights

procedure directly or indirectly. 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 schema of the invoker.

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

For all other external references, such as direct PL/SQL function calls, Oracle Database checks the privileges of the owner at compile time, and does not performs a run-time check. Therefore, the user of an invoker's rights procedure does not need 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.

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 run an entry-point procedure can also execute internal program units indirectly, but cannot directly call the internal programs.

See Also:

- "Configuring an Oracle Virtual Private Database Policy" on page 8-5
- *Oracle Database PL/SQL Packages and Types Reference* for detailed documentation of the Oracle Database 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 need to have been explicitly granted the necessary privileges (system or object) on all objects referenced by the procedure. You cannot obtain the required privileges through roles. This includes the `EXECUTE` privilege for any procedures that are called inside the procedure being created.

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

How Procedure Privileges Affect 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 can access or modify the value of any public package variable. You cannot grant specific `EXECUTE` privileges for individual constructs in a package. Therefore, you may find it useful to consider two alternatives for establishing security when developing procedures, functions, and packages for a database application. The following examples describe these alternatives.

Procedure Privileges and Packages and Package Objects: Example 1

Example 4–8 shows four procedures created in the bodies of two packages.

Example 4–8 Package Objects Affected by Procedure Privileges

```
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;
```

The following `GRANT EXECUTE` statements enable the `big_bosses` and `little_bosses` roles to run the appropriate procedures:

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

> **Note:** Granting `EXECUTE` privilege for a package provides uniform access to all package objects.

Procedure Privileges and 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;
```

Managing Type Privileges

The following sections 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

Table 4–4 lists system privileges for named types (object types, `VARRAY`s, and nested tables).

Table 4–4 System Privileges for Named Types

Privilege	Enables 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 `RESOURCE` role includes 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, then 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 methods in the type, including the type constructor. This is similar to the 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:** "Managing Procedure Privileges" on page 4-24

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 the schema of another user. These privileges can be acquired explicitly or through a role.
- The owner of the type must be 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 obtain the required privileges through roles.
- If the type owner intends to grant access to the type to other users, then the owner must receive the EXECUTE privileges to the referenced types with the GRANT OPTION or the EXECUTE ANY TYPE system privilege with the ADMIN OPTION. If not, then 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 the following 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 were granted the EXECUTE ANY TYPE system privilege. The owner cannot obtain the required privileges through roles.
- If the table owner intends to grant access to the table to other users, then the owner must have the EXECUTE privileges to the referenced types with the GRANT OPTION or the EXECUTE ANY TYPE system privilege with the ADMIN OPTION. If not, then the table owner has insufficient privileges to grant access on the type to other users.

> **See Also:** "Managing Table Privileges" on page 4-22 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

The following DDL is run in the schema of user1:

```
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;
```

The following DDL is performed in the schema of `user2`:

```
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.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.type1` with the `GRANT OPTION`:

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

The following statements can be successfully run by `user3`:

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

Note: Customers should discontinue using the `CONNECT` and `RESOURCE` roles, as they will be deprecated in future Oracle Database releases. The `CONNECT` role presently retains only the `CREATE SESSION` privilege.

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.

Table 4–5 lists the privileges for object tables.

Table 4–5 Privileges for Object Tables

Privilege	Enables you to...
`SELECT`	Access an object and its attributes from the table
`UPDATE`	Modify the attributes of the objects that make up the rows in the table
`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 to interpret the type instance images. When a client requests type information, Oracle Database checks for the 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;
```

In addition, consider the following two queries:

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

For either query, Oracle Database checks the SELECT privilege of the user 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 Database checks the EXECUTE privilege of the user.

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

In addition, by 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 Database 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 Database to check for the SELECT privilege on the containing object table.
- Modifying an existing object or flushing an object from the object cache causes Oracle Database to check for the UPDATE privilege on the destination object table.
- Flushing a new object causes Oracle Database to check for the INSERT privilege on the destination object table.
- Deleting an object causes Oracle Database to check for the DELETE privilege on the destination table.
- Pinning an object of a named type causes Oracle Database to check EXECUTE privilege on the object.

Modifying the attributes of an object in a client third-generation language application causes Oracle Database to update the entire object. Therefore, the user needs the UPDATE privilege on the object table. Having the 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 Database 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 on which tables depend. 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 are when necessary privileges required by the type are revoked, or the type or dependent types are dropped. If 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 was dropped can never be accessed again, and the only permissible action is to drop the table.

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

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

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 "Using a Middle Tier Server for Proxy Authentication" on page 3-31.

Granting System Privileges and Roles

You can use the `GRANT` SQL statement to grant system privileges and roles to users and roles. The following privileges are required:

- To grant a system privilege, a user must be granted the system privilege with the `ADMIN OPTION` or was granted the `GRANT ANY PRIVILEGE` system privilege.
- To grant a role, a user must be granted the role with the `ADMIN OPTION` or was granted the `GRANT ANY ROLE` system privilege.

Example 4–9 grants the system privilege `CREATE SESSION` and the `accts_pay` role to the user `jward`.

Example 4–9 Granting a System Privilege and a Role to a User

```
GRANT CREATE SESSION, accts_pay TO jward;
```

> **Note:** Object privileges cannot be granted along with system privileges and roles in the same `GRANT` statement.

Granting the ADMIN OPTION

A user or role that is granted a privilege or role, which specifies the `WITH ADMIN OPTION` clause, has the following 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 grant the system privilege or role with ADMIN OPTION.
- The grantee of a role can alter or drop the role.

Example 4–10 grants the new_dba role with the WITH ADMIN OPTION clause to user michael.

Example 4–10 Granting the ADMIN OPTION

```
GRANT new_dba TO michael WITH ADMIN OPTION;
```

User michael is able to not only use all of the privileges implicit in the new_dba role, but he can also grant, revoke, and drop the new_dba role as deemed necessary. Because of these powerful capabilities, use caution when granting system privileges or roles with the ADMIN OPTION. These privileges are usually reserved for a security administrator, and are rarely granted to other administrators or users of the system.

Note: 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 Database enables you to create a new user with the GRANT statement. If you specify a password using the IDENTIFIED BY clause, and the user name and password do not exist in the database, then a new user with that user name and password is created.

Example 4–11 creates psmith as a new user while granting psmith the CONNECT system privilege.

Example 4–11 Creating a New User with the GRANT Statement

```
GRANT CONNECT TO psmith IDENTIFIED BY two_4_one;
```

See Also: "Creating User Accounts" on page 2-1

Granting Object Privileges

You can 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 have been granted the GRANT ANY OBJECT PRIVILEGE system privilege. This privilege 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.

Example 4–12 grants the SELECT, INSERT, and DELETE object privileges for all columns of the emp table to the users jfee and tsmith.

Example 4–12 Granting Object Privileges to Users

```
GRANT SELECT, INSERT, DELETE ON emp TO jfee, tsmith;
```

To grant all object privileges on the `salary` view to user `jfee`, use the `ALL` keyword as shown in the following example:

```
GRANT ALL ON salary TO jfee;
```

> **Note:** A grantee cannot regrant access to objects unless the original grant included the `GRANT OPTION`. Thus in the example just given, `jfee` cannot use the `GRANT` statement to grant object privileges to anyone else.

Specifying the GRANT OPTION

Specify the `WITH GRANT OPTION` clause with the `GRANT` statement 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 user in the database, with or without the `GRANT OPTION`, and to any role in the database.
- If both of the following conditions are true, then 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.

> **Note:** The `GRANT OPTION` is not valid when granting an object privilege to a role. Oracle Database 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 enables users to grant and revoke any object privilege on behalf of the object owner. This privilege 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. Login credentials do not need to be maintained for schema owners who have this privilege, which reduces the number of connections required during configuration.

This system privilege is part of the Oracle Database 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`.

The *recorded* grantor of access rights to an object is either the object owner or the person exercising the `GRANT ANY OBJECT PRIVILEGE` system privilege. If the grantor with `GRANT ANY OBJECT PRIVILEGE` does *not* have the object privilege with the `GRANT OPTION`, then the object owner is shown as the grantor. Otherwise, when that grantor has the object privilege with the `GRANT OPTION`, then that grantor is recorded as the grantor of the grant.

Note: The audit record generated by the GRANT statement always shows the actual user who performed the grant.

For example, consider the following scenario. 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, then you will see that hr is shown as the grantor of the privilege:

```
SELECT GRANTEE, OWNER, GRANTOR, PRIVILEGE, GRANTABLE
  FROM DBA_TAB_PRIVS
  WHERE TABLE_NAME = 'EMPLOYEES' and OWNER = 'HR';

GRANTEE  OWNER GRANTOR PRIVILEGE    GRANTABLE
-------- ----- ------- -----------  ----------
BLAKE    HR    HR       SELECT      YES
```

Now assume that user 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 query the DBA_TAB_PRIVS view again, 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 4-37

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, ensure that each NOT NULL column can either be inserted into 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 the INSERT privilege on the acct_no column of the accounts table to user scott:

```
GRANT INSERT (acct_no) ON accounts TO scott;
```

In the following example, object privilege for the ename and job columns of the emp table are granted 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 (OLS).

See Also:

- Chapter 8, "Using Oracle Virtual Private Database to Control Data Access"
- "Adding Policies for Column-Level Oracle Virtual Private Database" on page 8-8
- *Oracle Label Security Administrator's Guide*

Revoking User Privileges and Roles

This section describes the following aspects of revoking user privileges and roles:

- 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 user tsmith:

```
REVOKE CREATE TABLE, accts_rec FROM tsmith;
```

Note: The ADMIN OPTION for a system privilege or role cannot be selectively revoked. Instead, revoke the privilege or role, and then grant the privilege or role again but without the ADMIN OPTION.

Revoking 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 object privilege is revoked.

Assuming you are the original grantor, the following statement revokes the SELECT and INSERT privileges on the emp table from 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. Instead, revoke the object privilege and then granted it again but 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 was 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 Database only revokes the object privilege granted by the user issuing the REVOKE statement. This can be illustrated by continuing the example started in "Granting Object Privileges on Behalf of the Object Owner" on page 4-34.

At this point, user blake 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 user HR also grants the SELECT privilege on HR.EMPLOYEES to user 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 user clark granted by user 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, then 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 4-34

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 grant again the column-specific privileges that should remain.

For example, assume that role human_resources was 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 the UPDATE privilege on all columns of the dept table from the role human_resources. The GRANT statement then grants again the 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), then 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. This is discussed in the following sections:

- Cascading Effects When Revoking System Privileges
- Cascading Effects When Revoking Object Privileges

Cascading Effects When Revoking 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 user jfee with the ADMIN OPTION.
2. User jfee creates a table.
3. User jfee grants the CREATE TABLE system privilege to user tsmith.
4. User tsmith creates a table.
5. The security administrator revokes the CREATE TABLE system privilege from user jfee.
6. The table created by user jfee continues to exist. User tsmith still has the table and the CREATE TABLE system privilege.

You can observe cascading effects when you revoke a system privilege related to a DML operation. If the SELECT ANY TABLE privilege is revoked from a user, then all

procedures contained in the users schema relying on this privilege fails until the privilege is reauthorized.

Cascading Effects When Revoking Object Privileges

Revoking an object privilege can have cascading effects. Remember the following:

- **Object definitions that depend on a DML object privilege can be affected if the DML object privilege is revoked.** For example, assume that the 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, then 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 that are defined by the user and require the dropped REFERENCES privilege are automatically dropped.** For example, assume that user `jward` is granted the `REFERENCES` privilege for the `deptno` column of the `dept` table. This user now creates a foreign key on the `deptno` column in the `emp` table that references the `deptno` column of the `dept` table. If the `REFERENCES` privilege on the `deptno` column of the `dept` table is revoked, then 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 the object privilege of a grantor 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` statement is also cascaded to `user2`. Any objects that depend on the revoked `SELECT` privilege of `user1` and `user2` can also be affected, as described earlier.

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 a table that belongs to another user, then the index continues to exist after the privilege is revoked.

Granting to and Revoking from the PUBLIC User Group

You can grant and revoke privileges and roles 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 have only the privileges required to accomplish the current group 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` or `UPDATE ON emp`), then all procedures in the database, including functions and packages, must be *reauthorized* before they can be used again. Therefore, be careful when you grant and revoke DML-related privileges to or from `PUBLIC`.

See Also:

- Managing Object Dependencies in *Oracle Database Administrator's Guide* for more information about object dependencies
- "Guidelines for Securing Data" on page 10-8

Granting Roles Using the Operating System or Network

This section describes the following aspects of granting roles through your operating system or network:

- About Granting Roles Using the Operating System or Network
- Using Operating System Role Identification
- Using Operating System Role Management
- Granting and Revoking Roles When OS_ROLES Is Set to TRUE
- Enabling and Disabling Roles When OS_ROLES Is Set to TRUE
- Using Network Connections with Operating System Role Management

About Granting Roles Using the Operating System or Network

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 Database can grant roles to users at connect time. Roles can be administered using the operating system and passed to Oracle Database when a user creates a session. As part of this mechanism, the default roles of a user and the roles granted to a user with the `ADMIN OPTION` can be identified. If the operating system is used to authorize users for roles, then 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 the database roles of a user is that privilege management for an Oracle database can be externalized. The security facilities offered by the operating system control user 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 database user roles.
- UNIX Oracle administrators want UNIX groups to identify database user roles.
- VMS Oracle administrators want to use rights identifiers to identify database user roles.

The main disadvantage of using the operating system to identify the database roles of a user is that privilege management can only be performed at the role level. Individual privileges cannot be granted using the operating system, but they can still be granted inside the database using `GRANT` statements.

A second 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 as described in "Using Network Connections with Operating System Role Management" on page 4-42.

> **Note:** The features described in this section are available only on some operating systems. See your operating system-specific Oracle Database documentation to determine if you can use these features.

Using Operating System Role Identification

To cause a database to use the operating system to identify the database roles of each user 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 tries to create a session with the database, Oracle Database initializes the user security domain using the database roles identified by the operating system.

To identify database roles for a user, the operating system account for each Oracle Database user 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]]
```

In this specification:

- `ID` has a definition that varies on different operating systems. For example, on VMS, `ID` is the instance identifier of the database; on VMS, it is the computer type; and 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` character is specified, then precede that character 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 Database, `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, remember 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 Database 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 Is Set to TRUE

If the `OS_ROLES` parameter is set to `TRUE`, then the operating system completely manages the granting and revoking of roles to users. Any previous granting of roles to users using `GRANT` statements do not apply. However, they are still listed in the data dictionary. Only the role grants to users made at the operating system level 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`, then the user can grant the role only to other roles.

Enabling and Disabling Roles When OS_ROLES Is Set to TRUE

If the `OS_ROLES` initialization parameter is set to `TRUE`, then 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 the operating system account of a user cannot be specified in a `SET ROLE` statement, even if a role was granted using a `GRANT` statement when `OS_ROLES = FALSE`. (If you specify such a role, then Oracle Database 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 have the operating system manage roles, then, 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 an unsecure 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, then set the initialization parameter `REMOTE_OS_ROLES` to `TRUE`. The change takes effect the next time you start the instance and mount the database. The default setting of this parameter is `FALSE`.

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 and revokes of system and object privileges to anything (users, roles, and `PUBLIC`) take immediate effect.
- All grants and revokes of roles to anything (users, other roles, `PUBLIC`) take effect only when a current user session issues a `SET ROLE` statement to reenable the role

after the grant and revoke, or when a new user session is created after the grant or revoke.

You can see which roles are currently enabled by examining the SESSION_ROLES data dictionary view.

How the SET ROLE Statement Affects Grants and Revokes

During the user 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. The user must already be granted the roles that are named in the SET ROLE statement. To specify the maximum number of roles number of roles that can be concurrently enabled, set the initialization parameter MAX_ENABLED_ROLES.

Example 4–13 enables the role clerk, which you have already been granted, and specifies the password.

Example 4–13 Using SET ROLE to Grant a Role and Specify a Password

```
SET ROLE clerk IDENTIFIED BY morework2do;
```

Example 4–14 shows how to use SET ROLE to disable all roles.

Example 4–14 Using SET ROLE to Disable All Roles

```
SET ROLE NONE;
```

Specifying Default Roles

When a user logs on, Oracle Database enables all privileges granted explicitly to the user and all privileges in the default roles of the user.

You can set and alter a list of default roles for a user by using the ALTER USER SQL statement. The ALTER USER statement specifies roles that are to be enabled when a user connects to the database, without requiring the user to specify the role passwords. The user must have 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).

Example 4–15 sets the default roles payclerk and pettycash for user jane:

Example 4–15 Using ALTER USER to Set Default Roles

```
ALTER USER jane DEFAULT ROLE payclerk, pettycash;
```

You cannot set default roles for a user in the CREATE USER statement. When you first create a user, the default user 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 default user roles.

Caution: When you create a role (other than a user role), it is granted implicitly and added as a default role. When the MAX_ENABLED_ROLES initialization parameter is set, users receive an error at login if they have more roles than MAX_ENABLED_ROLES specifies. You can avoid this error by changing the default user roles to be less than MAX_ENABLED_ROLES. Therefore, you should change the DEFAULT ROLE settings of SYS and SYSTEM before creating user roles. See *Oracle Database Reference* for more information about MAX_ENABLED_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.

For detailed information about the MAX_ENABLED_ROLES initialization parameter, see *Oracle Database Reference*.

Managing Fine-Grained Access to External Network Services

You can configure fine-grained access control for users and roles that need to access external network services from the database. This way, specific groups of users can connect to one or more host computers, based on privileges that you grant them. Typically, you use this feature to control access to applications that run on specific host addresses.

This section includes the following topics:

- About Fine-Grained Access to Database Network Services
- Upgrading Applications That Depend on the PL/SQL Network Utility Packages
- Creating an Access Control List for Database Network Services
- Examples of Creating Access Control Lists
- Using Wildcard Characters in Network Host Computers
- Precedence Order for a Host Computer in Multiple Access Control List Assignments
- Precedence Order for a Host in Access Control List Assignments with Port Ranges
- Checking Privilege Assignments That Affect User Access to a Network Host
- Setting the Precedence of Multiple Users and Roles in One Access Control List
- Using Data Dictionary Views to Find Information About Access Control Lists

About Fine-Grained Access to Database Network Services

To configure fine-grained access to database network services, you create an access control list (ACL), which is stored in Oracle XML DB. You can create the access control list by using Oracle XML DB itself, or by using the DBMS_NETWORK_ACL_ADMIN and DBMS_NETWORK_ACL_UTILITY PL/SQL packages. This guide explains how to use these packages to create and manage the access control list. To create an access control list by using Oracle XML DB and for general conceptual information about access control lists, see *Oracle XML DB Developer's Guide*.

This feature enhances security for network connections because it restricts the external network hosts that a database user can connect to using the PL/SQL network utility packages such as UTL_TCP, UTL_SMTP, UTL_MAIL, UTL_HTTP, and UTL_INADDR. Otherwise, an intruder who gained access to the database could maliciously attack the network, because, by default, the PL/SQL utility packages are created with the EXECUTE privilege granted to PUBLIC users.

Upgrading Applications That Depend on the PL/SQL Network Utility Packages

If you have upgraded from a previous release of Oracle Database, and your applications depend on PL/SQL network utility packages such as UTL_TCP, UTL_SMTP, UTL_MAIL, UTL_HTTP, and UTL_INADDR, the following error may occur when you try to run the application:

```
ORA-24247: network access denied by access control list (ACL)
```

Use the procedures in this section to reconfigure the network access for the application. See also *Oracle Database Upgrade Guide* for compatibility issues for applications that depend on the PL/SQL network utility packages. For detailed information on the network utility packages, see *Oracle Database PL/SQL Packages and Types Reference*.

Creating an Access Control List for Database Network Services

When you create access control lists for network connections, you should create one access control list dedicated to a group of common users, for example, users who need access to a particular application that resides on a specific host computer. For ease of administration and for good system performance, do not create too many access control lists. Network hosts accessible to the same group of users should share the same access control list.

To create the access control list by using the DBMS_NETWORK_ACL_ADMIN package, follow these steps:

- Step 1: Create the Access Control List and Its Privilege Definitions
- Step 2: Assign the Access Control List to One or More Network Hosts

Step 1: Create the Access Control List and Its Privilege Definitions

Use the DBMS_NETWORK_ACL_ADMIN.CREATE_ACL procedure to create the content of the access control list. It contains a name of the access control list, a brief description, and privilege settings for one user or role that you want to associate with the access control list. In an access control list, privileges for each user or role are grouped together as an access control entry (ACE). An access control list must have the privilege settings for at least one user or role.

Note: You cannot import or export the access control list settings by using the Oracle Database import or export utilities such as Oracle Data Pump.

The syntax for creating an access control list is as follows:

```
BEGIN
 DBMS_NETWORK_ACL_ADMIN.CREATE_ACL (
  acl          => 'file_name.xml',
  description  => 'file description',
  principal    => 'user_or_role',
  is_grant     => TRUE|FALSE,
  privilege    => 'connect|resolve',
  start_date   => null|timestamp_with_time_zone,
  end_date     => null|timestamp_with_time_zone);
END;
```

In this specification:

- `acl`: Enter a name for the access control list XML file. Oracle Database creates this file relative to the `/sys/acls` directory in the XML DB Repository in the database. Include the `.xml` extension. For example:

  ```
  acl => 'us-mycompany-com-permissions.xml',
  ```

- `description`: Enter a brief description of the file's purpose. For example:

  ```
  description => 'Network connection permission for ACCT_MGR role',
  ```

- `principal`: Enter the user account or role being granted or denied permissions. For example:

  ```
  principal => 'ACCT_MGR',
  ```

  Enter the name of the user account or role in case sensitive characters. For example, if the database stores the user name `preston` in all capital letters, entering it in mixed or lower case will not work. You can find the user accounts and roles in the current database instance by querying the `DBA_USERS` and `DBA_ROLES` views, for example:

  ```
  SELECT USERNAME FROM DBA_USERS;
  SELECT ROLE FROM DBA_ROLES;
  ```

- `is_grant`: Enter either `TRUE` or `FALSE`, to indicate whether the privilege is to be granted or denied. For example:

  ```
  is_grant => TRUE,
  ```

- `privilege`: Enter either `connect` or `resolve`. This setting is case sensitive, so always enter it in lowercase. For example:

  ```
  privilege => 'connect',
  ```

  A database user needs the `connect` privilege to an external network host computer if he or she is connecting using the `UTL_TCP`, `UTL_HTTP`, `UTL_SMTP`, and `UTL_MAIL` utility packages. To resolve the host name that was given a host IP address, or the IP address that was given a host name, with the `UTL_INADDR` package, grant the database user the `resolve` privilege instead.

You can use the data dictionary views described in "Using Data Dictionary Views to Find Information About Access Control Lists" on page 4-57 to find more information about existing privileges and network connections.

- `start_date`: (Optional) Enter the start date for the access control entry (ACE), in `TIMESTAMP WITH TIME ZONE` format (YYYY-MM-DD HH:MI:SS.FF TZR). When specified, the access control entry will be valid only on or after the specified date. The default is `null`. For example, to set a start date of February 28, 2007, at 6:30 a.m. in San Francisco, California, U.S., which is in the Pacific time zone:

  ```
  start_date => '2006-02-28 06:30:00.00 US/Pacific',
  ```

  The `NLS_TIMESTAMP_FORMAT` initialization parameter sets the default timestamp format. See *Oracle Database Reference* for more information.

- `end_date`: (Optional) Enter the end date for the access control entry (ACE), in `TIMESTAMP WITH TIME ZONE` format (YYYY-MM-DD HH:MI:SS.FF TZR). When specified, the access control entry will expire after the specified date. The `end_date` setting must be greater than or equal to the `start_date` setting. The default is `null`.

  For example, to set an end date of December 10, 2007, at 11:59 p.m. in San Francisco, California, U.S., which is in the Pacific time zone:

  ```
  end_date => '2007-12-10 23:59:00.00 US/Pacific');
  ```

To add more users or roles to the access control list, use the `DBMS_NETWORK_ACL.ADD_PRIVILEGE` procedure. The syntax is as follows:

```
BEGIN
 DBMS_NETWORK_ACL_ADMIN.ADD_PRIVILEGE (
  acl         => 'file_name.xml',
  principal   => 'user_or_role',
  is_grant    => TRUE|FALSE,
  privilege   => 'connect|resolve',
  position    => null|value,
  start_date  => null|timestamp_with_time_zone,
  end_date    => null|timestamp_with_time_zone);
END;
```

As you can see, the parameters to add the privilege are the similar to those in the `CREATE_ACL` procedure, except that `description` is not included and the `position` parameter, which sets the order of precedence for multiple users or roles, was added. Because you now are adding more than one user or role, you may want to consider setting their precedence. "Setting the Precedence of Multiple Users and Roles in One Access Control List" on page 4-56 provides more information.

Other `DBMS_NETWORK_ACL_ADMIN` procedures that are available for this step are `DELETE_PRIVILEGE` and `DROP_ACL`.

At this stage, you have created an access control list that defines the privileges needed to connect to a network host. However, the access control list has no effect until you complete Step 2: Assign the Access Control List to One or More Network Hosts.

Step 2: Assign the Access Control List to One or More Network Hosts

After you create the access control list, then you are ready to assign it to one or more network host computers. You can use the `DBMS_NETWORK_ACL_ADMIN.ASSIGN_ACL` procedure to do so.

For example:

```
BEGIN
 DBMS_NETWORK_ACL_ADMIN.ASSIGN_ACL (
  acl         => 'file_name.xml',
  host        => 'network_host',
  lower_port  => null|port_number,
  upper_port  => null|port_number);
END;
```

In this specification:

- `acl`: Enter the name of the access control list XML file to assign to the network host. Oracle Database creates this file relative to the `/sys/acls` directory in the XML DB Repository in the database. Include the `.xml` extension. For example:

  ```
  acl => 'us-mycompany-com-permissions.xml',
  ```

- `host`: Enter the network host to which this access control list will be assigned. This setting can be a name or IP address of the network host. Host names are case-insensitive. For example:

  ```
  host => 'us.mycompany.com',
  ```

  See the following sections for more information about how network host computers in access control list assignments work:

  - "Using Wildcard Characters in Network Host Computers" on page 4-52
  - "Checking Privilege Assignments That Affect User Access to a Network Host" on page 4-53
  - "Precedence Order for a Host Computer in Multiple Access Control List Assignments" on page 4-52
  - "Precedence Order for a Host in Access Control List Assignments with Port Ranges" on page 4-52

- `lower_port`: (Optional) For TCP connections, enter the lower boundary of the port range. Use this setting for the `connect` privilege only; omit it for the `resolve` privilege. The default is `null`. For example:

  ```
  lower_port => 80,
  ```

- `upper_port`: (Optional) For TCP connections, enter the upper boundary of the port range. Use this setting for `connect` privileges only; omit it for `resolve` privileges. The default is `null`. For example:

  ```
  upper_port => 3999);
  ```

  If you enter a value for the `lower_port` and leave the `upper_port` at `null` (or just omit it), Oracle Database assumes the `upper_port` setting is the same as the `lower_port`. For example, if you set `lower_port` to `80` and omit `upper_port`, the `upper_port` setting is assumed to be `80`.

  The `resolve` privilege in the access control list takes no effect when a port range is specified in the access control list assignment.

Only one access control list can be assigned to any host computer, domain, or IP subnet, and if specified, the TCP port range. When you assign a new access control list to a network target, Oracle Database unassigns the previous access control list that was assigned to the same target. However, Oracle Database does not drop the access control list. You can drop the access control list by using the `DROP_ACL` procedure. To remove an access control list assignment, use the `UNASSIGN_ACL` procedure.

Depending on how you create and maintain the access control list, the two steps may overlap. For example, you can create an access control list that has privileges for five users in it, and then apply it to two host computers. Later on, you can modify this access control list to have different or additional users and privileges, and assign it to different or additional host computers.

All access control list changes, including the assignment to network hosts, are transactional. They do not take effect until the transaction is committed.

You can find information about existing privileges and network connections by using the data dictionary views described in Table 4–6, " Data Dictionary Views That Display Information about Access Control Lists" on page 4-57.

For information about using the `DBMS_NETWORK_ACL_ADMIN` package, see *Oracle Database PL/SQL Packages and Types Reference*.

Examples of Creating Access Control Lists

The following examples demonstrate how to create access control lists.

- Example of Creating a Simple Access Control List
- Example of an Access Control List with Multiple Roles Assigned to Multiple Hosts

Example of Creating a Simple Access Control List

Example 4–16 shows how you would create an access control list called `us-mycompany-com-permissions.xml` to grant users who have the `ACCT_MGR` role access to network services that run on the host `us.mycompany.com`.

Example 4–16 Creating an Access Control List for a Single Role and Network Connection

```
-- First, create the access control list, which includes one role:

BEGIN
 DBMS_NETWORK_ACL_ADMIN.CREATE_ACL (
  acl          => 'us-mycompany-com-permissions.xml',
  description  => 'Network connection permission for ACCT_MGR',
  principal    => 'ACCT_MGR',
  is_grant     => TRUE,
  privilege    => 'connect');
END;

-- Second, assign the access control list a network host:

BEGIN
 DBMS_NETWORK_ACL_ADMIN.ASSIGN_ACL (
  acl         => 'us-mycompany-com-permissions.xml',
  host        => 'www.us.mycompany.com',
  lower_port  => 80,
  upper_port  => 80);
END;
```

This example creates the `us-mycompany-com-permissions.xml` file in the `/sys/acls` directory, which is the default location. The XML file appears as follows:

```
<acl description="Network connection permission for ACCT_MGR"
  xmlns="http://xmlns.oracle.com/xdb/acl.xsd"
  xmlns:plsql="http://xmlns.oracle.com/plsql"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
  xsi:schemaLocation="http://xmlns.oracle.com/xdb/acl.xsd
http://xmlns.oracle.com/xdb/acl.xsd">
  <security-class>plsql:network</security-class>
  <ace>
    <grant>true</grant>
    <principal>ACCT_MGR</principal>
    <privilege><plsql:connect/></privilege>
 </ace>
</acl>
```

The `xmlns` and `xsi` elements are fixed and should not be modified, for example, in a text editor.

You can check the contents of the access control list in SQL*Plus. See *Oracle XML DB Developer's Guide* for examples.

Example of an Access Control List with Multiple Roles Assigned to Multiple Hosts

Example 4–17 shows how to create a slightly more complex version of the `us-mycompany-com-permissions.xml` access control list. In this example, you specify multiple role privileges and their precedence position, and assigned to multiple host computers.

See"Using Wildcard Characters in Network Host Computers" on page 4-52 and "Precedence Order for a Host Computer in Multiple Access Control List Assignments" on page 4-52 for more information about host names. See also "Setting the Precedence of Multiple Users and Roles in One Access Control List" on page 4-56 to determine the order of multiple `ACE` elements in the access control list XML file.

Example 4–17 Creating an Access Control List for Multiple Roles and Network Connections

```
-- Create the access control list:

BEGIN
 DBMS_NETWORK_ACL_ADMIN.CREATE_ACL (
  acl          => 'us-mycompany-com-permissions.xml',
  description  => 'Network connection permission for ACCT_MGR and ACCT_CLERK',
  principal    => 'ACCT_MGR',
  is_grant     => TRUE,
  privilege    => 'resolve');
 DBMS_NETWORK_ACL_ADMIN.ADD_PRIVILEGE ( -- Creates the second role privilege
  acl          => 'us-mycompany-com-permissions.xml',
  principal    => 'ACCT_CLERK',
  is_grant     => TRUE,
  privilege    => 'connect',
  position     => null);
END;

-- Assign the access control list to hosts:

BEGIN
 DBMS_NETWORK_ACL_ADMIN.ASSIGN_ACL ( -- Creates the first target host
  acl          => 'us-mycompany-com-permissions.xml',
  host         => '*.us.mycompany.com');
 DBMS_NETWORK_ACL_ADMIN.ASSIGN_ACL ( -- Creates the second target host
  acl          => 'us-mycompany-com-permissions.xml',
  host         => '*.uk.mycompany.com',
  lower_port   => 80,
  upper_port   => 99);
```

```
END;
```

The us-mycompany-com-permissions.xml appears as follows:

```
<acl description="Network connection permission for ACCT_MGR and ACCT_CLERK"
  xmlns="http://xmlns.oracle.com/xdb/acl.xsd"
  xmlns:plsql="http://xmlns.oracle.com/plsql"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.oracle.com/xdb/acl.xsd
http://xmlns.oracle.com/xdb/acl.xsd">
  <security-class>plsql:network</security-class>
  <ace>
    <grant>true</grant>
    <principal>ACCT_MGR</principal>
    <privilege><plsql:resolve/></privilege>
  </ace>
  <ace>
    <grant>true</grant>
    <principal>ACCT_CLERK</principal>
    <privilege><plsql:connect/></privilege>
  </ace>
</acl>
```

Example 4–18 shows how the DBA_NETWORK_ACL_PRIVILEGES data dictionary view displays the privilege granted in the previous access control list.

Example 4–18 Using the DBA_NETWORK_ACL_PRIVILEGES View to Show Granted Privileges

```
ACL
ACLID                PRINCIPAL  PRIVILEGE IS_GRANT INVERT
START_DATE END_DATE
-----------------------------------------
------------------------------- ---------- -------   -------- -------
---------- ----------
/sys/acls/us-mycompany-com-permissions.xml 2EF86135D0E29B2AE040578CE4043250 ACCT_
MGR   resolve   true   false
/sys/acls/us-mycompany-com-permissions.xml 2EF86135D0E29B2AE040578CE4043250 ACCT_
CLERK connect   true   false
```

Example 4–19 shows how the DBA_NETWORK_ACLS data dictionary view displays the host assignment of the access control list.

Example 4–19 Using the DBA_NETWORK_ACLS View to Show Host Assignments

```
HOST         LOWER_PORT UPPER_PORT
ACL
ACLID
-------------------- ---------- ----------
-----------------------------------------
--------------------------------
*.us.mycompany.com
/sys/acls/us-mycompany-com-permissions.xml 2EF86135D0E29B2AE040578CE4043250
*.uk.mycompany.com     80     99
/sys/acls/us-mycompany-com-permissions.xml 2EF86135D0E29B2AE040578CE4043250
```

In these examples, the ACCT_MGR role has the resolve privilege to the first host, and the ACCT_CLERK role has the connect privilege to the first and second target hosts. The

ACCT_MGR role does not have the resolve privilege to the second host because a port range is specified in the assignment to the second host.

To check the contents of the access control list in SQL*Plus, see *Oracle XML DB Developer's Guide* for examples.

Using Wildcard Characters in Network Host Computers

If you want to assign an access control list to a group of network host computers, you can use the asterisk (*) wildcard character. For example, enter *.mycompany.com for host computers that belong to a domain or 192.168.0.* for IP addresses that belong to an IP subnet. The asterisk wildcard must be at the beginning, before a period (.) in a domain, or at the end, after a period (.), in an IP subnet. For example, *.mycompany.com is valid, but *mycompany.com and *.mycompany.* are not.

Be aware that the use of wildcard characters affects the order of precedence for multiple access control lists that are assigned to the same host computer.

Precedence Order for a Host Computer in Multiple Access Control List Assignments

For multiple access control lists that are assigned to the host computer and its domains, the access control list that is assigned to the host computer takes precedence over those assigned to the domains. The access control list assigned to a domain has a lower precedence than those assigned to the subdomains.

For example, Oracle Database first selects the access control list assigned to the host server.us.mycompany.com, ahead of other access control lists assigned to its domains. If additional access control lists were assigned to the sub domains, their order of precedence is as follows:

1. server.us.mycompany.com
2. *.us.mycompany.com
3. *.mycompany.com
4. *.com
5. *

Similarly, here is how the order of precedence works for the IP address 111.222.0.4:

1. 111.222.0.4
2. 111.222.0.*
3. 111.222.*
4. 111.*
5. *

Precedence Order for a Host in Access Control List Assignments with Port Ranges

When an access control list is assigned to a host computer, a domain, or an IP subnet with a port range, it take precedence over the access control list assigned to the same host, domain, or IP subnet without a port range.

For example, for TCP connections to any port between port 80 and 99 at server.us.mycompany.com, Oracle Database first selects the access control list assigned to port 80 through 99 at server.us.mycompany.com, ahead of the other access control list assigned to server.us.mycompany.com that is without a port range.

Checking Privilege Assignments That Affect User Access to a Network Host

Database administrators can use the DBA_NETWORK_ACL_PRIVILEGES data dictionary view to query network privileges that have been granted to or denied from database users and roles in the access control lists, and whether those privileges take effect during certain times only. Using the information provided by the view, you may need to combine the data to determine if a user is granted the privilege at the current time, the roles the user has, the order of the access control entries, and so on. To simplify this privilege evaluation, you can use the following DBMS_NETWORK_ACL_ADMIN functions to check the privilege granted to a user in an access control list:

- CHECK_PRIVILEGE: Checks if the specified privilege is granted to or denied from the specified user in an access control list. This procedure identifies the access control list by its path in the XML DB Repository. Use CHECK_PRIVILEGE if you want to evaluate a single access control list with a known path.
- CHECK_PRIVILEGE_ACLID: Similar to the CHECK_PRIVILEGE procedure, except that it enables you to specify the object ID of the access control list. Use CHECK_PRIVILEGE_ACLID if you need to evaluate multiple access control lists, when you query the DBA_NETWORK_ACLS data dictionary view. For better performance, call CHECK_PRIVILEGE_ACLID on multiple access control lists rather than using CHECK_PRIVILEGE on each one individually.

Users without database administrator privileges do not have the privilege to access the access control lists or to invoke those DBMS_NETWORK_ACL_ADMIN functions. However, they can query the USER_NETWORK_ACL_PRIVILEGES data dictionary view to check their privileges instead.

Both database administrators and users can use the following DBMS_NETWORK_ACL_UTILITY functions to generate the list of domains or IP subnet a host belongs to and to sort the access control lists by their order of precedence according to their host assignments:

- DOMAINS: Returns a list of the domains or IP subnets whose access control lists may affect permissions to a specified network host, subdomain, or IP subnet
- DOMAIN_LEVEL: Returns the domain level of a given host

The following sections explain how database administrators and users can check permissions for the user to connect to a network host or to perform domain name resolutions:

- How a DBA Can Check User Network Connection and Domain Privileges
- How Users Can Check Their Network Connection and Domain Privileges

How a DBA Can Check User Network Connection and Domain Privileges

A database administrator can query the DBA_NETWORK_ACLS view to determine which access control lists are present for a specified host computer. This view shows the access control lists that determine the access to the network connection or domain, and then determines if each access control list grants (GRANTED), denies (DENIED), or does not apply (NULL) to the access privilege of the user. Only the database administrator can query this view.

This section provides examples that demonstrate how the database administrator can check user privileges for network connections and domain name resolution.

- Database Administrator Checking User Connection Privileges
- Database Administrator Checking User Privileges for Domain Name Resolution

Database Administrator Checking User Connection Privileges

Example 4–20 shows how a database administrator can check the privileges for user `preston` to connect to `www.us.mycompany.com`. Remember that the user name you enter for the `user` parameter in the `CHECK_PRIVILEGE_ACLID` procedure is case sensitive. In this example, entering the user name `preston` is correct, but entering `Preston` or `preston` is incorrect.

You can find the users in the current database instance by querying the `DBA_USERS` data dictionary view, for example:

```
SELECT USERNAME FROM DBA_USERS;
```

Example 4–20 Administrator Checking User Permissions for Network Host Connections

```
SELECT HOST, LOWER_PORT, UPPER_PORT, ACL,
   DECODE(
     DBMS_NETWORK_ACL_ADMIN.CHECK_PRIVILEGE_ACLID(aclid,  'PRESTON', 'connect'),
     1, 'GRANTED', 0, 'DENIED', null) PRIVILEGE
FROM DBA_NETWORK_ACLS
WHERE host IN
  (SELECT * FROM
     TABLE(DBMS_NETWORK_ACL_UTILITY.DOMAINS('www.us.mycompany.com')))
 ORDER BY
  DBMS_NETWORK_ACL_UTILITY.DOMAIN_LEVEL(host) DESC, LOWER_PORT, UPPER_PORT;

 HOST                 LOWER_PORT UPPER_PORT ACL                  PRIVILEGE
 -------------------- ---------- ---------- -------------------- ---------
 www.us.mycompany.com         80         80 /sys/acls/www.xml    GRANTED
 www.us.mycompany.com       3000       3999 /sys/acls/www.xml    GRANTED
 www.us.mycompany.com                       /sys/acls/www.xml    GRANTED
 *.mycompany.com                            /sys/acls/all.xml
 *                                          /sys/acls/all.xml
```

In this example, user `preston` was granted privileges for all the network host connections found for `www.us.mycompany.com`. However, suppose `preston` had been granted access to a host connection on port 80, but then denied access to the host connections on ports 3000–3999. In this case, you need to create one access control list for the host connection on port 80, and a separate access control list for the host connection on ports 3000–3999.

Database Administrator Checking User Privileges for Domain Name Resolution

Example 4–21 shows how a database administrator can check the privileges of user `preston` to perform domain name resolution for the host `www.us.mycompany.com`. In this example, only the access control lists assigned to hosts without a port range because the `resolve` privilege has no effect to those with a port range. (Remember that the user name you enter for the `user` parameter in `CHECK_PRIVILEGE_ACLID` is case sensitive.)

Example 4–21 Administrator Checking Permissions for Domain Name Resolution

```
SELECT HOST, ACL, DECODE(
     DBMS_NETWORK_ACL_ADMIN.CHECK_PRIVILEGE_ACLID(aclid, 'PRESTON', 'resolve'),
     1, 'GRANTED', 0, 'DENIED', null) privilege
FROM DBA_NETWORK_ACLS
WHERE host IN
 (SELECT * FROM
    TABLE(DBMS_NETWORK_ACL_UTILITY.DOMAINS('www.us.mycompany.com'))) AND
 LOWER_PORT IS NULL AND UPPER_PORT IS NULL
ORDER BY DBMS_NETWORK_ACL_UTILITY.DOMAIN_LEVEL(host) DESC;
```

```
HOST                 ACL                  PRIVILEGE
-------------------- -------------------- ---------
www.us.mycompany.com /sys/acls/www.xml    GRANTED
*.mycompany.com      /sys/acls/all.xml
*                    /sys/acls/all.xml
```

How Users Can Check Their Network Connection and Domain Privileges

Users can query the USER_NETWORK_ACL_PRIVILEGES view to check their network and domain permissions. The USER_NETWORK_ACL_PRIVILEGES view is PUBLIC, so all users can select from it.

This view hides the access control lists from the user. It evaluates the permission status for the user (GRANTED or DENIED) and filters out the NULL case because the user does not need to know when the access control lists do not apply to him or her. In other words, Oracle Database only shows the user on the network hosts that explicitly grant or deny access to him or her. Therefore, the output does not display the *.mycompany.com and * that appear in the output from the database administrator-specific DBA_NETWORK_ACLS view.

These sections provide examples that demonstrate how a database administrator can check user permissions for network connections and domain name resolution.

- User Checking His or Her Network Connection Privileges
- User Checking Own Privileges for Domain Name Resolution

User Checking His or Her Network Connection Privileges

Example 4–20 shows how user preston can check her privileges to connect to www.us.mycompany.com.

Example 4–22 User Checking Permissions for Network Host Connections

```
SELECT HOST, LOWER_PORT, UPPER_PORT, STATUS PRIVILEGE
FROM USER_NETWORK_ACL_PRIVILEGES
WHERE host IN
   (SELECT * FROM
      TABLE(DBMS_NETWORK_ACL_UTILITY.DOMAINS('www.us.mycompany.com'))) AND
      PRIVILEGE = 'connect'
ORDER BY DBMS_NETWORK_ACL_UTILITY.DOMAIN_LEVEL(host) DESC, LOWER_PORT;

HOST                 LOWER_PORT UPPER_PORT ACL                  PRIVILEGE
-------------------- ---------- ---------- -------------------- ---------
www.us.mycompany.com         80         80 /sys/acls/www.xml    GRANTED
www.us.mycompany.com       3000       3999 /sys/acls/www.xml    GRANTED
www.us.mycompany.com                       /sys/acls/www.xml    GRANTED
```

User Checking Own Privileges for Domain Name Resolution

Example 4–21 shows how the user preston can check her privileges to perform domain name resolution for www.us.mycompany.com:

Example 4–23 User Checking Privileges for Domain Name Resolution

```
SELECT host, status privilege
  FROM user_network_acl_privileges
 WHERE host IN
   (SELECT * FROM
      TABLE(DBMS_NETWORK_ACL_UTILITY.DOMAINS('www.us.mycompany.com'))) AND
      privilege = 'resolve'
```

```
ORDER BY DBMS_NETWORK_ACL_UTILITY.DOMAIN_LEVEL(host) DESC;

 HOST                 PRIVILEGE
 -------------------- ---------
 www.us.mycompany.com GRANTED
```

Setting the Precedence of Multiple Users and Roles in One Access Control List

By default, Oracle Database grants or denies privileges to users and roles based on their physical position in the access control list. The first user or role listed is granted or denied privileges first, followed the second user or role, and so on. For instance, suppose the code in Example 4–17 defined one role, `ACCT_MGR`, and two users, `sebastian` and `preston`, and the access control list XML file ordered these three as follows:

```
<acl ...>
  ...
  <ace>
    <principal>ACCT_MGR</principal>
    <grant>true</grant>
    <privilege><plsql:connect/></privilege>
   </ace>
  <ace>
    <principal>SEBASTIAN</principal>
    <grant>false</grant>
    <privilege><plsql:connect/></privilege>
  </ace>
  <ace>
    <principal>PRESTON</principal>
    <grant>false</grant>
    <privilege><plsql:connect/></privilege>
  </ace>
</acl>
```

`ACCT_MGR` is granted permissions first, followed by permission denials for `sebastian` and then `preston`. However, if `sebastian` and `preston` have been granted the `ACCT_MGR` role, they still could log in, because the `ACCT_MGR` role appears first in the list.

Even though these two users were granted the `acct_mgr` role, their specific jobs do not require them to have access to the `www.mycompany.com` host. If the positions were reversed—the `acct_mgr` role listed after `sebastian` and `preston`—they would be denied the privilege of connecting to the network. To set the order of precedence of the `ACE` elements irrespective of their physical location in the `CREATE_ACL` and `ADD_PRIVILEGE` statements, you can use the `position` attribute.

For example, the following statements set the `ACE` elements in the resultant XML file in this order:

1. The `ACE` element for `sebastian` appears first.
2. The `ACE` element for `preston` appears second.
3. The `acct_mgr` role appears last.

In this case, neither of these users will be able to connect, because their grant privileges, which are set to `FALSE`, are evaluated before the `acct_mgr` role.

```
BEGIN
 DBMS_NETWORK_ACL_ADMIN.CREATE_ACL (
  acl          => 'us-mycompany-com-permissions.xml',
  description  => 'Network connection permission for ACCT_MGR and users');
```

```
  principal    => 'ACCT_MGR',
  is_grant     => TRUE,
  privilege    => 'connect');
 DBMS_NETWORK_ACL_ADMIN.ADD_PRIVILEGE (
  acl          => 'us-mycompany-com-permissions.xml'
  principal    => 'SEBASTIAN',
  is_grant     => FALSE,
  privilege    => 'connect',
  position     => 1);
 DBMS_NETWORK_ACL_ADMIN.ADD_PRIVILEGE (
  acl          => 'us-mycompany-com-permissions.xml'
  principal    => 'PRESTON',
  is_grant     => FALSE,
  privilege    => 'connect',
  position     => 2);
END;
```

Using Data Dictionary Views to Find Information About Access Control Lists

Table 4–6 lists data dictionary views that you can use to find information about existing access control lists. See *Oracle Database Reference* for more information about these views.

Table 4–6 Data Dictionary Views That Display Information about Access Control Lists

View	Description
DBA_NETWORK_ACLS	Shows the access control list assignments to the network hosts. The SELECT privilege on this view is granted to the SELECT_CATALOG_ROLE role only.
DBA_NETWORK_ACL_PRIVILEGES	Shows the network privileges defined in all access control lists that are currently assigned to network hosts. The SELECT privilege on this view is granted to the SELECT_CATALOG_ROLE role only.
USER_NETWORK_ACL_PRIVILEGES	Shows the status of the network privileges for the current user to access network hosts. The SELECT privilege on the view is granted to PUBLIC.

Finding Information About User Privileges and Roles

Table 4–7 lists data dictionary views that you can query to access information about grants of privileges and roles. See *Oracle Database Reference* for detailed information about these views.

Table 4–7 Views That Display Grant Information about Privileges and Roles

View	Description
ALL_COL_PRIVS	Describes all column object grants for which the current user or PUBLIC is the object owner, grantor, or grantee
ALL_COL_PRIVS_MADE	Lists column object grants for which the current user is object owner or grantor.
ALL_COL_PRIVS_RECD	Describes column object grants for which the current user or PUBLIC is the grantee
ALL_TAB_PRIVS	Lists the grants on objects where the user or PUBLIC is the grantee
ALL_TAB_PRIVS_MADE	Lists the all object grants made by the current user or made on the objects owned by the current user.
ALL_TAB_PRIVS_RECD	Lists object grants for which the user or PUBLIC is the grantee
DBA_COL_PRIVS	Describes all column object grants in the database

Table 4–7 (Cont.) Views That Display Grant Information about Privileges and Roles

View	Description
DBA_TAB_PRIVS	Lists all grants on all objects in the database
DBA_ROLES	This view lists all roles that exist in the database
DBA_ROLE_PRIVS	Lists roles granted to users and roles
DBA_SYS_PRIVS	Lists system privileges granted to users and roles
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.
USER_COL_PRIVS	Describes column object grants for which the current user is the object owner, grantor, or grantee
USER_COL_PRIVS_MADE	Describes column object grants for which the current user is the grantor
USER_COL_PRIVS_RECD	Describes column object grants for which the current user is the grantee
USER_ROLE_PRIVS	Lists roles granted to the current user
USER_TAB_PRIVS	Lists grants on all objects where the current user is the grantee
USER_SYS_PRIVS	Lists system privileges granted to the current user
USER_TAB_PRIVS_MADE	Lists grants on all objects owned by the current user
USER_TAB_PRIVS_RECD	Lists object grants for which the current user is the grantee
SESSION_PRIVS	Lists the privileges that are currently enabled for the user
SESSION_ROLES	Lists the roles that are currently enabled to the user

This section provides some examples of using these views. For these examples, assume the following statements were issued:

```
CREATE ROLE security_admin IDENTIFIED BY honcho2all;

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
```

See *Oracle Database Reference* for detailed information about the `DBA_SYS_PRIVS` view.

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
```

See *Oracle Database Reference* for detailed information about the `DBA_ROLE_PRIVS` view.

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
```

See *Oracle Database Reference* for detailed information about the DBA_TAB_PRIVS view.

Listing the Current Privilege Domain of Your Session

The following query lists all roles currently enabled for the issuer:

```
SELECT * FROM SESSION_ROLES;
```

If user swilliams has the security_admin role enabled and issues the previous query, then Oracle Database returns the following information:

```
ROLE
------------------------------
SECURITY_ADMIN
```

The following query lists all system privileges currently available in the security domain of the issuer, both from explicit privilege grants and from enabled roles:

```
SELECT * FROM SESSION_PRIVS;
```

If user swilliams has the security_admin role enabled and issues the previous query, then Oracle Database 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 user swilliams, then the first query would return no rows, while the second query would only return a row for the CREATE SESSION privilege grant.

See *Oracle Database Reference* for detailed information about the SESSION_ROLES view.

Listing Roles of the Database

You can use the DBA_ROLES data dictionary view 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
```

See *Oracle Database Reference* for detailed information about the DBA_ROLES view.

Listing Information About the Privilege Domains of Roles

The ROLE_ROLE_PRIVS, ROLE_SYS_PRIVS, and ROLE_TAB_PRIVS data dictionary views contain information about 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
```

See *Oracle Database Reference* for detailed information about the ROLE_ROLE_PRIVS, ROLE_SYS_PRIVS, and ROLE_TAB_PRIVS views.

5

Managing Security for Application Developers

Creating an application security policy is the first step to create a secure database application. 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 you draft security policies for database applications. It contains the following topics:

- About Application Security Policies
- Considerations for Using Application-Based Security
- Managing Application Privileges
- Creating a Secure Application Role to Control Access to Applications
- Associating Privileges with User Database Roles
- Protecting Database Objects by Using Schemas
- Managing Object Privileges in an Application
- Parameters for Enhanced Security of Database Communication

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. You can then grant the database roles to user roles or directly to specific user names.

Applications that can potentially allow unrestricted SQL statement processing (through tools such as SQL*Plus or SQL Developer) 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.

Considerations for Using Application-Based Security

Two main questions to consider when you formulate and implement application security are covered in the following sections:

- Are Application Users Also Database Users?
- Is Security Better Enforced in the Application or in the Database?

Are Application Users Also Database Users?

Where possible, you should 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. This is called the *One Big Application User* model.

Applications built in this way generally cannot use many of the intrinsic security features of the database, because the identity of the user is not known to the database.

Table 5–1 lists the features that the One Big Application User model compromises:

Table 5–1 Features Compromised by the One Big Application User Model

Oracle Database Feature	Limitations of One Big Application User Model
Auditing	A basic principle of security is accountability through auditing. If One Big Application User performs all actions in the database, then database auditing cannot hold individual users accountable for their actions. The application must implement its own auditing mechanisms to capture individual user 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	The Enterprise user management 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 Better Enforced in the Application or in the Database?

Applications, whose users are also database users, can either build security into the application, or rely on intrinsic database security mechanisms such as granular privileges, virtual private databases (fine-grained access control with application context), roles, stored procedures, and auditing (including fine-grained auditing). Oracle recommends that applications use the security enforcement mechanisms of the database as much as possible.

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, therefore, 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, that recognizes users; the application itself must enforce security measures for each user.

This approach means that each application that accesses data must reimplement security. Security becomes expensive, because organizations must implement the same security policies in multiple applications, and each new application requires an expensive reimplementation.

> **See Also:** "Potential Security Problems of Using Ad Hoc Tools" on page 4-18

Managing Application Privileges

Most database applications involve different privileges on different schema objects. Keeping track of the privileges that 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 can have several roles, each granted a specific subset of privileges that allow greater or lesser capabilities while running the application.

For example, suppose every administrative assistant uses the Vacation application to record the 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. Better yet, create a role that defines the privileges the administrative assistants have, and then grant the `VACATION` role to that role.

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 the privileges that 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 4, "Configuring Privilege and Role Authorization" for a complete discussion of creating, enabling, and disabling roles, and granting and revoking privileges
- "Finding Information About User Privileges and Roles" on page 4-57 for more information about the security uses of the `ROLE_TAB_PRIVS`, `ROLE_SYS_PRIVS`, and `DBA_ROLE_PRIVS` data dictionary views

Creating a Secure Application Role to Control Access to Applications

As explained in "Further Securing Role Privileges by Using Secure Application Roles" on page 4-19, a secure application role is a role that is only enabled through its associated PL/SQL package. This package defines the policy needed to control access to an application.

This section includes the following topics:

- Step 1: Create the Secure Application Role
- Step 2: Create a PL/SQL Package to Define the Access Policy for the Application

See Also: *Oracle Database 2 Day + Security Guide* for an example of how to create a secure application role

Step 1: Create the Secure Application Role

You create a secure application role by using the SQL statement `CREATE ROLE` with the `IDENTIFIED USING` clause. You must have the `CREATE ROLE` system privilege to execute this statement. Normally, you create this role and its associated package in the schema of the security administrator.

For example, to create a secure application role called `hr_admin` that is associated with the `sec_mgr.hr_admin` package, follow these steps:

1. Create the security application role as follows:

   ```
   CREATE ROLE hr_admin IDENTIFIED USING sec_mgr.hr_admin;
   ```

   This statement indicates the following:

   - The role `hr_admin` to be created is a secure application role.
   - The role can only be enabled by modules defined inside the PL/SQL package `system.hr_admin`. At this stage, this package does not need to exist; "Step 2: Create a PL/SQL Package to Define the Access Policy for the Application" on page 5-5 describes how to create the package.

2. Grant the security application role the privileges you would normally associate with this role.

   For example, to grant the `hr_admin` role `SELECT`, `INSERT`, `UPDATE`, and `DELETE` privileges on the `HR.EMPLOYEES` table, you enter the following statement:

   ```
   GRANT SELECT, INSERT, UPDATE, DELETE ON hr.employees TO hr_admin;
   ```

   You do not need to grant the role directly to the user. The PL/SQL package will do that for you, assuming the user passes its security policies. If your site requires that you directly grant users the role, then you *must* disable the role for that user. This is because the role needs to be initially disabled before the security policies in

the package can begin performing their checks. To disable the role for user `psmith` (assuming `psmith` was granted it in the first place), enter the following statement:

```
ALTER USER psmith DEFAULT ROLE NONE
```

Step 2: Create a PL/SQL Package to Define the Access Policy for the Application

To enable or disable the secure application role, you create the security policies of the role within a PL/SQL package. You also can create an individual function or procedure to do this, but a package lets you group a set of functions or procedure together. Normally, you create this package in the schema of the security administrator.

The package defines a simple, clear interface to a set of related procedures and types that can be accessed by SQL statements. Packages also make code more reusable and easier to maintain. The advantage here for secure application roles is that you can create a group of security policies that, used together, present a solid security strategy designed to protect your applications. For users (or potential intruders) who fail the security policies, you can add auditing checks to the package to record the failure.

The package must accomplish the following:

- It must use invoker's rights to enable the role.

  To create the package using invoker's rights, include the `AUTHID CURRENT_USER` clause in the package definition. You *must* create the package using invoker's rights in order for the package to work. Invoker's rights allow the user to have `EXECUTE` privileges on all objects that the package accesses.

  Roles that are enabled inside an invoker's right procedure remain in effect even after the procedure exits. In this case, you can have a dedicated procedure that deals with enabling the role for the rest of the session to use. Because users cannot change the security domain inside **definer's rights procedure**, secure application roles can only be enabled inside **invoker's rights procedures**.

- It must enable the application to perform 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 in which the user was authenticated, the IP address of the client, and whether the user connected through proxy authentication. To find session information for a user, you can configure an application context. See Chapter 7, "Using Application Contexts to Retrieve User Information" for details on application context.

- The application must issue a `SET_ROLE` procedure by using dynamic SQL (`DBMS_SESSION.SET ROLE`).

See Also:

- SYS_CONTEXT in the *Oracle Database SQL Language Reference* for complete details about the USERENV namespace and its predefined attributes
- *Oracle Database PL/SQL Language Reference* formation about whether to use definer's rights or invoker's rights procedures, and how to create them

For example, suppose you wanted to restrict anyone using the hr_admin role to employees who are on site (that is, using certain terminals) and between the hours of 8 a.m. and 5 p.m. As the system or security administrator, follow these steps:

1. Create the procedure as follows:

```
SQL> CREATE OR REPLACE PROCEDURE hr_admin_role_check AUTHID CURRENT_USER
  2 AS
  3 BEGIN
  4 IF (SYS_CONTEXT ('userenv','ip_address')
  5   BETWEEN '130.35.44.0' and '130.35.44.255'
  6    AND
  7   TO_CHAR (SYSDATE, 'HH24') BETWEEN 8 AND 17)
  8 THEN
  9   DBMS_SESSION.SET_ROLE('hr_admin');
 10 END IF;
 11 END;
```

   In this example:

   - **Line 4**: Validates the user by using the SYS_CONTEXT SQL function to retrieve the user session information with the USERENV namespace attributes ('userenv', *session_attribute*). The information returned by this function can indicate the way in which the user was authenticated, the IP address of the client, and whether the user was proxied. See *Oracle Database SQL Language Reference* for more information about SYS_CONTEXT.
   - **Lines 5–7**: Create a test to grant or deny access. The test restricts access to users who are on site (that is, using certain terminals) and working between the hours of 8:00 a.m. and 5:00 p.m. If the user passes this check, the hr_admin role is granted.
   - **Lines 8–9**: Assuming the user passes the test, grants the role to the user by using the DBMS_SESSION.SET_ROLE procedure.

2. Grant EXECUTE permissions for the hr_admin_role_check package to any user who was assigned it.

   For example:

```
GRANT EXECUTE ON hr_admin_role_check TO psmith;
```

Associating Privileges with User Database Roles

Ensure that users have only the privileges associated with the current database role.

Topics in this section include:

- Why Users Should Only Have the Privileges of the Current Database Role
- Using the SET ROLE Statement to Automatically Enable or Disable Roles
- Using the DBMS_SESSION.SET_ROLE Procedure to Enable or Disable Roles

- Example of Assigning Roles with Static and Dynamic SQL

Why Users Should Only Have the Privileges of the Current 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 current database role. Consider the following scenario:

- The ORDER role (for an application called Order) contains the UPDATE privilege for the INVENTORY table.
- The INVENTORY role (for an application called Inventory) contains the SELECT privilege for the INVENTORY table.
- Several order entry clerks were granted both the ORDER and INVENTORY roles.

In this scenario, an order entry clerk who was 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 for the INVENTORY application. It is an authorized action for the ORDER application.

To avoid this problem, use either the SET ROLE statement or the DBMS_SESSION.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.

Using the SET ROLE Statement to Automatically Enable or Disable Roles

Use a SET ROLE statement at the beginning of each application to automatically enable its associated role and to disable all others. 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, then the user cannot combine these privileges to perform unauthorized operations.

See Also:

- "When Do Grants and Revokes Take Effect?" on page 4-42 for information about enabling and disabling roles
- "How the SET ROLE Statement Affects Grants and Revokes" on page 4-43

Using the DBMS_SESSION.SET_ROLE Procedure to Enable or Disable Roles

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 you cannot call the DBMS_SESSION.SET_ROLE procedure from them. However, the DBMS_SESSION.SET_ROLE procedure can be called from the following:

- Anonymous PL/SQL blocks
- Invoker's rights stored procedures (except those invoked from within definer's rights procedures)

DBMS_SESSION.SET ROLE takes effect only at run 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 run time, at the same time as the `SET ROLE` statement enables roles.

Note: If you use `DBMS_SESSION.SET_ROLE` within an invoker's rights procedure, then 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:

- *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_SESSION` package
- *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_SQL` package

Example of Assigning Roles with Static and Dynamic SQL

This example shows how static and dynamic SQL affect the assignment of roles.

Follow these steps:

1. Connect to SQL*Plus as `SYSTEM` and then run the following SQL statements:

```
CONNECT system
Enter password: password
Connected.
DROP USER joe CASCADE;
CREATE USER joe IDENTIFIED BY bflstick2;
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;
```

2. Connect as user `joe` and then create a simple table.

```
CONNECT joe
Enter password: bflstick2
Connected.

CREATE TABLE finance (empno NUMBER);
GRANT SELECT ON finance TO acct;
```

3. Connect as user `SCOTT`.

```
CONNECT SCOTT
Enter password: password
Connected.
```

4. As user `SCOTT`, try creating the following procedure:

```
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 that 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.

5. Now, still as user `scott`, try creating the following procedure:

```
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;
/
```

In contrast, the `DBMS_SQL` package, which uses dynamic SQL, is not subject to the restriction in the procedure that you tried to create in Step 4. When you use this package, the security checks are performed when the procedure executes, and not when it is compiled. Therefore, this PL/SQL block is successful.

Protecting Database Objects by Using 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 includes the following topics:

- Protecting Database Objects in a Unique Schema
- Protecting Database Objects in a Shared Schema

Protecting Database Objects in a Unique Schema

You can think of most schemas as user names: the accounts that enable users to connect to a database and access the database objects. However, a *unique schema* does not allow connections to the database, but is used to contain a related set of objects. Schemas of this sort are created as typical users, and 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 a unique schema if you want to use the `CREATE SCHEMA` statement to create multiple tables and views in a single transaction.

For example, a given schema might own the schema objects for a specific application. If application users have the privileges to do so, then they can connect to the database using typical database user names 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.

Protecting Database Objects in a Shared Schema

For many applications, users do not need their own accounts or 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 to only access the `payroll` objects. To address this issue, Oracle Advanced Security provides the enterprise users, which are 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, you 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.

For more information about managing enterprise users, see *Oracle Database Enterprise User Security Administrator's Guide*.

Managing Object Privileges in an Application

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 that 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 includes the following topics:

- 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 5–2 summarizes the object privileges available for each type of object.

Table 5–2 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	No	No	No
`SELECT`	Yes	Yes[3]	Yes	No
`UPDATE`	Yes	Yes	No	No

[1] Standalone stored procedures, functions, and public package constructs

[2] Privilege that cannot be granted to a role

[3] Can also be granted for snapshots

See also "Auditing Schema Objects" on page 6-26 for detailed information about how schema objects can be audited.

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 ensure 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 5–3 lists the SQL statements permitted by the object privileges shown in Table 5–2.

Table 5–3 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 "About Privileges and Roles" on page 4-1 for a discussion of object privileges. See also "Auditing SQL Statements" on page 6-23 for detailed information about how SQL statements can be audited.

Parameters for Enhanced Security of Database Communication

Database administrators can manage security for their applications by following the procedures in this section.

- Reporting Bad Packets Received on the Database from Protocol Errors
- Terminating or Resuming Server Execution After Receiving a Bad Packet
- Configuring the Maximum Number of Authentication Attempts
- Controlling the Display of the Database Version Banner
- Configuring Banners for Unauthorized Access and Auditing User Actions

Reporting Bad Packets Received on the Database from Protocol Errors

Networking communication utilities such as Oracle Call Interface (OCI) or Two-Task Common (TTC) can generate a large disk file containing the stack trace and heap dump when the server receives a bad packet, out-of-sequence packet, or a private or an unused remote procedure call. Typically, this disk file can grow quite large. An intruder can potentially cripple a system by repeatedly sending bad packets to the server, which can result in disk flooding and denial of service. An unauthenticated client can also mount this type of attack.

You can prevent these attacks by setting the `SEC_PROTOCOL_ERROR_TRACE_ACTION` initialization parameter to one of the following values:

- `None`: Configures the server to ignore the bad packets and does not generate any trace files or log messages. Use this setting if the server availability is overwhelmingly more important than knowing that bad packets are being received.

  For example:

  ```
  SEC_PROTOCOL_ERROR_TRACE_ACTION = None
  ```

- `Trace` (default setting): Creates the trace files, but it is useful for debugging purposes, for example, when a network client is sending bad packets as a result of a bug.

  For example:

  ```
  SEC_PROTOCOL_ERROR_TRACE_ACTION = Trace
  ```

- `Log`: Writes a short, one-line message to the server trace file. This choice balances some level of auditing with system availability.

  For example:

  ```
  SEC_PROTOCOL_ERROR_TRACE_ACTION = Log
  ```

- `Alert`: Writes a short, one-line error message to the server trace file and alert log.

  For example:

  ```
  SEC_PROTOCOL_ERROR_TRACE_ACTION = Alert
  ```

Terminating or Resuming Server Execution After Receiving a Bad Packet

After Oracle Database detects a client or server protocol error, it needs to continue execution. However, this could subject the server to further bad packets, which could lead to disk flooding or denial-of-service attacks.

You can control the further execution of a server process when it is receiving bad packets from a potentially malicious client by setting the `SEC_PROTOCOL_ERROR_FURTHER_ACTION` initialization parameter to one of the following values:

- `Continue` (default setting): Continues the server execution. However, be aware that the server may be subject to further attacks.

  For example:

  ```
  SEC_PROTOCOL_ERROR_FURTHER_ACTION = Continue
  ```

- `Delay,`*m*: Delays the client *m* seconds before the server can accept the next request from the same client connection. This setting prevents malicious clients from

excessively using server resources while legitimate clients experience a degradation in performance but can continue to function.

For example:

```
SEC_PROTOCOL_ERROR_FURTHER_ACTION = Delay,3
```

- `Drop,n`: Forcefully terminates the client connection after *n* bad packets. This setting enables the server to protect itself at the expense of the client, for example, loss of a transaction. However, the client can still reconnect, and attempt the same operation again.

  For example:

  ```
  SEC_PROTOCOL_ERROR_FURTHER_ACTION = Drop,10
  ```

Configuring the Maximum Number of Authentication Attempts

With Oracle Database, a server process is first started, and then the client authenticates with this server process. An intruder could start a server process first, and then issue an unlimited number of authenticated requests with different user names and passwords in an attempt to gain access to the database.

You can limit the number of failed login attempts for application connections by setting the `SEC_MAX_FAILED_LOGIN_ATTEMPTS` initialization parameter to restrict the number of authentication attempts on a connection. After the specified number of authentication attempts fail, the database process drops the connection. By default, `SEC_MAX_FAILED_LOGIN_ATTEMPTS` is set to 10.

Remember that the `SEC_MAX_FAILED_LOGIN_ATTEMPTS` initialization parameter is designed to prevent potential intruders from attacking your applications; it does not apply to valid users. The `sqlnet.ora` `INBOUND_CONNECT_TIMEOUT` parameter and the `FAILED_LOGIN_ATTEMPTS` initialization parameter also restrict failed logins, but the difference is that these two parameters only apply to valid user accounts.

For example, to limit the maximum attempts to 5, set `SEC_MAX_FAILED_LOGIN_ATTEMPTS` as follows in the `init`*`sid`*`.ora` initialization parameter file:

```
SEC_MAX_FAILED_LOGIN_ATTEMPTS = 5
```

Controlling the Display of the Database Version Banner

Detailed product version information should not be accessible before a client connection (including an Oracle Call Interface client) is authenticated. An intruder could use the database version to find information about security vulnerabilities that may be present in the database software.

You can restrict the display of the database version banner to unauthenticated clients by setting the `SEC_RETURN_SERVER_RELEASE_BANNER` initialization parameter in the `init`*`sid`*`.ora` initialization parameter file to either `YES` or `NO`. By default, `SEC_RETURN_SERVER_RELEASE_BANNER` is set to `NO`.

For example, if you set it to `YES`, the Oracle Database displays the full correct database version:

```
Oracle Database 11g Enterprise Edition Release 11.1.0.0 - Production
```

In the future, if you install Oracle Database 11.2.0.2, for example, it will display the following banner:

```
Oracle Database 11g Enterprise Edition Release 11.2.0.2 - Production
```

However, if in that same release, you set it to `NO`, then Oracle Database restricts the banner to display the following fixed text starting with Release 11.1:

```
Oracle Database 11g Release 11.1.0.0.0 - Production
```

Configuring Banners for Unauthorized Access and Auditing User Actions

You should create and configure banners to warn users against unauthorized access and possible auditing of user actions. The notices are available to the client application when it logs into the database.

To configure these banners to display, set the following `sqlnet.ora` parameters on the database server side to point to a text file that contains the banner information:

- `SEC_USER_UNAUTHORIZED_ACCESS_BANNER`. For example:

  ```
  SEC_USER_UNAUTHORIZED_ACCESS_BANNER = /opt/Oracle/11g/dbs/unauthaccess.txt
  ```

- `SEC_USER_AUDIT_ACTION_BANNER`. For example:

  ```
  SEC_USER_AUDIT_ACTION_BANNER = /opt/Oracle/11g/dbs/auditactions.txt
  ```

By default, these parameters are not set.

After you set the these parameters, the Oracle Call Interface application needs to the use the appropriate OCI APIs to retrieve these banners and present them to the end user.

6

Configuring Auditing

Auditing is about accountability, and is frequently performed to protect and preserve privacy for the information stored in databases.

This chapter discusses the following topics:

- About Auditing
- Creating a Record of Audited Activity
- Managing the Database Audit Trail
- Using Default Auditing for Security-Relevant SQL Statements and Privileges
- Using Standard Auditing to Monitor General Activities
- Auditing Administrative Users
- Using Triggers to Record Customized Standard Auditing Information
- Using Fine-Grained Auditing to Monitor Specific Activities
- Archiving the Standard and Fine-Grained Audit Trails
- Finding Information About Audited Activities

See "Guidelines for Auditing" on page 10-15 for general guidelines to follow when deciding how to audit your system.

About Auditing

Auditing is the monitoring and recording of selected user database actions. You can base auditing on individual actions, such as the type of SQL statement executed, or on combinations of factors that can include the user name, application, time, and so on. You can create security policies to trigger auditing when someone accesses or alters specified objects in an Oracle database, including the contents within a specified object.

This section includes the following topics:

- Why Is Auditing Used?
- What Is Audited?

See also *Oracle Audit Vault Administrator's Guide* for information about Oracle Audit Vault, which provides advanced auditing features.

Why Is Auditing Used?

You typically use auditing to perform the following activities:

- **Enable future accountability for current actions.** These include actions taken in a particular schema, table, or row, or affecting specific content.
- **Deter users (or others, such as intruders) from inappropriate actions based on their accountability.**
- **Investigate suspicious activity.** For example, if a user is deleting data from tables, then a 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 of the actions of an unauthorized user.** For example, an unauthorized user could be changing or deleting data, or a 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 generate audit records, then you will know the other security controls are not properly implemented.
- **Address auditing requirements for compliance.** Regulations such as the following have common auditing-related requirements:
  - Sarbanes-Oxley Act
  - Health Insurance Portability and Accountability Act (HIPAA)
  - International Convergence of Capital Measurement and Capital Standards: a Revised Framework (Basel II)
  - Japan Privacy Law
  - European Union Directive on Privacy and Electronic Communications

What Is Audited?

You can design auditing to be focused or broad, enabling you to audit the following:

- 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 6–1 describes the different Oracle Database auditing mechanisms. Each entry in the first column is a link to a more extensive discussion of that particular method.

Table 6–1 Auditing Types and Descriptions

Type of Auditing (Link to Discussion)	Description
Auditing SQL Statements	Audits SQL statements by type of statement. 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. You can audit a set of default SQL statements that Oracle recommends be audited. See "Using Default Auditing for Security-Relevant SQL Statements and Privileges" on page 6-10 for more information.
Auditing Privileges	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. You can audit a set of default privileges that Oracle recommends be audited. See "Using Default Auditing for Security-Relevant SQL Statements and Privileges" on page 6-10 for more information.
Auditing Schema Objects	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.
Auditing SQL Statements and Privileges in a Multitier Environment	Audits actions taken on behalf of the client by a middle-tier application.
Auditing Network Activity	Audits unexpected errors in network protocol or internal errors in the network layer.
Using Fine-Grained Auditing to Monitor Specific Activities	Audits at the most granular level, data access, and actions based on content, using Boolean measures, such as `value > 1,000,000`. Enables auditing based on access to or changes in a column.

Creating a Record of Audited Activity

This section explains the different types of audit records that you can create. It explores the following topics:

- Where Are Audited Activities Recorded?
- Activities That Are Always Audited
- Activities That Are Always Recorded in the Operating System and Syslog Audit Trails

See Also:

- "Managing the Standard Audit Trail" on page 6-12
- "Managing the Operating System Audit Trail" on page 6-20
- "Deciding Whether to Use the Database or Operating System Audit Trail" on page 6-22

Where Are Audited Activities Recorded?

Audit records include information about 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**.

There are three general types of auditing:

- **Standard auditing.** Use standard auditing for SQL statements, privileges, schemas, objects, and network and multitier activity. Standard audit records are written to either of the following locations:
  - **SYS.AUD$ system table.** You can view the contents of this table by querying the `DBA_AUDIT_TRAIL` data dictionary view, or the `DBA_COMMON_AUDIT_TRAIL` view, which combines standard and fine-grained audit log records.
  - **Operating system files.** In addition to writing the audit trail in operating system file format, you can write it in XML format as well. See "Managing the Operating System Audit Trail" on page 6-20.

  To control how standard audit trail records are written, you set the `AUDIT_TRAIL` initialization parameter. Table 6–3 on page 6-14 describes the `AUDIT_TRAIL` parameter settings.

  See "Using Standard Auditing to Monitor General Activities" on page 6-11 for more information about standard auditing.

- **Fine-grained auditing.** Use fine-grained auditing to monitor specific activities, such as actions on a database table or times that activities occur. Fine-grained audit records are written to the `SYS.FGA_LOG$` system table. To view the contents of this table, query the `DBA_FGA_AUDIT_TRAIL` data dictionary view or the `DBA_COMMON_AUDIT_TRAIL` view.

  To control how fine-grained audit trail records are written, set the `audit_trail` parameter in the `DBMS_FGA.ADD_POLICY` procedure. The `audit_trail` parameter writes the records to either the `FGA_LOG$` system table or to an XML operating system file.

  See "Using Fine-Grained Auditing to Monitor Specific Activities" on page 6-38 for more information.

- **Administrator auditing.** On UNIX systems, you can monitor the activities of system administrators (user `SYS`, and users connecting with the `SYSDBA` or `SYSOPER` privilege) by using the syslog audit trail. Syslog is another destination audit trail, similar to operating system files, XML format files, and database tables. On Windows, these activities are recorded in the Windows Event Log, along with other types of activities.

  For both UNIX and Windows, to control how administrator audit files are written, you set the following initialization parameters:
  - **AUDIT_SYS_OPERATIONS parameter.** Enables or disables administrator auditing. Setting it to `TRUE` records system administrator activities in the operating system file that contains the audit trail.
  - **AUDIT_SYSLOG_LEVEL parameter.** When the `AUDIT_TRAIL` parameter is set to `OS`, writes `SYS` and standard operating system audit records to the system audit log using the `syslog` utility.

  See "Auditing Administrative Users" on page 6-33 for more information.

See Also: "Finding Information About Audited Activities" on page 6-47 for how you can use data dictionary views that capture audited information to find suspicious behavior

Activities That Are Always Audited

Regardless of whether database auditing is enabled, Oracle Database *always* audits certain database-related operations and writes them to the operating system audit file. The operating system audit file captures the complete archived messages for these types of activities. You can set the location of this file by using the `AUDIT_FILE_DEST` initialization parameter, which is described in "Specifying a Directory for the Operating System Audit Trail" on page 6-21. This is called **mandatory auditing**, and it includes the following operations:

- **Administrative privilege connections to the database instance.** An audit record is generated that lists the operating system user connecting to Oracle Database as `SYSOPER` or `SYSDBA`. This provides for accountability of users with administrative privileges. You can fully audit these users, as explained in "Auditing Administrative Users" on page 6-33.
- **Database startup.** An audit record is generated that lists the operating system user starting the instance, the user terminal identifier, and 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 the startup has successfully completed.
- **Database shutdown.** An audit record is generated that lists the operating system user shutting down the instance, the user terminal identifier, and the date and time stamp.

See Also:

- "Activities That Are Always Recorded in the Operating System and Syslog Audit Trails" on page 6-5 for more information about the operating system audit file
- "Activities That Are Always Recorded in the Standard Audit Trail" on page 6-13
- "Activities That Are Always Recorded in Fine-Grained Auditing" on page 6-40

Activities That Are Always Recorded in the Operating System and Syslog Audit Trails

Some database-related actions are always recorded into the operating system audit trail and for UNIX systems, the syslog audit trail, regardless of whether database auditing is enabled. (The syslog audit trail is described in "Using the Syslog Audit Trail to Audit System Administrators on UNIX Systems" on page 6-35.) The fact that these records are always created is sometimes referred to as **mandatory auditing**. (See "Activities That Are Always Recorded in the Standard Audit Trail" on page 6-13 for more information.)

On operating systems that do not make an audit trail accessible to Oracle Database, these audit trail records are placed in an Oracle Database audit trail file in the same directory as background process trace files, and in a similar format.

See Also: Operating system-specific Oracle Database documentation for more information about the operating system and syslog audit trail

Managing the Database Audit Trail

This section contains the following topics:

- Database Audit Trail Contents
- Example of Auditing Changes to the SYS.AUD$ Table

Database Audit Trail Contents

The database audit trail is a pair of tables, `SYS.AUD$` and `SYS.FGA_LOGS$`, in the `SYS` schema of each Oracle Database data dictionary. It records both standard and fine-grained audit activities. Several predefined views are provided to help you use the information in this table, such as `DBA_AUDIT_TRAIL`.

The database audit trail record contains different types of information, depending on the events audited and the auditing options set. Table 6–2 contains a partial list in the that shows columns that always appear in the audit trail. If the data they represent is available, then that data populates the corresponding column. For certain columns, this list has the column name as it displays in the audit record, shown inside parentheses. The operating system audit trail has only those columns that have Yes in the corresponding column.

Table 6–2 Audit Trail Record Data

Data Populated in Database Audit Trail	In Operating System Audit Trail?
(*) Bind values used for the SQL statement, if any	Footnote 1
(*) SQL text (the SQL text that triggered the auditing)	Footnote 1
Completion code of the operation	Yes
Database user name (`DATABASE USER`)	Yes
Date and time stamp in UTC (Coordinated Universal Time) format	No
Distinguished name	Yes
Global User unique ID	No
Instance number	No
Name of the schema object accessed	Yes
Operating system login user name (`CLIENT USER`)	Yes
Operation performed or attempted (`ACTION`)	Yes
Process number	Footnote 2
Proxy Session audit ID	No
SCN (system change number) for the SQL statement	No
Session identifier	Yes
System privileges used (`PRIVILEGE`)	Yes
Terminal identifier	Yes
Transaction ID	No

Footnote 1: Columns with an asterisk (*) in Table 6–2 appear in the audit records only if you have set the `AUDIT_TRAIL` initialization parameter to `DB, EXTENDED` or `XML, EXTENDED`. Also, for an array, the values recorded are only the last set of bind values.

Footnote 2: Process number is populated as `ProcessId` on UNIX systems. On Windows systems, the label is `ProcessId:ThreadId` (or `ProcessId` if it is not running as a thread).

Note: If the `AUDIT_TRAIL` initialization parameter is set to `XML` or `XML, EXTENDED`, then Oracle Database sends standard audit records to operating system files in XML format. Because XML is a standard document format, many utilities are available to parse and analyze XML data.

If the database destination for audit records becomes full or unavailable, and, therefore, unable to accept new records, then an audited action cannot complete. Instead, Oracle Database generates an error message and does not audit the action. In most cases, using an operating system log as the audit trail destination allows the action to complete.

See Also:

- "Keeping Audited Information Manageable" on page 10-15
- "Controlling the Growth and Size of the Standard Audit Trail" on page 6-17

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.

The `DBA_COMMON_AUDIT_TRAIL` view combines standard and fine-grained audit log records.

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:

- "Using Fine-Grained Auditing to Monitor Specific Activities" on page 6-38 for more information about methods of fine-grained auditing
- *Oracle Database Administrator's Guide* for information about auditing table changes by using Flashback Transaction Query
- Flashback entries in the table of system privileges listed in the `GRANT` SQL statement section of *Oracle Database SQL Language Reference*

Note: To read from `FLASHBACK_TRANSACTION_TABLE` or `V$LOGMNR_CONTENTS`, you need to have the `SELECT ANY TRANSACTION` system privilege.

Example of Auditing Changes to the SYS.AUD$ Table

This example demonstrates the auditing of changes made to the SYS.AUD$ table.

Follow these steps:

- Step 1: Create a User for This Example
- Step 2: Enable Auditing and Truncate the SYS.AUD$ Table
- Step 3: Perform and Audit Actions by the User
- Step 4: Remove the Components for This Example

Step 1: Create a User for This Example

1. Log in to SQL*Plus as user SYS and connect with the AS SYSDBA privilege.

```
sqlplus "SYS/AS SYSDBA"
Enter password: password
Connected.
```

2. Create the following user:

```
GRANT CREATE SESSION TO smith IDENTIFIED BY test2day;
```

3. Grant user smith the following privileges:

```
GRANT SELECT, INSERT, UPDATE, DELETE ON AUD$ TO smith;
GRANT SELECT ON DBA_AUDIT_TRAIL TO smith;
```

4. Enter the following commands to format the output in later steps of this procedure:

```
col username format a10
col action_name format a13
col owner format a7
col obj_name format a10
```

See *SQL*Plus User's Guide and Reference* for more information about formatting commands in SQL*Plus.

Step 2: Enable Auditing and Truncate the SYS.AUD$ Table

1. Enable auditing on the SYS.AUD$ table.

```
AUDIT SELECT ON AUD$ BY ACCESS;
```

The BY ACCESS clause enables the audit operation to write one record each time the SYS.AUD$ table is accessed.

2. Truncate the SYS.AUD$ table.

```
TRUNCATE TABLE AUD$;
```

The TRUNCATE TABLE statement purges all records from the SYS.AUD$ table, and removes the extents allocated for the table. If a table is very large, using TRUNCATE TABLE is faster than using DELETE to remove rows from a table.

Step 3: Perform and Audit Actions by the User

1. Connect as user smith.

```
CONNECT smith
Enter password: test2day
```

```
Connected.
```

2. Enter the following statement:

```
SELECT COUNT(*) FROM SYS.AUD$;

 COUNT(*)
---------
        1
```

3. Enter the following SELECT statement:

```
SELECT USERNAME, ACTION_NAME, OWNER, OBJ_NAME FROM DBA_AUDIT_TRAIL
WHERE ACTION NOT IN (100, 101);

USERNAME   ACTION_NAME   OWNER   OBJ_NAME
---------- ------------- ------- ----------
SMITH      SELECT        SYS     AUD$
```

   This SELECT statement shows the SELECT statement user smith performed on the DBA_AUDIT_TRAIL view, which lists the contents of the SYS.AUD$ table.

4. Perform the following UPDATE statement on the SYS.AUD$ table:

```
UPDATE SYS.AUD$ SET USERID = 0;

3 rows updated.
```

5. Repeat the SELECT statement from Step 3 and note the changed output:

```
SELECT USERNAME, ACTION_NAME, OWNER, OBJ_NAME FROM DBA_AUDIT_TRAIL
WHERE ACTION NOT IN (100, 101);

USERNAME   ACTION_NAME   OWNER   OBJ_NAME
---------- ------------- ------- ----------
0          SELECT        SYS     AUD$
0          SELECT        SYS     AUD$
SMITH      UPDATE        SYS     AUD$
```

   As you can see, the SYS.AUD$ table is recording each action performed by user smith.

6. Delete the rows from the SYS.AUD$ table.

```
DELETE FROM SYS.AUD$;

4 rows deleted.
```

7. Repeat the SELECT statement from Step 3 and note the changed output:

```
SELECT USERNAME, ACTION_NAME, OWNER, OBJ_NAME FROM DBA_AUDIT_TRAIL
WHERE ACTION NOT IN (100, 101);

USERNAME   ACTION_NAME   OWNER   OBJ_NAME
---------- ------------- ------- ----------
SMITH      UPDATE        SYS     AUD$
SMITH      DELETE        SYS     AUD$
```

Step 4: Remove the Components for This Example

1. Connect as user SYS with the AS SYSDBA privilege.

```
CONNECT SYS/AS SYSDBA
Enter password: password
```

2. Remove auditing from the `SYS.AUD$` table.

```
NOAUDIT SELECT, INSERT, UPDATE, DELETE ON AUD$;
```

3. Drop user `smith`.

```
DROP USER smith;
```

Using Default Auditing for Security-Relevant SQL Statements and Privileges

When you create a new database or modify an existing database, you can use the Security Settings window in Database Configuration Assistant (DBCA) to enable or disable the default security settings. Oracle recommends that you enable these settings. When you enable the default security settings, Oracle Database audits some of the security-relevant SQL statements and privileges. It also sets the `AUDIT_TRAIL` initialization parameter to `DB`.

Oracle Database audits the `AUDIT ROLE` SQL statement by default. The privileges that are audited by default are as follows:

ALTER ANY PROCEDURE	CREATE ANY JOB	DROP ANY TABLE
ALTER ANY TABLE	CREATE ANY LIBRARY	DROP PROFILE
ALTER DATABASE	CREATE ANY PROCEDURE	DROP USER
ALTER PROFILE	CREATE ANY TABLE	EXEMPT ACCESS POLICY
AUDIT ROLE BY ACCESS	CREATE EXTERNAL JOB	GRANT ANY OBJECT PRIVILEGE
ALTER SYSTEM	CREATE PUBLIC DATABASE LINK	GRANT ANY PRIVILEGE
ALTER USER	CREATE SESSION	GRANT ANY ROLE
AUDIT SYSTEM	CREATE USER	
AUDIT SYSTEM BY ACCESS	DROP ANY PROCEDURE	

Oracle Database also audits all privileges and statements `BY ACCESS` in one statement.

If you are concerned that the auditing of these statements and privileges will adversely affect your applications, you can disable this auditing in the Security Settings window of Database Configuration Assistant. You also should be aware that auditing may adversely affect performance. If you choose the Oracle 10*g* Release 10.2 default for auditing, auditing will be disabled.

Oracle recommends that you enable auditing by default. Auditing is an effective method of enforcing strong internal controls so that your site can meet its regulatory compliance requirements, as defined in the Sarbanes-Oxley Act. This enables you to monitor business operations, and find any activities that may deviate from company policy. Doing so translates into tightly controlled access to your database and the application software, ensuring that patches are applied on schedule and preventing ad hoc changes. By enabling auditing by default, you can generate an audit record for audit and compliance personnel. However, be aware that auditing may affect database performance. Change the audit settings based on your enterprise security and compliance needs.

To individually control the auditing of SQL statements and privileges, use the AUDIT and NOAUDIT statements. For more information, see "Auditing SQL Statements" on page 6-23 and "Auditing Privileges" on page 6-25.

See Also:

- *Oracle Database 2 Day + Security Guide* for instructions about using Database Configuration Assistant to enable default auditing
- *Oracle Database Reference* for detailed information about the AUDIT_TRAIL initialization parameter.
- *Oracle Database SQL Language Reference* for detailed information about the SQL statements described in this section

Using Standard Auditing to Monitor General Activities

This section describes how to monitor general activities, such as SQL statements or privileges, by using standard auditing. It contains the following topics:

- About Standard Auditing
- Who Can Perform Standard Auditing?
- Managing the Standard Audit Trail
- Managing the Operating System Audit Trail
- Deciding Whether to Use the Database or Operating System Audit Trail
- Auditing SQL Statements
- Auditing Privileges
- Auditing SQL Statements and Privileges in a Multitier Environment
- Auditing Schema Objects
- Focusing Statement, Privilege, and Schema Auditing
- Auditing Network Activity

See Also:

- "Auditing Administrative Users" on page 6-33 to learn how to use standard auditing to audit SYS users
- "Using Triggers to Record Customized Standard Auditing Information" on page 6-37 to learn how to create triggers that perform standard auditing

About Standard Auditing

In standard auditing, you audit SQL statements, privileges, schema objects, and network activity. You accomplish this by using the AUDIT SQL statement to enable the auditing, and NOAUDIT to disable it. Alternatively, you can use Enterprise Manager Database Control to enable or disable standard auditing.

Who Can Perform Standard Auditing?

Any user can audit the objects in his or her own schema, by using the AUDIT statement. To disable auditing of an object, the user can use the NOAUDIT statement. No additional privileges are needed to perform this task. Users can run AUDIT

statements to set auditing options regardless of the AUDIT_TRAIL parameter setting. If auditing has been disabled, the next time it is enabled, Oracle Database will record the auditing activities set by the AUDIT statements. "Enabling or Disabling the Standard Audit Trail" on page 6-13 explains how to enable standard auditing.

Note the following:

- To audit objects in another schema, the user needs to have the AUDIT ANY system privilege.
- To audit system privileges, the user must have the AUDIT SYSTEM privilege.
- If the O7_DICTIONARY_ACCESSIBILITY initialization parameter has been set to FALSE (the default), then only users who have the SYSDBA privilege can audit objects in the SYS schema. For greater security, set the O7_DICTIONARY_ACCESSIBILITY parameter to FALSE.

See Also:

- GRANT in *Oracle Database SQL Language Reference* for a listing of available system and object privileges
- AUDIT in *Oracle Database SQL Language Reference* for a full listing of audit options

Managing the Standard Audit Trail

Oracle Database writes the standard audit records to either the SYS.AUD$ table (accessible by querying the DBA_AUDIT_TRAIL view) or to an operating system file.

The following sections explain how to manage the standard audit trail:

- When Are Standard Audit Records Created?
- Activities That Are Always Recorded in the Standard Audit Trail
- Enabling or Disabling the Standard Audit Trail
- Enabling Standard Auditing Options
- Disabling Standard Audit Options
- Controlling the Growth and Size of the Standard Audit Trail
- Protecting the Standard Audit Trail
- Auditing the Standard Audit Trail

See also *Oracle Database 2 Day + Security Guide* an example of how to use standard auditing.

When Are Standard Audit Records Created?

You, as the security administrator, enable or disable standard auditing for the entire database. If it is disabled, then 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 you enable database auditing, then individual audit options become effective. Any authorized database user can set these audit options for the database objects he or she owns. It is important that users exercise caution when selecting objects to audit

because auditing too many objects can fill up the SYSTEM tablespace, which impacts performance.

When auditing is enabled in the database and an action set to be audited occurs, Oracle Database generates an audit record during or after the execution phase of the SQL statement. Oracle Database individually audits SQL statements inside PL/SQL program units, as necessary, when the program unit is run.

The generation and insertion of an audit trail record is independent of a user transaction being committed. That is, even if a user 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 take effect in that session. The modified statement or privilege audit options take effect only when the current session ends and a new session is created.

In contrast, changes to schema object audit options become immediately effective for current sessions.

Note: AUDIT_SYS_OPERATIONS does not depend on the standard auditing parameter, AUDIT_TRAIL. Storing the auditing records in a location separate from the usual database audit trail in the SYS schema provides greater auditing security. To specify a location for the AUDIT_SYS_OPERATIONS audit records, set the AUDIT_FILE_DEST initialization parameter. By default, Oracle Database stores these audit records in the $ORACLE_HOME/rdbms/audit directory for UNIX systems and in the Event Viewer log file for Microsoft Windows systems.

See Also: *Oracle Database Concepts* for information about the different phases of SQL statement processing and shared SQL

Activities That Are Always Recorded in the Standard Audit Trail

Oracle Database records all data manipulation language (DML) statements, such as INSERT, UPDATE, MERGE, and DELETE on SYS.AUD$ and SYS.FGA_LOGS$ in the standard audit trail table SYS.AUD$. It performs the audit even if auditing is not enabled for the table in which these activities occur. You can check these activities by running the DBA_AUDIT_TRAIL and DBA_COMMON_AUDIT_TRAIL views.

Enabling or Disabling the Standard Audit Trail

Before you can use standard auditing, you need to enable the standard audit trail by setting the AUDIT_TRAIL initialization parameter. This setting determines whether to create the audit trail in the database audit trail, write the audit activities to an operating system file, or to disable auditing.

To enable or disable the standard audit trail, log in to SQL*Plus (or SQL Developer) with administrative privileges, and use the ALTER SYSTEM statement. Afterwards, you need to restart the database instance.

If you want to check the current value of the AUDIT_TRAIL parameter, use the SHOW PARAMETERS statement in SQL*Plus.

Example 6–1 shows how to run the SHOW PARAMETERS statement.

Example 6–1 Checking the Current Value of the AUDIT_TRAIL Initialization Parameter

```
SHOW PARAMETERS AUDIT_TRAIL

NAME                                 TYPE        VALUE
------------------------------------ ----------- -------
audit_trail                          string      DB
```

Example 6–2 shows how to log onto SQL*Plus, enable the standard audit trail, and then restart the database instance.

Example 6–2 Enabling the Standard Audit Trail

```
sqlplus "SYS/AS SYSDBA"
Enter password: password
Connected.
SQL> ALTER SYSTEM SET AUDIT_TRAIL=DB, EXTENDED SCOPE=SPFILE;
System altered.
SQL> CONNECT SYS/AS SYSOPER
Enter password: password
Connected.
SQL> SHUTDOWN;
Database closed.
Database dismounted.
ORACLE instance shut down.
SQL> STARTUP;
ORACLE instance started.
```

This examples uses the `SCOPE` clause because the database instance had been started using a server parameter file (SPFILE). Starting the database with a server parameter file is the preferred way of starting a database instance. See *Oracle Database Administrator's Guide* for information about creating configuring server parameter files.

Table 6–3 lists the settings you can use for the `AUDIT_TRAIL` initialization parameter.

Table 6–3 AUDIT_TRAIL Parameter Settings

AUDIT_TRAIL Value	Description
`DB`	Enables database auditing and directs audit records to the database audit trail (the `SYS.AUD$` table), except for records that are always written to the operating system audit trail. Use this setting for a general database for manageability. (This value is the default.) If the database was started in read-only mode with `AUDIT_TRAIL` set to `DB`, then Oracle Database internally sets `AUDIT_TRAIL` to `OS`. Check the alert log for details. See also "Managing the Database Audit Trail" on page 6-6.

Table 6–3 (Cont.) AUDIT_TRAIL Parameter Settings

AUDIT_TRAIL Value	Description
DB, EXTENDED	Performs all actions of AUDIT_TRAIL=DB, and also populates the SQL bind and SQL text CLOB-type columns of the SYS.AUD$ table, when available. These two columns are populated only when this parameter is specified. DB, EXTENDED captures the SQL triggered by an audit. You can capture both the SQL statement that caused the audit, and any associated bind variables. However, be aware that you can only capture data from scalar column types, such as integers. You cannot capture data from object columns, LOBS, CLOBS, BLOBS, or user-defined column types. If the database was started in read-only mode with AUDIT_TRAIL set to DB, EXTENDED, then Oracle Database internally sets AUDIT_TRAIL to OS. Check the alert log for details.
OS	Enables database auditing, and directs all audit records to an operating system file. If you are using an ultra-secure database configuration, Oracle recommends that you use this setting because it reduces the likelihood of a Denial of Service (DoS) attack. This setting also makes it easier to secure the audit trail. If the auditor is distinct from the database administrator, you must use the OS setting. Any auditing information stored in the database can be viewed and modified by the DBA. To specify the location of the operating system audit record file, set the AUDIT_FILE_DEST initialization parameter. The default directory is $ORACLE_BASE/admin/$DB_UNIQUE_NAME/adump. See also "Managing the Operating System Audit Trail" on page 6-20.
XML	Writes to the operating system audit record file in XML format. Records all elements of the AuditRecord node except Sql_Text and Sql_Bind to the operating system XML audit file.
XML, EXTENDED	Performs all actions of AUDIT_TRAIL=XML, and populates the SQL bind and SQL text CLOB-type columns of the SYS.AUD$ table, wherever possible. These columns are populated only when this parameter is specified.
NONE	Disables standard auditing.

Note the following:

- **You do not need to restart the database if you change the object auditing.** You only need to restart the database if you made a universal change, such as turning off *all* auditing.
- **You do not need to set AUDIT_TRAIL to enable either fine-grained auditing or SYS auditing.** For fine-grained auditing, you add and remove fine-grained audit policies as necessary, applying them to the specific operations or objects you want to monitor. You can use the AUDIT_SYS_OPERATIONS parameter to enable and disable SYS auditing.

Enabling Standard Auditing Options

To use standard auditing, use the AUDIT SQL statement. Table 6–4 lists the categories in which you can use the AUDIT statement.

Table 6–4 Standard Auditing Levels and Their Effects

Level	Effect
Statement	Audits 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 `HR.EMPLOYEES` table.
Network	Audits unexpected errors in network protocol or internal errors in the network layer.

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.

Example 6–3 shows how to use the `BY` clause to audit statements by users `jward` and `jane`.

Example 6–3 AUDIT Statement Using BY Clause

```
AUDIT SELECT TABLE, UPDATE TABLE, DELETE TABLE
 BY jward, jane;
```

When setting auditing options, you can also specify the following conditions for auditing:

- `BY SESSION/BY ACCESS`

  `BY SESSION` writes a single record for all SQL statements of the same type issued in the same session. `BY ACCESS` writes one record for each access.

  For example:

  ```
  AUDIT SELECT TABLE, UPDATE TABLE, DELETE TABLE
   BY SESSION;
  ```

  Note: If `AUDIT_TRAIL` is set to `OS` or `AUDIT_TRAIL` is set to `XML`, then you can still write multiple records to the audit trail when `BY SESSION` is specified. Multiple records occur because, while Oracle Database can write to the operating system file, the database cannot read it to detect that an audit entry already exists for the action.

- `WHENEVER SUCCESSFUL/WHENEVER NOT SUCCESSFUL`

  `WHENEVER SUCCESSFUL` audits only statements that succeed. `WHENEVER NOT SUCCESSFUL` audits only statements that fail or result in errors.

  For example:

```
AUDIT SELECT UPDATE TABLE, DELETE TABLE
 WHENEVER NOT SUCCESSFUL;
```

Subsequent sections discuss the implications of your choice of auditing options and the specification of AUDIT statement clauses.

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. To turn auditing on and control whether Oracle Database generates audit records based on the audit options currently set, set the initialization parameter AUDIT_TRAIL as described in "Enabling or Disabling the Standard Audit Trail" on page 6-13.

See Also: *Oracle Database SQL Language Reference* for a description of the AUDIT statement syntax

Disabling Standard Audit Options

The NOAUDIT statement turns off the audit options. 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, then the auditing option is disabled entirely, for both successful and unsuccessful cases.

The NOAUDIT statement does not support the BY SESSION/BY ACCESS option pair. You can turn off audit options, no matter how they were turned on, by using an appropriate NOAUDIT statement.

Caution: The NOAUDIT statement only specifies auditing options. It does not disable auditing. To turn auditing off and stop Oracle Database from generating audit records, set the AUDIT_TRAIL initialization parameter as described in "Enabling or Disabling the Standard Audit Trail" on page 6-13.

See Also: *Oracle Database SQL Language Reference* for a description of the NOAUDIT statement syntax

Controlling the Growth and Size of the Standard Audit Trail

If the audit trail is full and no more audit records can be inserted, then audited statements cannot be successfully run until you purge the audit trail. Warnings are returned to all users who issue audited statements. Therefore, you must control the growth and size of the audit trail.

When auditing is enabled and audit records are being generated, the audit trail file increases 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, then audit records are generated and stored in the audit trail. If it is disabled, then audit records are not generated.
- **Be selective about the audit options that are turned on.** If more selective auditing is performed, then 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 accomplished in two ways:
  - A security administrator owns all objects and never grants the `AUDIT ANY` system privilege 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). The security administrator is the only user granted the `AUDIT ANY` system privilege.

  In both scenarios, a security administrator controls entirely object auditing.

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:** Operating system-specific Oracle Database documentation for more information about managing the operating system audit trail when directing audit records to that location

Archiving Standard Audit Trail Information If you need to archive audit trail information for historical purposes, then you can copy the relevant records to a typical database table (for example, using `INSERT INTO` *`table`* `SELECT ... FROM SYS.AUD$ ...`), or export the audit trail table to an operating system file. "Archiving the Standard and Fine-Grained Audit Trails" on page 6-47 explains how to use Oracle Data Pump Export to export the `SYS.AUD$` table to an operating system file.

Purging the Standard Audit Trail After auditing is enabled for some time, you should periodically purge (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:** Oracle Database audits all deletions from the audit trail, without exception. See "Auditing the Standard Audit Trail" on page 6-19 and "Auditing Administrative Users" on page 6-33.

Only the user SYS, a user who has the DELETE ANY TABLE privilege, or a user to whom SYS granted the DELETE privilege on SYS.AUD$ can delete records from the database audit trail.

> **Note:** If the audit trail is full and connections are being audited (that is, if the SESSION option is set), then 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.

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, then you can reduce the space allocated to the database audit trail by following these steps:

1. If you want to save information currently in the audit trail, then copy it to another database table, or export it by using the Oracle Data Pump Export.

   See "Archiving the Standard and Fine-Grained Audit Trails" on page 6-47 for an example of using Oracle Data Pump Export.

2. Connect as a user with administrator privileges.
3. Truncate SYS.AUD$ using the TRUNCATE TABLE statement.
4. Reload archived audit trail records generated in Step 1.

The new version of SYS.AUD$ is allocated only as many extents as are necessary to maintain 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, you should protect the integrity of the audit trail 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 who has connected with administrator privileges. Remember that administrators are also audited for unauthorized use. See "Auditing Administrative Users" on page 6-33 for more information.

Auditing the Standard Audit Trail

If an application needs to give SYS.AUD$ access to regular users (non-SYSDBA users), remember that DML statements such as INSERT, UPDATE, MERGE, and DELETE are always audited and recorded in the SYS.AUD$ table. You can check these activities by running the DBA_AUDIT_TRAIL and DBA_COMMON_AUDIT_TRAIL views.

If a typical user has SELECT, UPDATE, INSERT, and DELETE privileges on SYS.AUD$ and executes a SELECT operation, then 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 for example row 1.

If a user later tries to DELETE this row from SYS.AUD$, then the DELETE succeeds, because the user has the privilege to perform this action. However, this DELETE action on SYS.AUD$ 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 log file for an illustrative test case appears in "Example of Auditing Changes to the SYS.AUD$ Table" on page 6-8.

Note: DELETE, INSERT, UPDATE, and MERGE operations on SYS.AUD$ table are always audited, and such audit records are not allowed to be deleted.

Managing the Operating System Audit Trail

As an alternative to creating standard audit records in the DBA_AUDIT_TRAIL (SYS.AUD$ table), you can create standard audit records in operating system files. This section describes the following topics:

- Contents of the Operating System Trail
- How the Operating System Audit Trail Works
- Specifying a Directory for the Operating System Audit Trail
- Decoding Operating System Audit Trial Records

See Also:

- "Using the Syslog Audit Trail to Audit System Administrators on UNIX Systems" on page 6-35
- "Activities That Are Always Recorded in the Operating System and Syslog Audit Trails" on page 6-5

Contents of the Operating System Trail

The operating system file that contains the audit trail can include any of the following data:

- Audit records generated by the operating system
- Database audit trail records
- Database actions that are always audited
- Audit records for administrative users (SYS)

How the Operating System Audit Trail Works

You can direct audit trail records to an operating system audit trail if the operating system makes an audit trail available to Oracle Database. If not, then Oracle Database writes the audit records to a file outside the database. The target directory varies by platform. On most UNIX platforms, it is $ORACLE_BASE/admin/$DB_UNIQUE_NAME/adump, but for other platforms, check the platform documentation to learn the correct target directory. In Microsoft Windows, you can access this information through Event Viewer.

If you set the AUDIT_TRAIL initialization parameter to XML, then Oracle Database writes audit records to the operating system as XML files. The V$XML_AUDIT_TRAIL view makes XML audit records available to database administrators through a SQL

query, providing enhanced usability. Querying this view parses all XML files (all files with a `.xml` extension) in the `AUDIT_FILE_DEST` directory to, and then presents them in relational table format. Because XML is a standard document format, many utilities are available to parse and analyze XML data. Consult the operating system-specific Oracle Database documentation to find if this feature has been implemented on your operating system.

Be aware that 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 case, Oracle Database still allows 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 ensure this does not occur.

If you configure auditing to use the database audit trail, you can prevent this potential loss of audit information. Oracle Database prevents audited events from occurring if the audit trail is unable to accept the database audit record for the statement.

Specifying a Directory for the Operating System Audit Trail

Use the `AUDIT_FILE_DEST` initialization parameter to specify an operating system directory into which the audit trail is written, when the `AUDIT_TRAIL` initialization parameter is set to `OS` or to `XML`. You must set `AUDIT_FILE_DEST` to a valid directory with permissions restricted to the owner of the Oracle software and the `DBA` group. Mandatory auditing information also goes into that directory, as do audit records for user `SYS` if the `AUDIT_SYS_OPERATIONS` initialization parameter is specified. You must change `AUDIT_FILE_DEST` using the following `ALTER SYSTEM` statement, which enables the new destination to be effective for all subsequent sessions.

```
ALTER SYSTEM SET AUDIT_FILE_DEST = directory_path DEFERRED;
```

If you do not set the `AUDIT_FILE_DEST` parameter, then Oracle Database places the file in the following default locations:

- **Linux and Solaris**: `$ORACLE_BASE/admin/$DB_UNIQUE_NAME/adump`

  For example:

  ```
  /opt/oracle/app/oracle/admin/orcl/adump
  ```

- **Windows**: `%ORACLE_BASE%\admin\%DB_UNIQUE_NAME\adump`

  For example:

  ```
  C:\ORACLE\ADMIN\ORCL\ADUMP
  ```

Notes:

- If your operating system supports an audit trail, then its location is operating system-specific. For example, when the AUDIT_TRAIL initialization parameter is set to OS, then Windows operating systems write audit records as events to the application event log. On most UNIX platforms, it is $ORACLE_BASE/admin/$DB_UNIQUE_NAME/adump, but for other platforms, check the platform documentation to learn the correct target directory.
- When the AUDIT_TRAIL initialization parameter is set to XML (or XML, EXTENDED), then Oracle Database writes audit records to XML-formatted operating system files. The XML-format audit records are written to the directory specified by the AUDIT_FILE_DEST parameter on all platforms, including Windows.

Decoding Operating System Audit Trial Records

Oracle Database encodes the operating system audit trail records. You can decode this information by referring to the appropriate data dictionary tables and error messages.

Table 6–5 describes the information that is encoded and where you can find its decoded version.

Table 6–5 Encoding Information in Audit Trail Records

Encoded Information	How to Decode
Action code	Describes the operation performed or attempted, using codes listed in the AUDIT_ACTIONS data dictionary table, with their descriptions.
Privileges used	Describes any system privileges used to perform the operation, using codes listed in the SYSTEM_PRIVILEGE_MAP table, with their descriptions.
Completion code	Describes the result of the attempted operation, using codes listed in *Oracle Database Error Messages*, with their descriptions. Successful operations return a value of zero, and unsuccessful operations return an Oracle Database error code corresponding to the reason the operation was unsuccessful.

See also "Activities That Are Always Audited" on page 6-5 for how the operating system file captures audit information for activities that always audited.

Deciding Whether to Use the Database or Operating System Audit Trail

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.

Using the operating system audit trail offers these advantages:

- Audit records stored in operating system files can be more secure than database-stored audit records because access can require file permissions that database administrators do not have. Greater availability is another advantage to operating system storage for audit records, because they remain available even if the database is temporarily inaccessible.
- If the AUDIT_TRAIL initialization parameter is set to XML (or XML, EXTENDED), then Oracle Database writes audit records to the operating system as XML files. You can use the V$XML_AUDIT_TRAIL view to make such XML audit records available to DBAs through a SQL query, providing enhanced usability. Querying this view causes all XML files (all files with an .xml extension) in the AUDIT_FILE_DEST directory to be parsed and presented in relational table format.
- The DBA_COMMON_AUDIT_TRAIL view includes the standard and fine grained audit trails written to database tables, XML-format audit trail records, and the contents of the V$XML_AUDIT_TRAIL dynamic view (standard, fine grained, SYS and mandatory).
- Using your operating system audit trail can enable you to consolidate audit records from multiple sources, including Oracle Database and other applications. Examining system activity can be more efficient with all audit records in one place. If you use XML audit records, then you can use of any standard XML editing tool to review or extract information from those records.

See Also:

- Your operating system-specific documentation for information about its auditing capabilities.
- Table 6–10 on page 6-47 for a list of database audit trails that are created when you first install Oracle Database

Auditing SQL Statements

SQL statement auditing is the selective auditing of related groups of SQL statements regarding a particular type of database structure or schema object, but not a specifically named structure or schema object.

This section includes the following topics:

- Types of SQL Statements That Are Audited
- Enabling SQL Statement Auditing
- Disabling SQL Statement Auditing

See also "Focusing Statement, Privilege, and Schema Auditing" on page 6-29 for additional information about auditing SQL statements.

Types of SQL Statements That Are Audited

The statements that you can audit are in the following 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, for example, by auditing the activities of all database users or of only a select list.

Enabling SQL Statement Auditing

Use the AUDIT statement to enable SQL statement auditing. You must have the AUDIT SYSTEM system privilege before you can enable auditing. Typically, only the security administrator is granted this system privilege.

Example 6–4 shows how to audit the DROP TABLE SQL statement.

Example 6–4 Using AUDIT to Enable SQL Statement Auditing

```
AUDIT DROP TABLE;
```

If you plan to audit a statement using the SESSION or NOT EXISTS option of the AUDIT statement, follow these guidelines:

- **Auditing Connections and Disconnections**. The SESSION option of AUDIT 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. It inserts an audit record into the audit trail at connection time, and then updates the audit record at disconnect time. Cumulative information about a session is stored in a single audit record that corresponds to the session. This record can include the connection time, disconnection time, and logical and physical I/O 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 following example:

  ```
  AUDIT SESSION
  BY jward, swilliams;
  ```

- **Auditing Statements That Fail Because an Object Does Not Exist**. The NOT EXISTS option of the AUDIT statement specifies auditing of all SQL statements that fail because the target object does not exist.

  For example:

  ```
  AUDIT NOT EXISTS;
  ```

See *Oracle Database SQL Language Reference* for detailed information about the AUDIT statement.

Disabling SQL Statement Auditing

To disable SQL statement auditing, use the use the NOAUDIT SQL statement. You must have the AUDIT SYSTEM system privilege before you can disable auditing.

Example 6–5 shows examples of using the NOAUDIT statement to disable auditing.

Example 6–5 Using NOAUDIT to Disable Session and SQL Statement Auditing

```
NOAUDIT session;
NOAUDIT session BY preston, sebastian;
NOAUDIT DELETE ANY TABLE;
NOAUDIT SELECT TABLE, INSERT TABLE, DELETE TABLE,
    EXECUTE PROCEDURE;
```

Example 6–6 shows how to disable all auditing by using the NOAUDIT statement.

Example 6–6 Using NOAUDIT to Disable All Auditing

```
NOAUDIT ALL;
```

See *Oracle Database SQL Language Reference* for detailed information about the NOAUDIT statement.

Auditing Privileges

Privilege auditing audits statements that use a system privilege, such as SELECT ANY TABLE.

This section includes the following topics:

- Types of Privileges That Can Be Audited
- Enabling Privilege Auditing
- Disabling Privilege Auditing

See "Focusing Statement, Privilege, and Schema Auditing" on page 6-29 for additional information about auditing privileges.

Types of Privileges That Can Be Audited

You can audit the use of any system privilege. Similar to statement auditing, privilege auditing audits the activities of all database users or only a specified list.

If similar statement and privilege audit options are both set, then only a single audit record is generated. For example, if the statement clause TABLE and the system privilege CREATE TABLE are both audited, then only a single audit record is generated each time a table is created.

Privilege auditing does not occur if the action is already permitted by the existing owner and schema object privileges. Privilege auditing is 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.

See the listing of system privileges in the GRANT SQL statement section of *Oracle Database SQL Language Reference*.

Enabling Privilege Auditing

Privilege audit options are the same as their corresponding system privileges. For example, the option to audit use of the DELETE ANY TABLE privilege is DELETE ANY TABLE.

Example 6–7 shows how to audit the DELETE ANY TABLE privilege.

Example 6–7 Using AUDIT to Enable Privilege Auditing

```
AUDIT DELETE ANY TABLE
    BY ACCESS
```

```
    WHENEVER NOT SUCCESSFUL;
```

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. Usually, only the security administrator is granted this system privilege.

Disabling Privilege Auditing

The following statement turns off all privilege audit options:

```
NOAUDIT ALL PRIVILEGES;
```

To disable privilege auditing options, you must have the `AUDIT SYSTEM` system privilege. Usually, only the security administrator is granted this system privilege.

Auditing SQL Statements and Privileges in a Multitier Environment

You can use the `AUDIT` statement to audit the activities of a client in a multitier environment. In a multitier environment, Oracle Database preserves the identity of a client through all tiers. Thus, you can audit actions taken on behalf of the client by a middle-tier application. To do so, use the `BY proxy` clause in your `AUDIT` statement.

This clause allows you the following options:

- 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 client identity in a database session, enabling the auditing of end-user actions through the middle-tier application. The end-user client identity then shows up in the audit trail.

Example 6–8 shows how to audit `SELECT TABLE` statements issued on behalf of client `jackson` by the proxy application server `appserve`.

Example 6–8 Using AUDIT to Audit a SQL Statement on Behalf of a Proxy User

```
AUDIT SELECT TABLE
 BY appserve ON BEHALF OF jackson;
```

> **See Also:** *Oracle Database Concepts* for more information about proxies and multitier applications

Auditing Schema Objects

Schema object auditing audits 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.

This section includes the following topics:

- Types of Schema Objects That Can Be Audited
- Schema Object Audit Options for Views, Procedures, and Other Elements
- Enabling Schema Object Auditing
- Disabling Object Auditing

See "Focusing Statement, Privilege, and Schema Auditing" on page 6-29 for additional information about auditing schema objects.

Types of Schema Objects That Can Be Audited

You can audit statements that refer to tables, views, sequences, standalone stored procedures or functions, and packages, but not individual procedures within packages.

You cannot directly audit statements that reference clusters, database links, indexes, or synonyms. However, you can indirectly audit access to these schema objects, by auditing the operations that affect the base table.

Schema object audit options are always set for all users of the database. You cannot set these options for a specific list of users. You can set default schema object audit options for all auditable schema objects.

> **See Also:** *Oracle Database SQL Language 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 also apply to the resulting SQL statements.

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 the `employees_departments` view, 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 removed from, and subsequently replaced in, the shared pool. Auditing a schema object invalidates that schema object in the cache and then reloads it in the cache. 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 does not generate an audit record for the `employees` table.

Table 6–6 lists auditing actions that are now available in Oracle Database 11*g* Release 1 (11.1).

Table 6–6 Auditing Actions Newly Enabled by Oracle Database 11g Release 1 (11.1)

Object or Element	Auditable Action
Mining Model	`ALTER, AUDIT, COMMENT, GRANT, RENAME, SELECT`
OLAP Primary Dimension	`ALTER, AUDIT, DELETE, INSERT, SELECT, CREATE`
OLAP Cube	`ALTER, AUDIT, DELETE, SELECT, UPDATE, CREATE`
OLAP Measure Folder	`AUDIT, DELETE, INDEX, SELECT, CREATE`
OLAP InterAction	`AUDIT, UPDATE, CREATE`
Edition	`ALTER, AUDIT, COMMENT, GRANT`

Table 6–7 lists auditing options that are now enabled in Oracle Database 11*g* Release 1 (11.1).

Table 6–7 System Auditing Options Enabled in Oracle Database 11g Release 1 (11.1)

System	Auditable Action
Edition	`CREATE ANY EDITION, DROP ANY EDITION, ALTER ANY EDITION, COMMENT EDITION, GRANT EDITION, USE EDITION`
Primary Dimension	`CREATE PRIMARY DIMENSION, ALTER ANY PRIMARY DIMENSION, CREATE ANY PRIMARY DIMENSION, DELETE ANY PRIMARY DIMENSION, DROP ANY PRIMARY DIMENSION, INSERT ANY PRIMARY DIMENSION, SELECT ANY PRIMARY DIMENSION, UPDATE ANY PRIMARY DIMENSION`
Cube	`CREATE CUBE, ALTER ANY CUBE, CREATE ANY CUBE, DROP ANY CUBE, SELECT ANY CUBE, UPDATE ANY CUBE`
Measure Folder	`CREATE MEASURE FOLDER, CREATE ANY MEASURE FOLDER, DELETE ANY MEASURE FOLDER, DROP ANY MEASURE FOLDER, INSERT ANY MEASURE FOLDER`
Interaction	`CREATE INTERACTION, CREATE ANY INTERACTION, DROP ANY INTERACTION, UPDATE ANY INTERACTION`

Enabling Schema Object Auditing

You can use the `AUDIT` statement to enable object auditing. *Oracle Database SQL Language Reference* lists valid object audit options for `AUDIT` 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 schema of the user. The `AUDIT ANY` system privilege is required to set an object audit option for an object contained in another user schema or to set the default object auditing option. Usually, only the security administrator is granted the `AUDIT ANY` privilege.

Example 6–9 shows how to audit all successful and unsuccessful DELETE statements on the laurel.emp table, BY SESSION (the default value).

Example 6–9 Using AUDIT to Enable Auditing for Schema Objects

```
AUDIT DELETE ON laurel.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;
```

Disabling Object Auditing

Use the NOAUDIT statement to disable object auditing. The following statements turn off the corresponding auditing options:

```
NOAUDIT DELETE
   ON emp;
NOAUDIT SELECT, INSERT, DELETE
   ON jward.dept;
```

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 that are 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 the schema of another user 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.

Focusing Statement, Privilege, and Schema Auditing

Oracle Database lets you focus statement, privilege, and schema object auditing in three areas.

- Auditing Statement Executions: Successful, Unsuccessful, or Both
- Number of Audit Records from Multiple Executions of a Statement
- Auditing Actions Performed by Specific Users

Auditing Statement Executions: Successful, Unsuccessful, or Both

For statement, privilege, and schema object auditing, Oracle Database permits 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 statement can expose users who are snooping or acting maliciously; though most unsuccessful SQL statements are 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 fail to execute because they 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 failed for other reasons. One example is when a `CREATE TABLE` auditing condition is set, but some `CREATE TABLE` statements fail due to insufficient quota for the specified tablespace.

When your audit statement includes the `WHENEVER SUCCESSFUL` clause, you will be able to audit 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 able to audit 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, then the audit trail can have one or more related records. The controlling clause `BY ACCESS` in the `AUDIT` statement generates a separate audit record for each execution of an auditable operation within a cursor. If you use the `BY SESSION` clause instead, then the audit trail contains 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, some 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:

- Creating One Audit Record for Each Operation with the BY ACCESS Clause
- Creating One Audit Record for Each Operation with the BY ACCESS Clause

See *Oracle Database SQL Language Reference* for additional information about the `BY ACCESS` clause in `AUDIT`.

Creating One Audit Record for Each Operation with the BY ACCESS Clause 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 or not 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 to `BY ACCESS`.
- The user `jward` connects to the database and issues five `SELECT` statements against the table named `departments` and then disconnects from the database.
- The user `swilliams` connects to the database and issues three `SELECT` statements against the `departments` table and then disconnects from the database.

The single audit trail contains eight records, one recored for each `SELECT` statement.

Creating a Single Audit Record for Each Session with the BY SESSION Clause 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 then disconnects from Oracle Database.

BY SESSION: Example 1

Assume the following:

- The `SELECT TABLE` statement auditing option is set to `BY SESSION`.
- The user `jward` connects to the database and issues five `SELECT` statements against the table named `departments` and then disconnects from the database.
- The user `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 to `BY SESSION`.
- The user `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, then Oracle Database generates and stores an audit record each time an access is made. Therefore, in this auditing configuration, `BY SESSION` is equivalent to `BY ACCESS`.

Auditing Actions Performed by Specific Users

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.

Example 6–10 shows how to audit statements by users `scott` and `blake` when they query or update a table or view.

Example 6–10 Using AUDIT to Audit User Actions

```
AUDIT SELECT TABLE, UPDATE TABLE
     BY scott, blake;
```

See *Oracle Database SQL Language Reference* for additional information about auditing by user.

Auditing Network Activity

You can use the `AUDIT` statement to audit unexpected errors in network protocol or internal errors in the network layer. This section includes the following topics:

- Enabling Network Auditing
- Types of Errors Recorded in Network Auditing
- Disabling Network Auditing

Enabling Network Auditing

To enable network auditing, use the `AUDIT` statement. For example:

```
AUDIT NETWORK;
```

See *Oracle Database SQL Language Reference* for additional information about the `AUDIT` statement.

Types of Errors Recorded in Network Auditing

The errors that network auditing uncovers (such as `ACTION 122 Network Error in AUDIT_ACTIONS`) are not connection failures. There can be several possible causes of network errors. One possible cause could be an internal event set by a database engineer for testing purposes. Other causes include conflicting configuration settings for encryption, such as the network not finding the information required to create or process expected encryption. Table 6–8 shows four network error conditions.

Table 6–8 Auditable Network Error Conditions

Error	Cause	Action
`TNS-02507` `Encryption algorithm not installed`	After picking an algorithm, the server was unable to find an index for it in its table of algorithms. This should be impossible because the algorithm was chosen (indirectly) from that list.	Turn on tracing for further details, and then rerun the operation. (Note that this error is not normally visible to the user.) If the error persists, then contact Oracle Support Services.
`TNS-12648` `Encryption or data integrity algorithm list empty`	An Oracle Advanced Security list-of-algorithms parameter was empty.	Change the list to contain the name of at least one installed algorithm, or remove the list entirely if every installed algorithm is not acceptable.

Table 6–8 (Cont.) Auditable Network Error Conditions

Error	Cause	Action
TNS-12649 Unknown encryption or data integrity algorithm	An Oracle Advanced Security list-of-algorithms parameter included an algorithm name that was not recognized.	Remove that algorithm name, correct it if it was misspelled, or install the driver for the missing algorithm.
TNS-12650 No common encryption or data integrity algorithm	The client and server have no algorithm in common for either encryption or data integrity or both.	Choose sets of algorithms that overlap. In other words, add one of the client algorithm choices to the server list, or add one of the server list choices to the client algorithm.

Disabling Network Auditing

Example 6–11 shows how to turn off network auditing.

Example 6–11 Using NOAUDIT to Disable Network Auditing

```
NOAUDIT NETWORK;
```

Oracle Database disables network auditing, including auditing for DB link usage and login types.

See *Oracle Database SQL Language Reference* for more information about the NOAUDIT statement.

Auditing Administrative Users

You can audit administrative users by using the following methods:

- Auditing Users Who Connect as SYS
- Using the Syslog Audit Trail to Audit System Administrators on UNIX Systems

Auditing Users Who Connect as SYS

You can fully audit sessions for users who connect as SYS, including all users connecting using the SYSDBA or SYSOPER privileges. Use the AUDIT_SYS_OPERATIONS initialization parameter to specify whether these users are to be audited.

Example 6–12 shows how to set the AUDIT_SYS_OPERATIONS initialization parameter to TRUE, which specifies that SYS is to be audited.

Example 6–12 Enabling Auditing for Users Who Connect as SYS

```
ALTER SYSTEM SET AUDIT_SYS_OPERATIONS=TRUE SCOPE=SPFILE;
```

By default, AUDIT_SYS_OPERATIONS is set to TRUE.

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, when the AUDIT_TRAIL initialization parameter is set to OS, Oracle Database writes audit records as events to the Event Viewer log file. If either XML or XML, EXTENDED is specified, then audit records are written as XML files in the directory specified by the AUDIT_FILE_DEST parameter.

Notes: The adump directory is the first default location used if the AUDIT_FILE_DEST initialization parameter is not set or does not point to a valid directory. If writing to that first default location fails, then Oracle Database uses the *$ORACLE_HOME/rdbms/audit* directory as the backup default location. If that attempt fails, then the audited operation fails and a message is written to the alert log.

When AUDIT_TRAIL is set to OS (for operating system), audit file names continue to be in the following form:

short_form_of_process_name_processid.aud

For example, the short process name ora is used for dedicated server processes, and the names s001, s002, and so on are used for shared server processes.

When AUDIT_TRAIL is set to XML or XML, EXTENDED, the same audit file names have the extension xml instead of aud.

If you do not specify the AUDIT_FILE_DEST initialization parameter, then the default location is $ORACLE_BASE/admin/$DB_UNIQUE_NAME/adump in Linux and Solaris, and $ORACLE_BASE\admin\$DB_UNIQUE_NAME\adump in Windows.

For other operating systems, refer to 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='laurel';
```

When SYS auditing is enabled, both the ALTER SYSTEM and UPDATE statements are displayed in the operating system audit file, similar to the following:

```
Thu Jun 24 12:58:00 2007
ACTION: 'CONNECT'
DATABASE USER: '/'
OSPRIV: SYSDBA
CLIENT USER: laurel
CLIENT TERMINAL: pts/2
STATUS: 0

Thu Jan 24 12:58:00 2007
ACTION: 'alter system flush shared_pool'
DATABASE USER: ''
OSPRIV: SYSDBA
CLIENT USER: laurel
CLIENT TERMINAL: pts/2
STATUS: 0

Thu Jan 24 12:58:00 2007
ACTION: 'update salary set base=1000 where name='myname''
DATABASE USER: ''
OSPRIV: SYSDBA
CLIENT USER: laurel
CLIENT TERMINAL: pts/2
STATUS: 0
```

Because of the superuser privileges available to users who connect as SYSDBA, Oracle recommends that database administrators rarely use this connection and only when necessary. Database administrators can usually perform normal day-to-day maintenance activity. These database administrators are typical database users with the DBA role, or have a DBA role (for example, mydba or jr_dba) that your organization customizes.

Using the Syslog Audit Trail to Audit System Administrators on UNIX Systems

On UNIX systems, you can audit the activities of system administrators by creating a syslog audit trail. This section includes the following topics:

- About the Syslog Audit Trail
- Format of the Information Stored in the Syslog Audit Trail
- Configuring Syslog Auditing

See "Activities That Are Always Recorded in the Operating System and Syslog Audit Trails" on page 6-5.

> **Note:** The security vulnerability that is exposed with an operating system audit trail is not an issue on Windows operating systems. This is because audit records cannot by modified directly. Instead, audit records on Windows operating systems are stored and monitored through Event Viewer.

About the Syslog Audit Trail

A potential security vulnerability for an operating system audit trail is that a privileged user, such as a database administrator, can modify or delete database audit records. To minimize this risk, you can use a syslog audit trail. Syslog is a standard protocol on UNIX-based systems for logging information from different components of a network. Applications call the syslog() function to log information to the syslog daemon, which then determines where to log the information. You can configure syslog to log information to a file name syslog.conf, to the console, or to a remote, dedicated log host. (The syslog.conf file is only used for configuration.) You can also configure syslog to alert a specified set of users when information is logged.

Because applications, such as an Oracle process, use the syslog() function to log information to the syslog daemon, a privileged user would not have permissions to the file system where syslog messages are logged. For this reason, audit records stored using a syslog audit trail can be more secure than audit records stored using an operating system audit trail. In addition to restricting permissions to a file system for a privileged user, for a syslog audit trail to be secure, neither privileged users nor the Oracle process should have root access to the system where the audit records are written.

Caution: You should have a strong understanding of how to work with `syslog` before enabling `syslog` auditing. See the following references for more information about syslog:

- *Oracle Database Reference* for information about the `AUDIT_SYSLOG_LEVEL` initialization parameter
- The UNIX man page for the `syslogd` utility for more information about the *`facility.priority`* settings and their directory paths

Format of the Information Stored in the Syslog Audit Trail

Similar to the operating system audit trail records, Oracle Database encodes the syslog records to ensure greater security. To find the contents of the syslog records, query the appropriate data dictionary tables and error messages. See "Finding Information About Audited Activities" on page 6-47 for ways to query the data dictionary tables for security-related information.

Table 6–5 describes the information that is encoded and where you can find its decoded version.

Table 6–9 Encoded Information in Audit Trail Records

Encoded Information	How to Decode
Action code	Describes the operation performed or attempted, using codes listed in the `AUDIT_ACTIONS` data dictionary table, with their descriptions.
Privileges used	Describes any system privileges used to perform the operation, using codes listed in the `SYSTEM_PRIVILEGE_MAP` table, with their descriptions.
Completion code	Describes the result of the attempted operation, using codes listed in *Oracle Database Error Messages*, with their descriptions. Successful operations return a value of zero, and unsuccessful operations return an Oracle error code corresponding to the reason the operation was unsuccessful.

Configuring Syslog Auditing

To enable syslog auditing, follow these steps:

1. Assign a value of `OS` to the `AUDIT_TRAIL` initialization parameter, as described in "Enabling or Disabling the Standard Audit Trail" on page 6-13.

   For example:

   ```
   ALTER SYSTEM SET AUDIT_TRAIL=OS SCOPE=SPFILE;
   ```

2. Manually add and set the `AUDIT_SYSLOG_LEVEL` parameter to the initialization parameter file, `init`*`sid`*`.ora`.

   Set the `AUDIT_SYSLOG_LEVEL` parameter to specify a facility and priority in the format `AUDIT_SYSLOG_LEVEL=`*`facility.priority`*.

   - *`facility`*: Describes the part of the operating system that is logging the message. Accepted values are `user`, `local0`–`local7`, `syslog`, `daemon`, `kern`, `mail`, `auth`, `lpr`, `news`, `uucp`, and `cron`.

     The `local0`–`local7` values are predefined tags that enable you to sort the syslog message into categories. These categories can be log files or other

destinations that the syslog utility can access. To find more information about these types of tags, refer to the `syslog` utility MAN page.

- *`priority`*: Defines the severity of the message. Accepted values are `notice`, `info`, `debug`, `warning`, `err`, `crit`, `alert`, and `emerg`.

The syslog daemon compares the value assigned to the facility argument of the `AUDIT_SYSLOG_LEVEL` parameter with the `syslog.conf` file to determine where to log information.

For example, the following statement identifies the facility as `local1` with a priority level of `warning`:

```
AUDIT_SYSLOG_LEVEL=local1.warning
```

See *Oracle Database Reference* for more information about `AUDIT_SYSLOG_LEVEL`.

3. Add the audit file destination to the `syslog` configuration file `/etc/syslog.conf`.

   For example, assuming you had set the `AUDIT_SYSLOG_LEVEL` to `local1.warning`, enter the following:

   ```
   local1.warning /var/log/audit.log
   ```

   This setting logs all warning messages to the `/var/log/audit.log` file.

4. Restart the syslog logger:

   ```
   $/etc/rc.d/init.d/syslog restart
   ```

   Now, all audit records will be captured in the file `/var/log/audit.log` through the syslog daemon.

5. Restart the database instance:

   ```
   CONNECT SYS / AS SYSOPER
   Enter password: password
   Connected.
   SQL> SHUTDOWN;
   Database closed.
   Database dismounted.
   ORACLE instance shut down.
   SQL> STARTUP;
   ORACLE instance started.
   ```

Using Triggers to Record Customized Standard Auditing Information

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 `employee_salaries` table to generate an audit record whenever the salary of an employee is increased by more than 10 percent. You can include selected information, such as the values of `salary` before and after it was changed.

Example 6–13 shows a trigger used to record customized audit information.

Example 6–13 Using a Trigger to Record Customized Audit Information

```
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, though Oracle Database 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 and 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 fine-grained auditing, which provides an extensible auditing mechanism supporting definition of key conditions for granular audit and as an event handler to actively alert administrators to misuse of data access rights. See "Using Fine-Grained Auditing to Monitor Specific Activities" on page 6-38.

Using Fine-Grained Auditing to Monitor Specific Activities

Fine-grained auditing enables you to create policies that define specific conditions that must take place for the audit to occur. This section explores the following topics:

- About Fine-Grained Auditing
- Who Can Perform Fine-Grained Auditing?
- Activities That Are Always Recorded in Fine-Grained Auditing
- Archiving and Purging the Fine-Grained Audit Trail
- Using the DBMS_FGA Package to Manage Fine-Grained Audit Policies
- Creating Operating System XML Fine-Grained Audit Records

About Fine-Grained Auditing

In general, fine-grained audit 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 row, the query is audited.

For example, you can use fine-grained auditing to audit the following types of actions:

- A table being accessed between 9:00 p.m. and 6:00 a.m. or on Saturday and Sunday
- An IP address from outside the corporate network being used
- A table column being selected or updated
- A value in a table column being used

Fine-grained auditing creates a more meaningful audit trail, one that includes only very specific actions that you want to audit. It excludes unnecessary information that occurs if each table access was recorded. Fine-grained auditing has the following advantages over standard auditing:

- **It performs a Boolean condition check.** If the Boolean condition you specify is met, for example, a table being accessed on a Saturday, then the audit takes place.
- **It captures the SQL that triggered the audit.** You can capture both the SQL statement that caused the audit, and any associated bind variables. Be aware that you can only capture data from scalar column types. You cannot capture data from object columns, LOBs, or user-defined column types. For example, suppose you have the following query:

  ```
  SELECT NAME FROM EMPLOYEE WHERE SSN = :1
  ```

  If `:1` is of integer type and the value for `SSN` is `123566789`, then the audit trail can capture this information. However, the audit trail cannot capture this information if `:1` is a BLOB, CLOB, object, or user-defined type.

  This feature is available to standard auditing if you set the `AUDIT_TRAIL` parameter to `DB, EXTENDED`.
- **It adds extra protection to sensitive columns.** You can audit specific relevant columns that may hold sensitive information, such as salaries or social security numbers.
- **It provides an event handler feature.** For example, you can write a function that calls an alert when an audited column that should not be changed at midnight is updated.
- **You do not need to set initialization parameters to enable fine-grained auditing.** Instead of setting initialization parameters such as `AUDIT_TRAIL`, you use the `DBMS_FGA PL/SQL` package to add and remove fine-grained auditing policies as necessary 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 `SYS.FGA_LOG$` table. To find information about fine-grained audit policies, you can use the `DBA_FGA_AUDIT_TRAIL` view. The `DBA_COMMON_AUDIT_TRAIL` view combines both standard and fine-grained audit log records. In addition, you can use the `V$XML_AUDIT_TRAIL` view to find fine-grained audit records that were written in XML formatted files. For detailed information about these views, see *Oracle Database Reference*.

Note:

- Fine-grained auditing is supported only with cost-based optimization. For queries using rule-based optimization, fine-grained auditing checks before applying row filtering, which could result in an unnecessary audit event trigger.
- 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 system change number (SCN).

Who Can Perform Fine-Grained Auditing?

To perform fine-grained auditing, you must have `EXECUTE` privileges on the `DBMS_FGA` PL/SQL package. The package is owned by the `SYS` user.

Activities That Are Always Recorded in Fine-Grained Auditing

Oracle Database records all data manipulation language (DML) statements, such as INSERT, UPDATE, MERGE, and DELETE on the SYS.FGA_LOG$ in the table SYS.AUD$. It performs the audit even if auditing has not been enabled for the table in which these activities occur. You can check these activities by running the DBA_FGA_AUDIT_TRAIL and DBA_COMMON_AUDIT_TRAIL views.

Archiving and Purging the Fine-Grained Audit Trail

To archive fine-grained audit records, you can copy the relevant records to a normal database table, for example, using INSERT INTO *table* SELECT ... FROM SYS.FGA_LOG$ .... Alternatively, you can export the SYS.FGA_LOG$ table to an operating system file. "Archiving the Standard and Fine-Grained Audit Trails" on page 6-47 explains how to use Oracle Data Pump Export to export the SYS.FGA_LOG$ table to an operating system file.

To purge fine-grained audit records, you can delete them records from the SYS.FGA_LOG$ table. For example, to delete *all* fine-grained audit records, enter the following statement:

```
DELETE FROM SYS.FGA_LOG$;
```

Alternatively, to delete all audit records from the fine-grained audit trail generated as a result of auditing the table emp, enter the following statement:

```
DELETE FROM SYS.FGA_LOG$
     WHERE obj$name='EMP';
```

Using the DBMS_FGA Package to Manage Fine-Grained Audit Policies

This section explores the following topics:

- About the DBMS_FGA PL/SQL Package
- Creating a Fine-Grained Audit Policy
- Adding Alerts to a Fine-Grained Audit Policy
- Disabling and Enabling a Fine-Grained Audit Policy
- Dropping a Fine-Grained Audit Policy

About the DBMS_FGA PL/SQL Package

To manage a fine-grained audit policy, you use the DBMS_FGA PL/SQL package. This package enables you to add all combinations of SELECT, INSERT, UPDATE, and DELETE statements to one policy. You can also audit MERGE statements, by auditing the underlying actions of INSERT and UPDATE. To audit MERGE statements, configure fine-grained access on the INSERT and UPDATE statements. Only one record is generated for each policy for successful MERGE operations. To administer fine-grained audit policies, you need to have EXECUTE privileges on the DBMS_FGA package.

The audit policy is bound to the table for which you created it. This simplifies the management of audit policies because the policy only needs to be changed once in the database, not in each application. In addition, no matter how a user connects to the database—from an application, a Web interface, or through SQL*Plus or Oracle SQL Developer—Oracle Database records any actions that affect the policy.

If any rows returned from a query match the audit condition that you define, then Oracle Database inserts an audit entry into the fine-grained audit trail. This entry excludes all the information that is reported in the regular audit trail. In other words,

only one row of audit information is inserted into the audit trail for every fine-grained audit policy that evaluates to true. You can optionally define an event handler to process this event, for example, by sending an alert to the pager of an administrator.

For detailed information about the syntax of the DBMS_FGA package, see *Oracle Database PL/SQL Packages and Types Reference*. See also *Oracle Database Advanced Application Developer's Guide*.

Creating a Fine-Grained Audit Policy

To create a fine-grained audit policy, use the DBMS_FGA.ADD_POLICY procedure. This procedure creates an audit policy using the supplied predicate as the audit condition. The maximum number of fine-grained policies on any table or view object is 256. Oracle Database stores the policy in the data dictionary table, but you can create the policy on any table or view that is not in the SYS schema.

The syntax for the ADD_POLICY procedure is:

```
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);
```

In this specification:

- object_schema: Specifies the schema of the object to be audited.
- object_name: Specifies the name of the object to be audited.
- policy_name: Specifies the name of the policy to be created.
- audit_condition: Specifies a Boolean condition in a row. Null is allowed. See "Auditing Specific Columns and Rows" on page 6-42 for more information.
- audit_column: Specifies one or more columns to audit, including hidden columns. If null or omitted, all columns are audited.
- handler_schema: If an alert is used to trigger a response when the policy is violated, specifies the name of the schema that contains the event handler. See "Adding Alerts to a Fine-Grained Audit Policy" on page 6-43 for more information.
- handler_module: Specifies the name of the event handler.
- enable: Enables or disables the policy using true or false. If omitted, the policy is enabled.
- statement_types: Specifies the SQL statements to be audited.
- audit_trail: Specifies the destination (DB or XML) of fine-grained audit records. Also specifies whether to populate LSQLTEXT and LSQLBIND in FGA_LOG$.
- audit_column_opts: If more than one column is specified in the audit_column parameter, determines whether to audit all or specific columns. See "Auditing Specific Columns and Rows" on page 6-42 for more information.

See *Oracle Database PL/SQL Packages and Types Reference* for additional details about the ADD_POLICY syntax.

Example 6–14 shows how to audit statements INSERT, UPDATE, DELETE, and SELECT on table HR.EMPLOYEES. Note that this example omits the audit_column_opts parameter, because it is not a mandatory parameter.

Example 6–14 Using DBMS_FGA.ADD_POLICY to Create a Fine-Grained Audit Policy

```
BEGIN
  DBMS_FGA.ADD_POLICY(
   object_schema      => 'hr',
   object_name        => 'employees',
   policy_name        => 'chk_hr_employees',
   audit_condition    =>  NULL,
   audit_column       =>  NULL,
   handler_schema     =>  NULL,
   handler_module     =>  NULL,
   enable             =>  TRUE,
   statement_types    => 'INSERT, UPDATE, SELECT, DELETE',
   audit_trail        =>  DBMS_FGA.DB + DBMS_FGA.EXTENDED);
END;
```

At this point, if you run the DBA_AUDIT_POLICIES view, you will find the new policy listed:

```
SELECT policy_name FROM DBA_AUDIT_POLICIES;
POLICY_NAME
------------------------------
CHK_HR_EMPLOYEES
```

Afterwards, any of the following SQL statements log an audit event record.

```
SELECT count(*) FROM hr.employees WHERE commission_pct = 20 and salary > 4500;

SELECT salary FROM hr.employees WHERE department_id = 50;

DELETE from hr.employees WHERE salary > 1000000;
```

Auditing Specific Columns and Rows

You can fine-tune the audit behavior by targeting a specific column, referred to as a *relevant column*, to be audited if a condition is met. To accomplish this, you use the audit_column parameter to specify one or more sensitive columns. In addition, you can audit data in specific rows by using the audit_condition parameter to define a Boolean condition.

Example 6–14 on page 6-42 performs an audit if anyone in Department 50 tries to access the salary and commission_pct columns.

```
audit_condition    => 'department_id = 50',
audit_column       => 'salary,commission_pct,'
```

As you can see, this feature is enormously beneficial. It not only enables you to pinpoint particularly important types of data to audit, but it provides increased protection for columns that contain sensitive data, such as social security numbers, salaries, patient diagnoses, and so on.

If the audit_column lists more than one column, you can use the audit_column_opts parameter to specify whether a statement is audited when the query references *any* column specified in the audit_column parameter or only when *all* columns are referenced. For example:

```
audit_column_opts   => DBMS_FGA.ANY_COLUMNS,
audit_column_opts   => DBMS_FGA.ALL_COLUMNS,
```

If you do not specify a relevant column, then auditing applies to all columns. That is, without a relevant column specified, auditing occurs whenever any specified statement type affects any column.

Using NULL for Audit Conditions

If you want to guarantee auditing of the specified actions (`statement_types`) affecting the specified columns (`audit_column`), specify the `audit_condition` parameter as `NULL` (or omit it), which is interpreted as `TRUE`. Only specifying `NULL` audits the specified actions (`statement_types`) affecting the specified columns (`audit_column`).

Follow these guidelines:

- **Do not enter 1=1 as an audit condition because this feature is no longer used, and hence will not achieve the desired result.** `NULL` performs the audit even if no rows were processed, so that all actions on an `audit_column` with the policy are audited.
- **Do not use an empty string to specify NULL.** Using an empty string is not equivalent to `NULL` and will not reliably cause auditing of all actions on a table with this policy.

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.

Adding Alerts to a Fine-Grained Audit Policy

You can add an alert to a fine-grained audit policy that goes into effect when a user (or an intruder) violates the policy. You first need to create a procedure that generates the alert, and then use the following `ADD_POLICY` parameters to call this function when someone violates this policy:

- `handler_schema`: The schema in which the handler event is stored
- `handler_module`: The name of the event handler

The alert can come in any form that best suits your environment: an e-mail or pager notification, updates to a particular file or table, and so on. Creating alerts also helps to meet certain compliance regulations, such as California Senate Bill 1386.

Use the following syntax to create the alert procedure:

```
PROCEDURE fname (
  object_schema VARCHAR2,
  object_name VARCHAR2,
  policy_name VARCHAR2 )
AS ...
```

In this specification:

- *`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.
- *`policy_name`* is the name of the policy being enforced.

For example, suppose a clerk wanted to find the salaries of highly paid coworkers. With the audit policy created in Example 6–14 on page 6-42 in place, his actions would

be immediately logged. To notify an administrator of the overly curious behavior of the clerk, you would create a procedure to record this information, and then modify the chk_hr_employee audit policy to call this procedure.

To create this type of alert, log on to SQL*Plus with administrative privileges (user SYSTEM), and follow these steps:

1. Create a table to record the violations to the chk_hr_employees policy:

```
CREATE TABLE emp_violations (
 username VARCHAR(20),
 userhost VARCHAR(20),
 time TIMESTAMP);
```

2. Create the procedure that will generate the alert:

```
CREATE OR REPLACE PROCEDURE emp_violations_alert (
 hr_schema VARCHAR2,
 employees_table VARCHAR2,
 hr_policy VARCHAR2)
AS
BEGIN
 INSERT INTO sec_mgr.emp_violations (
 username, userhost, time)
 SELECT user, sys_context('userenv','terminal'), sysdate FROM DUAL;
END emp_violations_alert;
```

3. If you already created the example chk_hr_employees policy in Example 6–14, then drop that policy:

```
BEGIN
 DBMS_FGA.DROP_POLICY(
  object_schema  => 'hr',
  object_name    => 'employees',
  policy_name    => 'chk_hr_employees');
END;
```

4. Re-create the chk_hr_employees policy to include a call to the emp_violations_alert alert.

   Because you now are concerned with financial data access, also modify this policy to protect only the salary and commission_pct columns.

```
BEGIN
 DBMS_FGA.ADD_POLICY (
  object_schema      =>  'hr',
  object_name        =>  'employees',
  policy_name        =>  'chk_hr_employees',
  audit_condition    =>  'department_id = 50',
  audit_column       =>  'salary,commission_pct',
  handler_schema     =>  'system',
  handler_module     =>  'emp_violations_alert',
  enable             =>   TRUE,
  statement_types    =>  'INSERT, UPDATE, SELECT, DELETE',
  audit_trail        =>   DBMS_FGA.XML + DBMS_FGA.EXTENDED,
  audit_column_opts  =>   DBMS_FGA.ANY_COLUMNS);
END;
```

Now you are ready to test the alert:

1. Connect to SQL*Plus as user HR and perform a SELECT statement on the employees table.

```
CONNECT HR
Enter password: password
Connected.

SQL> SELECT COUNT(*) FROM employees WHERE SALARY > 4500;

  COUNT(*)
----------
       60
```

2. Connect as SYSTEM and then perform the same SELECT statement that HR performed.

```
CONNECT SYSTEM
Enter password: password
Connected.

SQL> SELECT COUNT(*) FROM hr.employees WHERE SALARY > 4500;

  COUNT(*)
----------
       60
```

3. As user SYSTEM, check the emp_violations table, which contains the two violations.

```
SQL> SELECT * FROM emp_violations;

USERNAME             USERHOST             TIME
-------------------- -------------------- ------------------------------
HR                   SHOBEEN-PC           17-APR-07 03.30.47.000000 PM
SYSTEM               SHOBEEN-PC           17-APR-07 03.53.18.000000 PM
```

As you can see, anyone who violates the chk_hr_employees policy is recorded in the emp_violations table, including users who have administrative privileges.

Oracle Database executes the audit function as an autonomous transaction, committing only the actions of the handler_module setting and not any user transaction. The function has no effect on any user SQL transaction.

After the first row of interest is fetched, the event is recorded, and the emp_violations_alert audit function runs. The audit event record generated is stored in the DBA_FGA_AUDIT_TRAIL view, which is fga_log$ in the SYS schema in the SYSTEM tablespace. This table has reserved columns (such as SQL_TEXT and SQL_BIND) for recording SQL text, policy name, and other information. The SQLBIND and SQLTEXT values are recorded in the LSQLTEXT and LSQLBIND columns of FGA_LOG$ only if the policy specifies audit_trail = DBMS_FGA.DB + DBMS_FGA.EXTENDED. If the policy specifies AUDIT_TRAIL=DBMS_FGA.XML, then the audit records would be written to XML-formatted operating system files.

Disabling and Enabling a Fine-Grained Audit Policy

You can disable a fine-grained audit policy by using the DBMS_FGA.DISABLE_POLICY procedure. The syntax for DISABLE_POLICY is:

```
DBMS_FGA.DISABLE_POLICY(
   object_schema  VARCHAR2,
   object_name    VARCHAR2,
   policy_name    VARCHAR2 );
```

Example 6–15 shows how to disable the fine-grained audit policy created in Example 6–14 on page 6-42.

Example 6–15 Disabling a Fine-Grained Audit Policy

```
DBMS_FGA.DISABLE_POLICY(
  object_schema        => 'hr',
  object_name          => 'employees'
  policy_name          => 'chk_hr_employees');
```

For detailed information about the DISABLE_POLICY syntax, see *Oracle Database PL/SQL Packages and Types Reference*.

Example 6–16 show how to reenable the chk_hr_emp policy by using the DBMS_FGA.ENABLE_POLICY procedure:

Example 6–16 Enabling a Fine-Grained Audit Policy

```
DBMS_FGA.ENABLE_POLICY(
  object_schema        => 'hr',
  object_name          => 'employees',
  policy_name          => 'chk_hr_employees'
  enable               => 'true');
```

For detailed information about the ENABLE_POLICY syntax, see *Oracle Database PL/SQL Packages and Types Reference*.

Dropping a Fine-Grained Audit Policy

Example 6–17 shows how to drop a fine-grained audit policy by using the DBMS_FGA.DROP_POLICY procedure.

Example 6–17 Dropping a Fine-Grained Audit Policy

```
DBMS_FGA.DROP_POLICY(
  object_schema      => 'hr',
  object_name        => 'employees',
  policy_name        => 'chk_hr_employees');
```

See *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the DROP_POLICY syntax.

Creating Operating System XML Fine-Grained Audit Records

The values for the AUDIT_TRAIL parameter (XML and XML, EXTENDED) write fine-grained auditing records to operating system files in XML format.

Audit records stored in operating system files can be more secure than database-stored audit records because file permissions that database administrators do not have may be required to access the records. Operating system storage for audit records also offers higher availability, because such records remain available even if the database is temporarily inaccessible.

You can use the V$XML_AUDIT_TRAIL view to make audit records from XML files available to DBAs through a SQL query, providing enhanced usability. Querying this view causes all XML files (all files with an .xml extension) in the AUDIT_FILE_DEST directory to be parsed and presented in relational table format.

The DBA_COMMON_AUDIT_TRAIL view includes the contents of the V$XML_AUDIT_TRAIL dynamic view for standard and fine-grained audit records.

Because the audit XML files are stored in files with the `.xml` extension on all platforms, the dynamic view presents audit information similarly on all platforms. See *Oracle Database Reference* for detailed information about the `V$XML_AUDIT_TRAIL` view contents.

Archiving the Standard and Fine-Grained Audit Trails

You can create an archive of the standard audit and fine-grained audit trails by exporting their system tables (`SYS.AUD$` and `SYS.FGA_LOG$`) to operating system dump files. You should periodically archive the audit trail to prevent it from growing too large.

To archive the standard audit trail or fine-grained audit trail:

- that are less than the `scn` column

4. standard

Finding Information About Audited Activities

You can use data dictionary views to work with the audit trail. This section explores the following topics:

- Using Data Dictionary Views to Find Information About the Audit Trial
- Using Audit Trail Views to Investigate Suspicious Activities
- Deleting the Audit Trail Views

Using Data Dictionary Views to Find Information About the Audit Trial

Oracle Database stores audit records for standard auditing in the `SYS.AUD$` table and audit records for fine-grained auditing the `SYS.FGA_LOG$` table. Each of these tables is a single table in each Oracle database 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, then you can later delete these views. For detailed information about these views, see *Oracle Database Reference.*

Table 6–10 lists views that are provide auditing information.

Table 6–10 Views That Display Information about the Database Audit Trail

View	Description
`ALL_AUDIT_POLICIES`	Describes the fine-grained auditing policies on the tables and views accessible to the current user
`ALL_AUDIT_POLICY_COLUMNS`	Describes the fine-grained auditing policy columns on the tables and views accessible to the current user.
`ALL_DEF_AUDIT_OPTS`	Lists default object-auditing options that will be applied when objects are created
`AUDIT_ACTIONS`	Describes audit trail action type codes
`DBA_AUDIT_EXISTS`	Lists audit trail entries produced `BY AUDIT NOT EXISTS`
`DBA_AUDIT_OBJECT`	Lists audit trail records for all objects in the system
`DBA_AUDIT_POLICIES`	Lists all the fine-grained auditing policies on the system
`DBA_AUDIT_SESSION`	Lists all audit trail records concerning `CONNECT` and `DISCONNECT`

Table 6–10 (Cont.) Views That Display Information about the Database Audit Trail

View	Description
DBA_AUDIT_STATEMENT	Lists audit trail records concerning GRANT, REVOKE, AUDIT, NOAUDIT, and ALTER SYSTEM statements throughout the database
DBA_AUDIT_TRAIL	Lists all audit trail entries
DBA_COMMON_AUDIT_TRAIL	Combines standard and fine-grained audit log records, and includes SYS and mandatory audit records written in XML format
DBA_FGA_AUDIT_TRAIL	Lists audit trail records for fine-grained auditing.
DBA_OBJ_AUDIT_OPTS	Describes auditing options on all objects
DBA_PRIV_AUDIT_OPTS	Describes current system privileges being audited across the system and by user
DBA_STMT_AUDIT_OPTS	Describes current statement auditing options across the system and by user
USER_AUDIT_OBJECT	Lists audit trail records for statements concerning objects that are accessible to the current user
USER_AUDIT_SESSION	Lists all audit trail records concerning connections and disconnections for the current user
USER_AUDIT_STATEMENT	Lists audit trail records concerning GRANT, REVOKE, AUDIT, NOAUDIT, and ALTER SYSTEM statements issued by the user
USER_AUDIT_TRAIL	Lists audit trail entries relating to current user
USER_OBJ_AUDIT_OPTS	Describes auditing options on all objects owned by the current user
STMT_AUDIT_OPTION_MAP	Describes information about auditing option type codes

Using Audit Trail Views to Investigate Suspicious Activities

This section provides 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 altered without authorization.
- A high number of deadlocks occur, most likely because of users acquiring exclusive table locks.
- Rows are arbitrarily deleted from the emp table in laurel's schema.

You suspect the users jward and swilliams of several of these detrimental actions.

To investigate, you issue the following statements (in the order specified):

```
AUDIT ALTER, INDEX, RENAME ON DEFAULT
    BY SESSION;
CREATE VIEW laurel.employee AS SELECT * FROM laurel.emp;
AUDIT SESSION BY jward, swilliams;
AUDIT ALTER USER;
AUDIT LOCK TABLE
    BY ACCESS
    WHENEVER SUCCESSFUL;
AUDIT DELETE ON laurel.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 laurel.emp IN EXCLUSIVE MODE;
DELETE FROM laurel.emp WHERE mgr = 7698;
ALTER TABLE laurel.emp ALLOCATE EXTENT (SIZE 100K);
CREATE INDEX laurel.ename_index ON laurel.emp (ename);
CREATE PROCEDURE laurel.fire_employee (empid NUMBER) AS
  BEGIN
    DELETE FROM laurel.emp WHERE empno = empid;
  END;
/

EXECUTE laurel.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
```

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 with names that start with the characters `emp` and that are contained in the schema of `laurel`:

```
SELECT * FROM DBA_OBJ_AUDIT_OPTS
    WHERE OWNER = 'LAUREL' AND OBJECT_NAME LIKE 'EMP%';

OWNER   OBJECT_NAME OBJECT_TY ALT AUD COM DEL GRA IND INS LOC ...
-----   ----------- --------- --- --- --- --- --- --- --- --- ...
LAUREL EMP         TABLE     S/S -/- -/- A/- -/- S/S -/- -/- ...
LAUREL EMPLOYEE    VIEW      -/- -/- -/- A/- -/- S/S -/- -/- ...
```

The view returns information about all the audit options for the specified object. The information in the view is interpreted as follows:

- A dash (`-`) indicates that the audit option is not set.
- The `S` character indicates that the audit option is set `BY SESSION`.
- The `A` character indicates that the audit option is set `BY ACCESS`.
- Each audit option has two possible settings, `WHENEVER SUCCESSFUL` and `WHENEVER NOT SUCCESSFUL`, separated by a slash (`/`). For example, the `DELETE` audit option for `laurel.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 REA
--- --- --- --- --- --- --- --- --- --- --- --- --- --- ---
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 (refer to previous example).

Listing Audit Records

The following query lists audit records generated for all objects in the database:

```
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, then delete them by connecting to the database as `SYS` and run the script file `CATNOAUD.SQL`. The location of the `CATNOAUD.SQL` script is operating system-dependent.

7

Using Application Contexts to Retrieve User Information

An application context retrieves information about a user session and then stores this information in cache so that it is easily accessible during the user session. You can use this information as a basis on which to permit or prevent that user from accessing data.

This chapter discusses the following topics:

- About Application Contexts
- Types of Application Contexts
- Using Database Session-Based Application Contexts
- Using Global Application Contexts
- Using Client Session-Based Application Contexts
- Finding Information About Application Contexts

About Application Contexts

An application context is a name-value pair that enables an application to access session information about a user, such as the user ID or other user-specific information, or a client ID, and then securely pass this data to the database. You can then use this information to either permit or prevent the user from accessing data through the application.

The name-value pair can be summarized as follows:

- **Name**: Refers to the name (called a *namespace*) of the application context. For example, a namespace for an application context that retrieves user data from an underlying order entry table could be called `OE_CTX`.
- **Value**: Refers to a value set by an attribute of the namespace. For example, for the `OE_CTX` namespace, if you wanted to retrieve a customer ID from the order entry table, you could create an attribute called `CUSTOMER_NUMBER` that can set the value for this ID.

Think of an application context as a global variable that holds information that is accessed during a database session. To set the values for the application context, you must create a PL/SQL package procedure. In fact, it is the *only* way that you can set application context values. The procedure assigns the values for the application context attributes at run time, not when you create the application context. Because the procedure, and not the user, assigns the values, it is a *trusted* procedure. This method enables you to better secure database access.

Oracle Database stores the application context values in a secure data cache available in the User Global Area (UGA) or the System (sometimes called "Shared") Global Area (SGA). This way, the application context values are retrieved during the session. Because the application context stores the values in this data cache, it increases performance for your applications. You can use an application context by itself, with Oracle Virtual Private Databases policies, or with other fine-grained access control policies. See "Using Oracle Virtual Private Database with an Application Context" on page 8-3 if you are interested in using application contexts with Virtual Private Database policies.

Most applications contain the kind of information that can be used for application contexts. For example, in an order entry application that uses a table containing the columns `ORDER_NUMBER` and `CUSTOMER_NUMBER`, you can use the values in these columns as security attributes to restrict access by a customer to his or her own orders, based on the ID of that customer.

Application contexts are useful for the following purposes:

- Enforcing fine-grained access control, for example, in Oracle Virtual Private Database polices
- Preserving user identity across multitier environments
- Enforcing stronger security for your applications, because the application context is controlled by a trusted procedure, not the user
- Increasing performance by serving as a secure data cache for attributes needed by an application for fine-grained auditing or for use in PL/SQL conditional statements or loops

  This cache saves the repeated overhead of querying the database each time these attributes are needed. Because the application context stores session data in cache rather than forcing your applications to retrieve this data repeatedly from a table, it greatly improves the performance of your applications.
- Serving as a holding area for name-value pairs that an application can define, modify, and access

Types of Application Contexts

There are three general categories of application contexts:

- **Database session-based application contexts.** This type retrieves data that is stored in the database user session (that is, the UGA) cache. There are three categories of database session-based application contexts:
  - **Initialized locally**. Initializes the application context locally, to the session of the user.
  - **Initialized externally**. Initializes the application context from an Oracle Call Interface (OCI) interface, a job queue process, or a connected user database link.
  - **Initialized globally**. Uses attributes and values from a centralized location, such as an LDAP directory.

  "Using Database Session-Based Application Contexts" on page 7-3 describes this type of application context.
- **Global application contexts.** This type retrieves data that is stored in the System Global Area (SGA) so that it can be used for applications that use a sessionless model, such as middle-tier applications in a three-tiered architecture. A global

application context is useful if the session context needs to be shared across sessions, for example, through connection pool implementations.

"Using Global Application Contexts" on page 7-20 describes this type.

- **Client session-based application contexts.** This type uses Oracle Call Interface functions on the client side to set the user session data, and then to perform the necessary security checks to restrict user access.

  "Using Client Session-Based Application Contexts" on page 7-38 describes this type.

Table 7–1 summarizes the different types of application contexts.

Table 7–1 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
Database session-based application context initialized locally	Yes	No	No	No	No
Database session-based application context initialized externally	Yes	No	Yes	No	No
Database session-based application context initialized globally	Yes	No	No	Yes	No
Global application context	No	Yes	No	No	Yes
Client session-based application context	Yes	No	Yes	No	Yes

Using Database Session-Based Application Contexts

This section explores the following topics:

- About Database Session-Based Application Contexts
- Creating a Database Session-Based Application Context
- Creating a PL/SQL Package to Set the Database Session-Based Application Context
- Creating a Logon Trigger to Run a Database Session Application Context Package
- Example of Creating and Using a Database Session-Based Application Context
- Initializing Database Session-Based Application Contexts Externally
- Initializing Database Session-Based Application Contexts Globally
- Using Externalized Database Session-Based Application Contexts

About Database Session-Based Application Contexts

If you need to retrieve session information for database users, use a database session-based application context. This type of application context uses a PL/SQL procedure within Oracle Database to retrieve, set, and secure the data it manages.

> **Note:** If your users are application users, that is, users who are not in your database, consider using a global application context instead. See "Using Global Application Contexts" on page 7-20 for more information.

The database session-based application context is managed entirely within Oracle Database. Oracle Database sets the values, and then when the user exits the session, automatically clears the application context values stored in cache. If the user connection ends abnormally, for example, during a power failure, then the PMON background process cleans up the application context data.You do not need to explicitly clear the application context from cache.

The advantage of having Oracle Database manage the application context is that you can centralize the application context management. Any application that accesses this database will need to use this application context to permit or prevent user access to that application. This provides benefits both in improved performance and stronger security.

You use the following components to create and use a database session-based application context:

- **The application context.** You use the `CREATE CONTEXT` SQL statement to create an application context. This statement names the application context (namespace) and associates it with a PL/SQL procedure that is designed to retrieve session data and set the application context.
- **A PL/SQL procedure to perform the data retrieval and set the context.** "About the Package That Manages the Database Session-Based Application Context" on page 7-6 describes the tasks this procedure must perform. Ideally, create this procedure within a package, so that you can include other procedures if you want, for example, to perform error checking tasks.
- **A way to set the application context when the user logs on.** Users who log on to applications that use the application context must run a PL/SQL package that sets the application context. You can achieve this with either a logon trigger that fires each time the user logs on, or you can embed this functionality in your applications.

"Example of Creating and Using a Database Session-Based Application Context" on page 7-11 shows how to create and use a database session-based application context that is initialized locally.

You can also initialize session-based application contexts either externally or globally. Either method stores the context information in the user session.

- **External initialization.** This type can come from an OCI interface, a job queue process, or a connected user database link. See "Initializing Database Session-Based Application Contexts Externally" on page 7-14 for detailed information.
- **Global initialization.** This type uses attributes and values from a centralized location, such as an LDAP directory. "Initializing Database Session-Based Application Contexts Globally" on page 7-16 provides more information.

Creating a Database Session-Based Application Context

To create a database session-based application context, you use the CREATE CONTEXT PL/SQL statement. Here, you create a namespace for the application context and then associate it with a PL/SQL package that manages the name-value pair that holds the session information of the user. You must have the CREATE ANY CONTEXT system privilege to run this statement, and the DROP ANY CONTEXT privilege to use the DROP CONTEXT statement if you drop the application context. In a database session-based application context, data is stored in the database user session (UGA) in a namespace that you create with the CREATE CONTEXT SQL statement.

Each application 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.

The ownership of the application context is as follows: Even though a user who has been granted the CREATE ANY CONTEXT and DROP ANY CONTEXT privileges can create and drop the application context, it is owned by the SYS schema. Oracle Database associates the context with the schema account that created it, but if you drop this user, the context still exists in the SYS schema. As user SYS, you can drop the application context.

Example 7–1 shows how to use CREATE CONTEXT to create a database session-based application context:

Example 7–1 Creating a Database Session-Based Application Context

```
CREATE CONTEXT empno_ctx USING set_empno_ctx_pkg;
```

Here, empno_ctx is the context namespace and set_empno_ctx_pkg is the package that sets attributes for the empno_ctx namespace. When you create the application context, the PL/SQL package does not need to exist, but it must exist at run time. "Step 3: Create a Package to Retrieve Session Data and Set the Application Context" on page 7-12 shows an example of how to create a package that can be used with this application context.

Notice that when you create the context, you do not set its name-value attributes in the CREATE CONTEXT statement. Instead, you set these in the package that you associate with the application context. The reason you do this is to prevent a malicious user from changing the context attributes without proper attribute validation.

> **Note:** You cannot create a context called CLIENTCONTEXT. This word is reserved for use with client session-based application contexts. See "Using Client Session-Based Application Contexts" on page 7-38 for more information about this type of application context.

For each application, you can create an application context that has 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:

- For the order entry application context, you can specify the attribute CUSTOMER_NUMBER.
- For the general ledger application context, you can specify the attributes SET_OF_BOOKS and TITLE.
- For the human resources application context, you can specify the attributes ORGANIZATION_ID, POSITION, and COUNTRY.

The data the attributes access is stored in the tables behind the applications. For example, the order entry application uses a table called OE.CUSTOMERS, which contains the CUSTOMER_NUMBER column, which provides data for the CUSTOMER_NUMBER attribute. In each case, you can adapt the application context to your precise security needs.

Creating a PL/SQL Package to Set the Database Session-Based Application Context

This section describes the following topics:

- About the Package That Manages the Database Session-Based Application Context
- Using SYS_CONTEXT to Retrieve Session Information
- Using Dynamic SQL with SYS_CONTEXT
- Using SYS_CONTEXT in a Parallel Query
- Using SYS_CONTEXT with Database Links
- Using DBMS_SESSION.SET_CONTEXT to Set Session Information

About the Package That Manages the Database Session-Based Application Context

The PL/SQL package, usually created in the schema of the security administrator, defines procedures that manage the session data represented by the application context. It must perform the following tasks:

- **Retrieve session information.** To retrieve the user session information, you can use the SYS_CONTEXT SQL function. The SYS_CONTEXT function returns the value of parameter associated with the context namespace. You can use this function in both SQL and PL/SQL statements. Typically, you will use the built-in USERENV namespace to retrieve the session information of a user.
- **Set the name-value attributes of the application context you created with CREATE CONTEXT.** You can use the DBMS_SESSION.SET_CONTEXT procedure to set the name-value attributes of the application context. The name-value attributes can hold information such as the user ID, IP address, authentication mode, the name of the application, and so on. The values of the attributes you set remain either until you reset them, or until the user ends the session.
- **Be executed by users.** After you create the package, the user will need to execute the package when he or she logs on. You can create a logon trigger to execute the package automatically when the user logs on, or you can embed this functionality in your applications. Remember that the application context session values are cleared automatically when the user ends the session, so you do not need to manually remove the session data.

It is important to remember that the procedure is a trusted procedure: It is designed to prevent the user from setting his or her own application context attribute values. The user runs the procedure, but the procedure sets the application context values, not the user.

"Example of Creating and Using a Database Session-Based Application Context" on page 7-11 shows how to create a database session-based application context.

Using SYS_CONTEXT to Retrieve Session Information

The syntax for the PL/SQL function SYS_CONTEXT is as follows:

```
SYS_CONTEXT ('namespace','parameter'[,length])
```

In this specification:

- *namespace*: The name of the application context. You can specify either a string or an expression.
- *parameter*: A parameter within the *namespace* application context.
- *length*: Optional. The default maximum size of the return type is 256 bytes, but you can override the length by specifying a value up to 4000 bytes. Enter a value that is a NUMBER data type, or a value that can be can be implicitly converted to NUMBER. The data type of the SYS_CONTEXT return type is a VARCHAR2.

The SYS_CONTEXT function provides a default namespace, USERENV, which describes the current session of the user logged on. You can use SYS_CONTEXT to retrieve different types of session-based information about a user, such as the user host computer ID, IP address, operating system user name, and so on. Remember that you only use USERENV to *retrieve* session data, not *set* it. The predefined attributes are listed in the description for the SYS_CONTEXT PL/SQL function in the *Oracle Database SQL Language Reference*.

For example, to retrieve the name of the host computer to which a client is connected, you can use the HOST parameter of USERENV as follows:

```
SYS_CONTEXT ('userenv','host')
```

You can check the SYS_CONTEXT settings by issuing a SELECT SQL statement on the DUAL table. The DUAL table is a small table in the data dictionary that Oracle Database and user-written programs can reference to guarantee a known result. This table has one column called DUMMY and one row that contains the value X.

Example 7–2 demonstrates how to find the host computer on which you are logged, assuming that you are logged on to the SHOBEEN_PC host computer under EMP_USERS.

Example 7–2 Finding SYS_CONTEXT Values

```
SELECT SYS_CONTEXT ('userenv', 'host') FROM dual;

SYS_CONTEXT(USERENV,HOST)
-------------------------
EMP_USERS\SHOBEEEN_PC
```

> **Note:** The USERENV application context namespace replaces the USERENV function provided in earlier Oracle Database releases.

Using Dynamic SQL with SYS_CONTEXT

During a session in which you expect a change in policy between executions of a given query, the 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 parsed again at execution time 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 run the statement. With static SQL, Policy A remains in force. Oracle Database parses the statement at compile time, but does not

parse it again upon execution. With dynamic SQL, Oracle Database parses the statement upon execution, then 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 the database user name. It is represented in the form of a SQL predicate in Oracle Virtual Private Database: the predicate is considered a policy. If the predicate changes, then the statement must be parsed again to produce the correct result.

> **See Also:** "Using Automatic Reparsing for Fine-Grained Access Control Policy Functions" on page 8-29

Using SYS_CONTEXT in a Parallel Query

If SYS_CONTEXT is used inside a SQL function that is embedded in a parallel query, then the function includes the application context.

Consider a user-defined function within a SQL statement, which sets the user ID to 5:

```
CREATE FUNCTION set_id
 RETURN NUMBER IS
BEGIN
 IF SYS_CONTEXT ('hr', 'id') = 5
   THEN RETURN 1; ELSE RETURN 2;
 END IF;
END;
```

Now consider the following statement:

```
SELECT * FROM emp WHERE set_id( ) = 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 child processes).

Using SYS_CONTEXT with Database Links

When SQL statements within a user session involve database links, then Oracle Database runs the SYS_CONTEXT SQL function at the host computer of the database link, and then captures the context information there (at the host computer).

If remote PL/SQL procedure calls are run on a database link, then Oracle Database runs any SYS_CONTEXT function inside such a procedure at the destination database of the link. In this case, only externally initialized application contexts are available at the database link destination site. For security reasons, Oracle Database propagates only the externally initialized application context information to the destination site from the initiating database link site.

Using DBMS_SESSION.SET_CONTEXT to Set Session Information

After you have used the SYS_CONTEXT function to retrieve the session data of a user, you are ready to set the application context values from the session of this user. To do so, use the DBMS_SESSION.SET_CONTEXT procedure. (Ensure that you have EXECUTE privileges for the DBMS_SESSION PL/SQL package.)

Its syntax is as follows:

```
DBMS_SESSION.SET_CONTEXT (
   namespace VARCHAR2,
```

```
  attribute VARCHAR2,
  value     VARCHAR2,
  username  VARCHAR2,
  client_id VARCHAR2);
```

In this specification:

- `namespace`: The namespace of the application context to be set, limited to 30 bytes. For example, if you were using a namespace called `custno_ctx`, you would specify it as follows:

  ```
  namespace => 'empno_ctx',
  ```

- `attribute`: The attribute of the application context to be set, limited to 30 bytes. For example, to create the `ctx_attrib` attribute for the `custno_ctx` namespace:

  ```
  attribute => 'ctx_attrib',
  ```

- `value`: The value of the application context to be set, limited to 4000 bytes. Typically, this is the value retrieved by the `SYS_CONTEXT` function and stored in a variable. For example:

  ```
  value => ctx_value,
  ```

- `username`: Optional. The database user name attribute of the application context. The default is `NULL`, which permits any user to access the session. For database session-based application contexts, omit this setting so that it uses the `NULL` default.

  The `username` and `client_id` parameters are used for globally accessed application contexts. See "Setting the username and client_id DBMS_SESSION.SET_CONTEXT Parameters" on page 7-22 for more information.

- `client_id`: Optional. The application-specific `client_id` attribute of the application context (64-byte maximum). The default is `NULL`, which means that no client ID is specified. For database session-based application contexts, omit this setting so that it uses the `NULL` default.

See *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `DBMS_SESSION` package.

For example, remember the application context created in Example 7–1 on page 7-5:

```
CREATE CONTEXT empno_ctx USING set_empno_ctx_proc;
```

Example 7–3 shows how to create a simple procedure that creates an attribute for the `empno_ctx` application context.

Example 7–3 Simple Procedure to Create an Application Context Value

```
SQL> CREATE OR REPLACE PROCEDURE set_empno_ctx_proc(
  2   emp_value IN VARCHAR2)
  3 IS
  4 BEGIN
  5  DBMS_SESSION.SET_CONTEXT('empno_ctx', 'empno_attrib', emp_value);
  6 END;
  7 /
```

In this example:

- **Line 2**: Takes `emp_value` as the input parameter. This parameter specifies the value associated with the application context attribue `empno_attrib`. Its limit is 4000 bytes.
- **Line 5**: Sets the value of the application context by using the `DBMS_SESSION.SET_CONTEXT` procedure:
  - `'empno_ctx'`: Refers to the application context namespace. Enclose its name in single quotation marks.
  - `'empno_attrib'`: Creates the attribute associated with the application context namespace.
  - `ctx_value`: Specifies the value for the `empno_attrib` attribute. Here, it refers to the `ctx_value` parameter defined in **Line 2**.

At this stage, you can run the `set_empno_ctx_proc` procedure to set the application context:

```
EXECUTE set_empno_ctx_proc ('42783');
```

(In a real world scenario, you would set the application context values in the procedure itself, so that it becomes a trusted procedure. This example is only used to show how data can be set for demonstration purposes.)

To check the application context setting, run the following `SELECT` statement:

```
SELECT SYS_CONTEXT ('empno_ctx', 'empno_attrib') empno_attrib FROM DUAL;

EMPNO_ATTRIB
--------------
42783
```

You can also query the `SESSION_CONTEXT` data dictionary view to find all the application context settings in the current session of the database instance. For example:

```
SELECT * FROM SESSION_CONTEXT;

NAMESPACE                ATTRIBUTE          VALUE
-------------------------------------------------
EMPNO_CTX                EMP_ID             42783
```

See Also:

- "Example of Creating and Using a Database Session-Based Application Context" on page 7-11 for how to create package that retrieves the user session information and then sets the application context based on this information
- *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `DBMS_SESSION.SET_CONTEXT` procedure

Creating a Logon Trigger to Run a Database Session Application Context Package

After you create the application context and its associated package, the user needs to run the package when he or she logs on. You can create a logon trigger that handles this automatically. You do not need to grant the user `EXECUTE` permissions to run the package.

For example:

```
CREATE OR REPLACE TRIGGER set_empno_ctx_trig AFTER LOGON ON DATABASE
 BEGIN
  sec_mgr.set_empno_ctx_proc;
 END;
```

Remember that logon triggers may affect performance. In addition, test the logon trigger on a sample schema user first before creating it for the database. That way, if there is an error, you can easily correct it.

Be aware of situations in which 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.

Note: A logon trigger can be used because the user context (information such as `EMPNO`, `GROUP`, `MANAGER`) should be set before the user accesses any data.

Example of Creating and Using a Database Session-Based Application Context

This example shows how to create an application context that limits logins only to anyone who has an e-mail account listed in the `HR.EMPLOYEES` table. The e-mail account is the same as the user account name, so this is used as the basis on which to control user access.

You will follow these steps:

- Step 1: Create User Accounts and Ensure the User SCOTT Is Active
- Step 2: Create the Database Session-Based Application Context
- Step 3: Create a Package to Retrieve Session Data and Set the Application Context
- Step 4: Create a Logon Trigger for the Package
- Step 5: Test the Application Context
- Step 6: Remove the Components for This Example

Step 1: Create User Accounts and Ensure the User SCOTT Is Active

1. Log on as user `SYS` and connect using the `AS SYSDBA` privilege.

   ```
   sqlplus "sys/as sysdba"
   Enter password: password
   ```

2. Create the `sysadmin_ctx` account, who will administer the database session-based application context.

   ```
   GRANT CREATE SESSION, CREATE ANY CONTEXT, CREATE PROCEDURE, CREATE TRIGGER,
   ADMINISTER DATABASE TRIGGER TO sysadmin_ctx IDENTIFIED BY omni2all;

   GRANT SELECT ON hr.employees TO sysadmin_ctx;

   GRANT EXECUTE ON DBMS_SESSION TO sysadmin_ctx;
   ```

3. Create the following user account for Lisa Ozer, who is listed as having `lozer` for her e-mail account in the `HR.EMPLOYEES` table.

   ```
   GRANT CREATE SESSION TO lozer IDENTIFIED BY ready2go;
   ```

4. The sample user SCOTT will also be used in this example, so query the DBA_USERS data dictionary view to ensure that SCOTT is not locked or expired.

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

If the DBA_USERS view lists user SCOTT as locked and expired, then enter the following statement to unlock the SCOTT account and create a new password for him:

```
ALTER USER SCOTT ACCOUNT UNLOCK IDENTIFIED BY tgris86d;
```

The password tgris86d is offered as an example of a valid password, but you can create any password that is secure, according to the requirements described in "How Oracle Database Checks the Complexity of Passwords" on page 3-7.

Step 2: Create the Database Session-Based Application Context

1. Log on to SQL*Plus as sysadmin_ctx.

```
CONNECT sysadmin_ctx
Enter password: omni2all
```

2. Create the application context using the following statement:

```
CREATE CONTEXT empno_ctx USING set_empno_ctx_pkg;
```

Remember that even though user sysadmin_ctx has created this application context, the SYS schema owns the context.

Step 3: Create a Package to Retrieve Session Data and Set the Application Context

Example 7–4 shows how to create the package you need to retrieve the session data and set the application context. Before creating the package, ensure that you are still logged on as user sysadmin_ctx, whose password is omni2all.

Example 7–4 Package to Retrieve Session Data and Set a Database Session Context

```
SQL> CREATE OR REPLACE PACKAGE set_empno_ctx_pkg IS
   2   PROCEDURE set_empno;
   3 END;
   4 /
   5 CREATE OR REPLACE PACKAGE BODY set_empno_ctx_pkg IS
   6   PROCEDURE set_empno
   7   IS
   8    emp_id NUMBER;
   9   BEGIN
  10    SELECT employee_id INTO emp_id FROM hr.employees
  11       WHERE email = SYS_CONTEXT('USERENV', 'SESSION_USER');
  12    DBMS_SESSION.SET_CONTEXT('empno_ctx', 'employee_id', emp_id);
  13   END;
  14 END;
  15 /
```

This package creates a procedure called SET_EMPNO that performs the following actions:

- **Line 8**: Declares a variable, emp_id, to store the employee ID for the user who logs on.
- **Line 10**: Performs a SELECT statement to copy the employee ID that is stored in the employee_id column data from the HR.EMPLOYEES table into the emp_id variable.

- **Line 11**: Uses a `WHERE` clause to find all employee IDs that match the e-mail account for the session user. The `SYS_CONTEXT` function uses the predefined `USERENV` context to retrieve the user session ID, which is the same as the `email` column data. For example, the user ID and e-mail address for Lisa Ozer are both the same: `lozer`.
- **Line 12**: Uses the `DBMS_SESSION.SET_CONTEXT` procedure to set the application context:
  - `'empno_ctx'`: Calls the application context `empno_ctx`. Enclose `empno_ctx` in single quotes.
  - `'employee_id'`: Creates the attribute value of the `empno_ctx` application context name-value pair, by naming it `employee_id`. Enclose `employee_id` in single quotes.
  - `emp_id`: Sets the value for the `employee_id` attribute to the value stored in the `emp_id` variable. The `emp_id` variable was created in **Line 8** and the employee ID was retrieved in **Lines 10–11**.

  To summarize, the `SET_EMPNO_CTX_PKG.SET_EMPNO` procedure says, "Get the session ID of the user, and then find the e-mail account name that matches this user ID. If there is a match, then let the user log on. If not, then deny the user access."

Step 4: Create a Logon Trigger for the Package

As user `SYSADMIN_VPD`, create the following trigger:

```
CREATE TRIGGER set_empno_ctx_trig AFTER LOGON ON DATABASE
 BEGIN
  sysadmin_ctx.set_empno_ctx_pkg.set_empno;
 END;
 /
```

Step 5: Test the Application Context

1. Log on as user `lozer`.

   ```
   CONNECT lozer
   Enter password: ready2go
   Connected.
   ```

   As you can see, `lozer` can log on, because she is listed in the `HR.EMPLOYEES` table. In fact, any user who is listed in the `HR.EMPLOYEES` table can log in.

2. Try to log on as user `SCOTT`.

   ```
   CONNECT SCOTT
   Enter password: tgris86d
   ```

   User `SCOTT` is not listed as an employee in the `HR.EMPLOYEES` table, so he cannot log on. The application context that you created prevents him from doing so. The following error messages appear:

   ```
   ORA-04088: error during execution of trigger 'SYS.SET_EMPNO_CTX_TRIG'
   ORA-01403: no data found
   ORA-06512: at "SYS.SET_EMPNO_CTX_PKG", line 6
   ORA-06512: at line 2
   ```

   However, users with administrative privileges are not affected by the application context, and can still log on.

Step 6: Remove the Components for This Example

1. Log on as `SYS` and connect using `AS SYSDBA`.

   ```
   CONNECT SYS/AS SYSDBA
   Enter password: password
   ```

2. Drop the users `sysadmin_ctx` and `lozer`:

   ```
   DROP USER sysadmin_ctx CASCADE;
   DROP USER lozer CASCADE;
   ```

3. Drop the application context.

   ```
   DROP CONTEXT empno_ctx;
   ```

   Remember that even though `sysadmin_ctx` created the application context, it is owned by the `SYS` schema.

4. If you want, lock and expire `SCOTT`, unless other users want to use this popular account:

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

After you have removed the application context and its associated package and trigger, users should be able to log on to the database instance again.

Initializing Database Session-Based Application Contexts Externally

When you initialize a database session-based application context externally, you specify a special type of namespace that accepts the initialization of attribute values from external resources and then stores them in the local user session. Initializing an application context externally enhances performance because it is stored in the UGA and enables the automatic propagation of attributes from one session to another. Connected user database links are supported only by application contexts initialized from OCI-based external sources.

This section contains these topics:

- Obtaining Default Values from Users
- Obtaining Values from Other External Resources
- Initializing Application Context Values from a Middle-Tier Server

Obtaining Default Values from Users

Sometimes you need the default values from users. Initially, these default values may be hints or preferences, and then after validation, they 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, the 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 the context that takes default values from resources other than the designated PL/SQL procedure by using logon triggers to reset the context when users log in.

Obtaining Values from Other External Resources

You can create an application context that accepts the initialization of attributes and values through external resources. Examples include an OCI interface, a job queue process, or a database link.

Externally initialized application contexts provide the following features:

- For remote sessions, automatic propagation of context values that are in the externally initialized application context namespace
- For job queues, restoration of context values that are in the externally initialized application context namespace
- For OCI interfaces, a mechanism to initialize context values that are in the externally initialized application context namespace

Although any client program that is using Oracle Call Interface can initialize this type of namespace, you can use login event triggers to verify the values. It is up to the application to interpret and trust the values of the attributes.

Example 7–5 shows how to create a database session-based application context that obtains values from an external source.

Example 7–5 Creating an Externalized Database Session-based Application Context

```
CREATE CONTEXT ext_ctx USING ext_ctx_pkg INITIALIZED EXTERNALLY;
```

Initializing Application Context Values from a Middle-Tier Server

Middle-tier servers can initialize application 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.

For example, a three-tier application creating lightweight user sessions through OCI or thick JDBC can access the `PROXY_USER` attribute in `USERENV`. This attribute enables you to determine if the user session was created by a middle-tier application. You could allow a user to access data only for connections where the user is proxied. If users connect directly to the database, then they would not be able to access any data.

You can use the `PROXY_USER` attribute from the `USERENV` namespace within Oracle Virtual Private Database to ensure that users only access data through a particular middle-tier application. For a different approach, you can develop a secure application role to enforce your policy that users access the database only through a specific proxy.

See Also:

- "Preserving User Identity in Multitiered Environments" on page 3-30 for information about proxy authentication and about using the `USERENV` attribute `CLIENT_IDENTIFIER` to preserve user identity across multiple tiers
- "Using a Middle Tier Server for Proxy Authentication" on page 3-31 for information about using a secure application role to enforce a policy through a specific proxy
- *Oracle Database JDBC Developer's Guide and Reference*
- *Oracle Call Interface Programmer's Guide*

Initializing Database Session-Based Application Contexts Globally

You can use a centralized location to store the database session-based application context of the user. This enables applications to set up a user context during initialization based upon user identity. In particular, this feature supports Oracle Label Security labels and privileges. Initializing an application context globally makes it easier 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 needs to retrieve from Lightweight Directory Access Protocol (LDAP) to use for Oracle Virtual Private Database enforcement, such as the user title, organization, or physical location. Initializing an application context globally enables you to retrieve these types of attributes.

This section contains these topics:

- Using Database Session-Based Application Contexts with LDAP
- How Globally Initialized Database Session-Based Application Contexts Work
- Example of Initializing a Database Session-Based Application Context Globally

Using Database Session-Based Application Contexts with LDAP

An application context that is initialized globally uses LDAP, a standard, extensible, and efficient directory access protocol. The LDAP directory stores a list of users to which this application is assigned. Oracle Database uses a directory service, typically Oracle Internet Directory, to authenticate and authorize enterprise users.

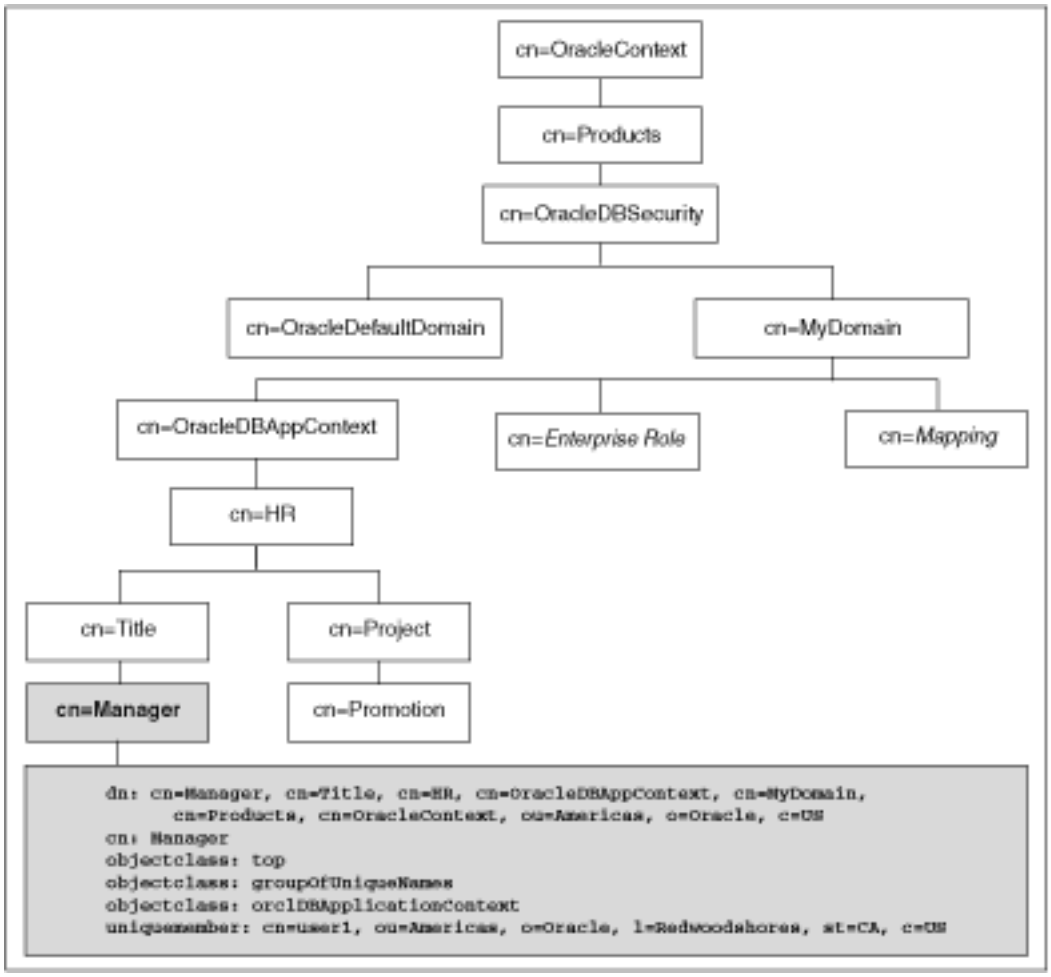
Note:

- Enterprise User Security requires Oracle Advanced Security.
- You can use third-party directories such as Microsoft Active Directory and Sun Microsystems SunONE as the directory service.

The `orclDBApplicationContext` LDAP object (a subclass of `groupOfUniqueNames`) stores the application context values in the directory. The location of the application context object is described in Figure 7–1, which is based on the Human Resources example.

On the LDAP side, an internal C function is required to retrieve the `orclDBApplicationContext` value, which returns a list of application context values to the database. In this example, `HR` is the namespace; Title and Project are the attributes; and Manager and Promotion are the values.

Figure 7–1 Location of Application Context in LDAP Directory Information Tree



How Globally Initialized Database Session-Based Application Contexts Work

To use a globally initialized secure application, you need to first configure Enterprise User Security, a feature of Oracle Advanced Security. Then, you set up the application context values for the user in the database and the directory.

When a global user (enterprise user) connects to the database, Enterprise User Security verifies the identity of the user connecting to the database. After authentication, the global user roles and application context are retrieved from the directory. When the user logs on to the database, the global roles and initial application context are already set.

> **See Also:** *Oracle Database Enterprise User Security Administrator's Guide* for information about configuring Enterprise User Security

Example of Initializing a Database Session-Based Application Context Globally

You can configure and store the initial application context for a user, such as the department name and title, 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 `SYS_USER_DEFAULTS` application context namespace. The following procedure shows how this is accomplished:

1. Create the 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 named `Title` with the attribute value `Manager` for the application (namespace) `HR`, and assign user names `user1` and `user2`.

```
dn:
cn=OracleDBAppContext,cn=myDomain,cn=OracleDBSecurity,cn=Products,cn=OracleCont
ext,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=Orac
leContext,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=Products,cn=OracleCont
ext,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=OracleDBSecurity,cn=Products,cn=OracleCont
ext,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, then the connection retrieves the attributes from `inetOrgPerson`, and assigns 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 the `myDomain` domain, `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 following 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 this example is:

   ```
   Title is Manager
   Telephone Number is +1 650 123 4567
   ```

Using Externalized Database Session-Based 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 also 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 Database Session-Based Application Contexts Externally" on page 7-14 for information about initializing local application context through external resources such as an OCI interface, a job queue process, or a database link
- "Initializing Database Session-Based Application Contexts Globally" on page 7-16 for information about initializing local application context through a centralized resource, such as Oracle Internet Directory
- *Oracle Database Enterprise User Security Administrator's Guide* for information about enterprise users

Using Global Application Contexts

This section explores the following topics:

- About Global Application Contexts
- Creating a Global Application Context
- Creating a PL/SQL Package to Manage a Global Application Context
- Embedding Calls in Middle-Tier Applications to Manage the Client Session ID
- Example of Creating a Global Application Context That Uses a Client Session ID
- Global Application Context Processes

About Global Application Contexts

A global application context enables application context values to be accessible across database sessions. Oracle Database stores the global application context information in the System (sometimes called "Shared") Global Area (SGA) so that it can be used for applications that use a sessionless model, such as middle-tier applications in a three-tiered architecture. These applications cannot use a session-based application context because users authenticate to the application, and then it typically connects to the database as a single identity. Oracle Database initializes the global application context once, rather than for each user session. This improves performance, because connections are reused from a connection pool.

There are three general uses for global application contexts:

- **You need to share application values globally for all database users.** For example, you may need to disable access to an application based on a specific situation. In this case, the values the application context sets are not user-specific, nor are they based on the private data of a user. The application context defines a situation, for example, to indicate the version of application module that is running.
- **You have database users who need to move from one application to another.** In this case, the second application the user is moving to has different access requirements from the first application.
- **You need to authenticate nondatabase users, that is, users who are not known to the database.** This type of user typically connects through a Web application by using a connection pool. These types of applications connect users to the database as single user, using the One Big Application User model. To authenticate this type of user, you use the client session ID of the user.

A global application context has the following components:

- **The global application context.** You use the `CREATE CONTEXT` SQL statement to create the global application context, and include the `ACCESSED GLOBALLY` clause in the statement. This statement names the application context and associates it with a PL/SQL procedure that is designed to set the application data context data. The global application context is created and stored in the database schema of the security administrator who creates it.
- **A PL/SQL package to set the attributes.** The package must contain a procedure that uses the `DBMS_SESSION.SET_CONTEXT` procedure to set the global application context. The `SET_CONTEXT` procedure provides parameters that enable you to create a global application context that fits any of the three user situations described in this section. You create, store, and run the PL/SQL package

on the database server. Typically, it belongs in the schema of the security administrator who created it.

- **A middle-tier application to get and set the client session ID.** For nondatabase users, which require a client session ID to be authenticated, you can use the Oracle Call Interface (OCI) calls in the middle-tier application to retrieve and set their session data. You can also use the `DBMS_SESSION.SET_IDENTIFIER` procedure to set the client session ID.

Creating a Global Application Context

To create a global application context, use the `CREATE CONTEXT` SQL statement to create the application context and include the `ACCESSED GLOBALLY` clause in the statement. You must have the `CREATE ANY CONTEXT` system privilege before you can use the `CREATE CONTEXT` statement, and the `DROP ANY CONTEXT` privilege before you can drop the context with the `DROP CONTEXT` statement. As with local application contexts, the global application context is created and stored in the database schema of a security administrator.

The ownership of the global application context is as follows: Even though a user who has been granted the `CREATE ANY CONTEXT` and `DROP ANY CONTEXT` privileges can create and drop the global application context, it is owned by the `SYS` schema. Oracle Database associates the context with the schema account that created it, but if you drop this user, the context still exists in the `SYS` schema. As user `SYS`, you can drop the application context.

Example 7–6 shows how to create the global application context `global_hr_ctx`, which is set by the `hr_ctx_pkg` package.

Example 7–6 Creating a Global Application Context

```
CREATE OR REPLACE CONTEXT global_hr_ctx USING hr_ctx_pkg ACCESSED GLOBALLY;
```

Creating a PL/SQL Package to Manage a Global Application Context

This section includes the following topics:

- About the Package That Manages the Global Application Context
- Setting the username and client_id DBMS_SESSION.SET_CONTEXT Parameters
- Sharing Global Application Context Values for All Database Users
- Setting a Global Context for Database Users Who Move Between Applications
- Setting a Global Application Context for Nondatabase Users
- Clearing Session Data When the Session Closes

For detailed information about the `DBMS_SESSION` package, see *Oracle Database PL/SQL Packages and Types Reference*.

About the Package That Manages the Global Application Context

The task of the PL/SQL package that you associate with a global application context is to use the `DBMS_SESSION` package to set and clear the global application context values. You must have `EXECUTE` privileges for the `DBMS_SESSION` package before you use its procedures. Typically, you create and store this package in the database schema of a security administrator. The `SYS` schema owns the `DBMS_SESSION` package.

Unlike PL/SQL packages used to set a local application context, you do not include a SYS_CONTEXT function to get the user session data. You do not need to include this function because the owner of the session, recorded in the USERENV context, is the same for every user who is connecting.

You can run the procedures within the PL/SQL package for a global application context at any time. You do not need to create logon and logoff triggers to execute the package procedures associated with the global application context. A common practice is to run the package procedures from within the database application. Additionally, for nondatabase users, you use middle-tier applications to get and set client session IDs.

Setting the username and client_id DBMS_SESSION.SET_CONTEXT Parameters

In addition to the namespace, attribute, and value parameters, the DBMS_SESSION.SYS_CONTEXT procedure provides the client_id and username parameters. Use these settings for global application contexts. Table 7–2 explains how the combination of these settings controls the type of global application context you can create.

Table 7–2 Setting the DBMS_SESSION.SET_CONTEXT username and client_id Parameters

Combination Settings	Result
username set to NULL client_id set to NULL	This combination enables all users to access the application context. See "Sharing Global Application Context Values for All Database Users" on page 7-22 for more information. These settings are also used for database session-based application contexts. See "Using Database Session-Based Application Contexts" on page 7-3 for more information.
username set to a value client_id set to NULL	This combination enables an application context to be accessed by multiple sessions, as long as the username setting is the same throughout. Ensure that the user name specified is a valid database user. See "Setting a Global Context for Database Users Who Move Between Applications" on page 7-24 for more information.
username set to NULL client_id set to a value	This combination enables an application to be accessed by multiple user sessions, as long as the client_id parameter is set to the same value throughout. This enables sessions of all users to see the application context values.
username set to a value client_id set to a value	This combination enables the following two scenarios: ■ **Lightweight users.** If the user does not have a database account, the username specified is a connection pool owner. The client_id setting is then associated with the nondatabase user who is logging in. ■ **Database users.** If the user is a database user, this combination can be used for stateless Web sessions. Setting the username parameter in the SET_CONTEXT procedure to USER calls the Oracle Database-supplied USER function. The USER function specifies the session owner from the application context retrieval process and ensures that only the user who set the application context can access the context. See *Oracle Database SQL Language Reference* for more information about the USER function. See "Setting a Global Application Context for Nondatabase Users" on page 7-25 for more information.

Sharing Global Application Context Values for All Database Users

To share global application values for all database users, set the namespace, attribute, and value parameters in the SET_CONTEXT procedure. In this scenario, *all* users who have database accounts will potentially have access to data in the database.

Example 7–7 shows how to create a package that sets and clears this type of global application context.

Example 7–7 Package to Manage Global Application Values for All Database Users

```
SQL> CREATE OR REPLACE PACKAGE hr_ctx_pkg
  2   AS
  3    PROCEDURE set_hr_ctx(sec_level IN VARCHAR2);
  4    PROCEDURE clear_context;
  5   END;
  6  /
  7  CREATE OR REPLACE PACKAGE BODY hr_ctx_pkg
  8   AS
  9    PROCEDURE set_hr_ctx(sec_level IN VARCHAR2)
 10    AS
 11    BEGIN
 12     DBMS_SESSION.SET_CONTEXT(
 13      namespace  => 'global_hr_ctx',
 14      attribute  => 'job_role',
 15      value      => sec_level);
 16     END set_hr_ctx;
 17
 18   PROCEDURE clear_context
 19    AS
 20    BEGIN
 21     DBMS_SESSION.CLEAR_CONTEXT('global_hr_ctx');
 22    END clear_context;
 23  END;
 24 /
```

In this example:

- **Lines 12–16**: Uses the `DBMS_SESSION.SET_CONTEXT` procedure to set values for the `namespace`, `attribute`, and `value` parameters. The `sec_level` value is specified when the database application runs the `hr_ctx_pkg.set_hr_ctx` procedure.

  The `username` and `client_id` values are not set, hence, they are `NULL`. This enables all users (database users) to have access to the values, which is appropriate for server-wide settings.
- **Line 13**: In the `SET_CONTEXT` procedure, sets the `namespace` to `global_hr_ctx`.
- **Line 14**: Creates the `job_role` attribute.
- **Line 15**: Sets the value for the `job_role` attribute to `sec_level`.
- **Lines 18–22**: Creates the `clear_context` procedure to clear the context values. See "Clearing Session Data When the Session Closes" on page 7-28 for more information.

Typically, you execute this procedure within a database application. For example, if all users logging in are clerks, and you want to use "clerk" as a security level, you would embed a call within a database application similar to the following:

```
BEGIN
 hr_ctx_pkg.set_hr_ctx('clerk');
END;
```

If the procedure successfully completes, you can check the application context setting as follows:

```
SELECT SYS_CONTEXT('global_hr_ctx', 'job_role') job_role FROM DUAL;

JOB_ROLE
-----------
clerk
```

To clear this application context, enter the following:

```
BEGIN
 hr_ctx_pkg.clear_context;
END;
```

To check that it is really cleared, the following `SELECT` statement should return no values:

```
SELECT SYS_CONTEXT('global_hr_ctx', 'job_role') job_role FROM DUAL;

JOB_ROLE
-----------
```

> **Note:** If Oracle Database returns error messages saying that you have insufficient privileges, ensure that you have correctly created the global application context. You should also query the `DBA_CONTEXT` database view to ensure that your settings are correct, for example, that you are calling the procedure from the schema in which you created it.
>
> If `NULL` is returned, then you may have inadvertently set a client identifier. To clear the client identifier, run the following procedure:
>
> ```
> EXEC DBMS_SESSION.CLEAR_IDENTIFIER;
> ```

Setting a Global Context for Database Users Who Move Between Applications

To set a global application context for database users who move from one application to another, particularly when the applications have different access requirements, include the `username` parameter in the `SET_CONTEXT` procedure. This parameter specifies that the same schema be used for all sessions.

Use the following `SET_CONTEXT` parameters:

- `namespace`
- `attribute`
- `value`
- `username`

Oracle Database matches the `username` value so that the other application can recognize the application context. This enables the user to move between applications.

By omitting the `client_id` setting, its value is `NULL`, the default. This means that values can be seen by multiple sessions if the `username` setting is the same for a database user who maintains the same context in different applications. For example, you can have a suite of applications that control user access with Oracle Virtual Private Database policies, with each user restricted to a job role.

Example 7–8 demonstrates how to set the `username` parameter so that a specific user can move between applications. This example is similar to the package that was created in Example 7–7 on page 7-23. The use of the `username` parameter is indicated in **bold** typeface.

Example 7–8 Package to Manage Global Application Context Values for a User Moving Between Applications

```
CREATE OR REPLACE PACKAGE hr_ctx_pkg
  AS
    PROCEDURE set_hr_ctx(sec_level IN VARCHAR2, user_name IN VARCHAR2);
    PROCEDURE clear_context;
   END;
  /
  CREATE OR REPLACE PACKAGE BODY hr_ctx_pkg
   AS
    PROCEDURE set_hr_ctx(sec_level IN VARCHAR2, user_name IN VARCHAR2)
    AS
     BEGIN
      DBMS_SESSION.SET_CONTEXT(
       namespace  => 'global_hr_ctx',
       attribute  => 'job_role',
       value      => sec_level,
       username   => user_name);
      END set_hr_ctx;

   PROCEDURE clear_context
    AS
     BEGIN
      DBMS_SESSION.CLEAR_CONTEXT('global_hr_ctx');
     END clear_context;
  END;
 /
```

Typically, you execute this procedure within a database application by embedding a call similar to the following example. Ensure that the value for the `user_name` parameter (`scott` in this case) is a valid database user name.

```
BEGIN
 hr_ctx_pkg.set_hr_ctx('clerk', 'scott');
END;
```

A secure way to manage this type of global application context is within your applications, embed code to grant a secure application role to the user. This code should include `EXECUTE` permissions on the trusted PL/SQL package that sets the application context. In other words, the application, not the user, will set the context for the user.

Setting a Global Application Context for Nondatabase Users

When a nondatabase user, that is, a user who is not known to the database (such as a Web application user), starts a client session, the application server generates a client session ID. Once this ID is set on the application server, it needs to be passed to the database server side. You do this by using the DBMS_SESSION.SET_IDENTIFIER procedure to set the client session ID. To set the context, you set the `client_id` parameter in the `DBMS_SESSION.SET_CONTEXT` procedure, in a PL/SQL procedure on the server side. This enables you to manage the application context globally, yet each client sees only his or her assigned application context.

The `client_id` value is the key here to getting and setting the correct attributes for the global application context. Remember that the client identifier is controlled by the middle-tier application, and once set, it remains open until it is cleared.

A typical way to manage this type of application context is to place the `session_id` (`client_identifier`) in a cookie, and send it to the end user's HTML page so that

is returned on the next request. A lookup table in the application should also keep client identifiers so that they are prevented from being reused for other users and to implement an end-user session time out.

For nondatabase users, configure the following `SET_CONTEXT` parameters:

- `namespace`
- `attribute`
- `value`
- `username`
- `client_id`

Example 7–9 shows how to create a package that manages this type of global application context.

Example 7–9 Package to Manage Global Application Context Values for Nondatabase Users

```
SQL> CREATE OR REPLACE PACKAGE hr_ctx_pkg
  2  AS
  3   PROCEDURE set_session_id(session_id_p IN NUMBER);
  4   PROCEDURE set_hr_ctx(sec_level_attr IN VARCHAR2, sec_level_val IN VARCHAR2);
  5   PROCEDURE clear_session(session_id_p IN NUMBER);
  6   PROCEDURE clear_context;
  7  END;
  8  /
  9  CREATE OR REPLACE PACKAGE BODY hr_ctx_pkg
 10   AS
 11    session_id_global NUMBER;
 12  PROCEDURE set_session_id(session_id_p IN NUMBER)
 13   AS
 14   BEGIN
 15    session_id_global := session_id_p;
 16    DBMS_SESSION.SET_IDENTIFIER(session_id_p);
 17  END set_session_id;
 18
 19  PROCEDURE set_hr_ctx(sec_level_attr IN VARCHAR2, sec_level_val IN VARCHAR2)
 20   AS
 21   BEGIN
 22    DBMS_SESSION.SET_CONTEXT(
 23     namespace  => 'global_hr_ctx',
 24     attribute  => sec_level_attr,
 25     value      => sec_level_val,
 26     username   => USER,
 27     client_id  => session_id_global);
 28   END set_hr_ctx;
 29
 30  PROCEDURE clear_session(session_id_p IN NUMBER)
 31   AS
 32   BEGIN
 33      DBMS_SESSION.SET_IDENTIFIER(session_id_p);
 34      DBMS_SESSION.CLEAR_IDENTIFIER;
 35   END clear_session;
 36
 37  PROCEDURE clear_context
 38  AS
 39  BEGIN
 40   DBMS_SESSION.CLEAR_CONTEXT('global_hr_ctx', session_id_global);
 41  END clear_context;
```

```
42 END;
43 /
```

In this example:

- **Line 11**: Creates the `session_id_global` variable, which will hold the client session ID. The `session_id_global` variable is referenced throughout the package definition, including the procedure that creates the global application context attributes and assigns them values. This means that the global application context values will always be associated with this particular session ID.
- **Lines 12–17**: Creates the `set_session_id` procedure, which writes the client session ID to the `session_id_global` variable.
- **Lines 19–28**: Creates the `set_hr_ctx` procedure, which creates global application context attributes and enables you to assign values to these attributes. Within this procedure:
  - **Line 26**: Specifies the `username` value. This example sets it by calling the Oracle Database-supplied `USER` function, which adds the session owner from the context retrieval process. The `USER` function ensures that only the user who set the application context can access the context. See *Oracle Database SQL Language Reference* for more information about the `USER` function.

    If you had specified `NULL` (the default for the `username` parameter), then any user can access the context.

    Setting both the `username` and `client_id` values enables two scenarios. For lightweight users, set the `username` parameter to a connection pool owner (for example, `APPS_USER`), and then set `client_id` to the client session ID. If you want to use a stateless Web session, set the `user_name` parameter to the same database user who has logged in, and ensure that this user keeps the same client session ID. See "Setting the username and client_id DBMS_SESSION.SET_CONTEXT Parameters" on page 7-22 for an explanation of how different `username` and `client_id` settings work.
  - **Line 27**: Specifies `client_id` value. This example sets it to the `session_id_global` variable. This associates the context settings defined here with a specific client session ID, that is, the one that is set when you run the `set_session_id` procedure. If you specify the `client_id` parameter default, `NULL`, then the global application context settings could be used by any session.
- **Lines 30–35**: Creates the `clear_session` procedure to clear the client session identifier. **Line 33** sets it to ensure that you are clearing the correct session ID, that is, the one stored in variable session_id_p defined in **Line 10**.
- **Lines 37–42**: Creates the `clear_context` procedure, so that you can clear the context settings for the current user session, which were defined by the `global_hr_ctx` variable. See "Clearing Session Data When the Session Closes" on page 7-28 for more information.

See Also:

- "Example of Creating a Global Application Context That Uses a Client Session ID" on page 7-31 for a tutorial that demonstrates how a global application context used for client session IDs works
- "Setting the Client Session ID Using a Middle-Tier Application" on page 7-29
- "Using Client Identifiers to Identify Application Users Not Known to the Database" on page 3-38 for information about how client identifiers work on middle-tier systems

Clearing Session Data When the Session Closes

The application context exists entirely within memory. When the user exits a session, you need to clear the context for the `client_identifier` value. This releases memory and prevents other users from accidentally using any left over values.

To clear session data when a user exits a session, use either of the following methods in the server-side PL/SQL package:

- **Clearing the client identifier when a user exits a session.** Use the `DBMS_SESSION.CLEAR_IDENTIFIER` procedure. For example:

  ```
  DBMS_SESSION.CLEAR_IDENTIFIER;
  ```

- **Continuing the session but still clearing the context.** If you want the session to continue, but you still need to clear the context, use the `DBMS_SESSION.CLEAR_CONTEXT` or the `DBMS_SESSION.CLEAR_ALL_CONTEXT` procedure. For example:

  ```
  DBMS_SESSION.CLEAR_CONTEXT(namespace, client_identifier, attribute);
  ```

  The `CLEAR_CONTEXT` procedure clears the context for the current user. To clear the context values for all users, for example, when you need to shut down the application server, use the `CLEAR_ALL_CONTEXT` procedure.

  Global application context values are available until they are cleared, so you should use `CLEAR_CONTEXT` or `CLEAR_ALL_CONTEXT` to ensure that other sessions do not have access to these values.

Embedding Calls in Middle-Tier Applications to Manage the Client Session ID

This section includes the following topics:

- About Managing Client Session IDs Using a Middle-Tier Application
- Retrieving the Client Session ID Using a Middle-Tier Application
- Setting the Client Session ID Using a Middle-Tier Application
- Clearing Session Data Using a Middle-Tier Application

About Managing Client Session IDs Using a Middle-Tier Application

The application server generates the client session ID. From a middle-tier application, you can get, set, and clear the client session IDs. To do so, embed either Oracle Call Interface (OCI) calls or `DBMS_SESSION` PL/SQL package procedures into the middle-tier application code.

The application authenticates the user, sets the client identifier, and sets it in the current session. The PL/SQL package `SET_CONTEXT` sets the `client_identifier`

value in the application context. See "Setting a Global Application Context for Nondatabase Users" on page 7-25 for more information.

Retrieving the Client Session ID Using a Middle-Tier Application

When a user starts a client session, the application server generates a client session ID. To retrieve this client ID, you can use the `OCIStmtExecute` call with any of the following statements:

```
SELECT SYS_CONTEXT('userenv', 'client_identifier') FROM dual;

SELECT CLIENT_IDENTIFIER from V$SESSION;

SELECT value FROM session_context WHERE attribute='CLIENT_IDENTIFIER';
```

Example 7–10 shows how to use the `OCIStmtExecute` call to retrieve a client session ID value.

Example 7–10 Using OCIStmtExecute to Retrieve a Client Session ID Value

```
 1 oratext    clientid[31];
 2 OCIDefine  *defnp1 = (OCIDefine *) 0;
 3 OCIStmt    *statementhndle;
 4 oratext    *selcid = (oratext *)"SELECT SYS_CONTEXT('userenv',
 5             'client_identifier') FROM  DUAL";
 6
 7 OCIStmtPrepare(statementhndle, errhp, selcid, (ub4) strlen((char *) selcid),
 8   (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT);
 9
10 OCIDefineByPos(statementhndle, &defnp1, errhp, 1, (dvoid *)clientid, 31,
11   SQLT_STR, (dvoid *) 0, (ub2 *) 0, (ub2 *) 0, OCI_DEFAULT);
12
13 OCIStmtExecute(servhndle, statementhndle, errhp, (ub4) 1, (ub4) 0,
14   (CONST OCISnapshot *) NULL, (OCISnapshot *) NULL, OCI_DEFAULT);
15
16 printf("CLIENT_IDENTIFIER = %s \n", clientid);
```

In this example:

- **Lines 1–5**: Create variables to store the client session ID, reference call for `OCIDefine`, the statement handle, and the `SELECT` statement to use.
- **Lines 7–8**: Prepare the statement `selcid` for execution.
- **Lines 10–11**: Define the output variable `clientid` for client session ID.
- **Lines 13–14**: Execute the statement in the `selcid` variable.
- **Line 16**: Prints the formatted output for the retrieved client session ID.

Setting the Client Session ID Using a Middle-Tier Application

After you use the `OCIStmtExecute` call to retrieve the client session ID, you are ready to set this ID. The `DBMS_SESSION.SET_CONTEXT` procedure in the server-side PL/SQL package then sets this session ID and optionally, overwrites the application context values.

Ensure that the middle-tier application code checks that the client session ID value (for example, the value written to `user_id` in the previous examples) matches the `client_id` setting defined in the server-side `DBMS_SESSION.SET_CONTEXT` procedure. The sequence of calls on the application server side should be as follows:

1. **Get the current client session ID.** The session should already have this ID, but it is safer to ensure that it truly has the correct value.
2. **Clear the current client session ID.** This prepares the application to service a request from a different end user.
3. **Set the new client session ID or the client session ID that has been assigned to the end user.** This ensures that the session is using a different set of global application context values.

You can use the following methods to set the client session ID on the application server side:

- **Oracle Call Interface.** Set the `OCI_ATTR_CLIENT_IDENTIFIER` attribute in an `OCIAttrSet` OCI call. This attribute sets the client identifier in the session handle to track the end user identity.

  The following example shows how to use `OCIAttrSet` with the `ATTR_CLIENT_IDENTIFIER` parameter. The `user_id` setting refers to a variable that stores the ID of the user who is logging on.

  ```
  OCIAttrSet((void *)session_handle, (ub4) OCI_HTYPE_SESSION,
             (void *) user_id, (ub4)strlen(user_id),
             OCI_ATTR_CLIENT_IDENTIFIER, error_handle);
  ```

- **DBMS_SESSION package.** Use the `DBMS_SESSION.SET_IDENTIFIER` procedure to set the client identifier for the global application context. For example, assuming you are storing the ID of the user logging on in a variable called `user_id`, you would enter the following line into the middle-tier application code:

  ```
  DBMS_SESSION.SET_IDENTIFIER(user_id);
  ```

  Note: When the application generates a session ID for use as a `CLIENT_IDENTIFIER`, then the session ID must be suitably random and protected over the network by encryption. If the session ID is not random, then a malicious user could guess the session ID and access the data of another user. If the session ID is not encrypted over the network, then a malicious user could retrieve the session ID and access the connection.

  You can encrypt the session ID by using Oracle Advanced Security. See *Oracle Database Advanced Security Administrator's Guide* for more information. To learn more about encrypting data over a network, see *Oracle Database 2 Day + Security Guide*.

For both `OCIAttrSet` and `DBMS_SESSION.SET_IDENTIFIER`, you can check the value of this identifier as follows:

```
SELECT SYS_CONTEXT('userenv', 'client_identifier') FROM dual;
```

Another way to check this value is to query the `V$SESSION` view:

```
SELECT CLIENT_IDENTIFIER from V$SESSION;
```

Clearing Session Data Using a Middle-Tier Application

The application context exists entirely within memory. When the user exits a session, you need to clear the context for the `client_identifier` value. This releases memory and prevents other users from accidentally using any left over values

To clear session data when a user exits a session, use either of the following methods in the middle-tier application code:

- **Clearing the client identifier when a user exits a session.** Use the DBMS_SESSION.CLEAR_IDENTIFIER procedure. For example:

  ```
  DBMS_SESSION.CLEAR_IDENTIFIER;
  ```

- **Continuing the session but still clearing the context.** If you want the session to continue, but you still need to clear the context, use the DBMS_SESSION.CLEAR_CONTEXT or the DBMS_SESSION.CLEAR_ALL_CONTEXT procedure. For example:

  ```
  DBMS_SESSION.CLEAR_CONTEXT(namespace, client_identifier, attribute);
  ```

  The CLEAR_CONTEXT procedure clears the context for the current user. To clear the context values for all users, for example, when you need to shut down the application server, use the CLEAR_ALL_CONTEXT procedure.

  Global application context values are available until they are cleared, so you should use CLEAR_CONTEXT or CLEAR_ALL_CONTEXT to ensure that other sessions do not have access to these values.

Example of Creating a Global Application Context That Uses a Client Session ID

This example shows how to create a global application context that uses a client session ID for a lightweight user application. It demonstrates how to control user access by using a connection pool.

Follow these steps:

- Step 1: Create User Accounts
- Step 2: Create the Global Application Context
- Step 3: Create a Package for the Global Application Context
- Step 4: Test the Global Application Context
- Step 5: Remove the Components for This Example

Step 1: Create User Accounts

You need to create two users for this example: a security administrator who will manage the application context and its package, and a user account that owns the connection pool.

Follow these steps:

1. Log on to SQL*Plus as SYS and connect using AS SYSDBA.

   ```
   sqlplus SYS/AS SYSDBA
   Enter password: password
   ```

2. Create the sysadmin_ctx account, who will administer the global application context.

   ```
   GRANT CREATE SESSION, CREATE ANY CONTEXT, CREATE PROCEDURE TO sysadmin_ctx
   IDENTIFIED BY omni2all;

   GRANT EXECUTE ON DBMS_SESSION TO sysadmin_ctx;
   ```

3. Create the account apps_user, who will own the connection pool.

   ```
   GRANT CREATE SESSION TO apps_user IDENTIFIED BY ready2go;
   ```

Step 2: Create the Global Application Context

1. Log on as the security administrator `sysadmin_ctx`.

```
CONNECT sysadmin_ctx
Enter password: omni2all
```

2. Create the `cust_ctx` global application context.

```
CREATE CONTEXT global_cust_ctx USING cust_ctx_pkg ACCESSED GLOBALLY;
```

The `cust_ctx` context is created and associated with the schema of the security administrator `sysadmin_ctx`. However, the `SYS` schema owns the application context.

Step 3: Create a Package for the Global Application Context

1. As `sysadmin_ctx`, create the following PL/SQL package:

```
CREATE OR REPLACE PACKAGE cust_ctx_pkg
  AS
   PROCEDURE set_session_id(session_id_p IN NUMBER);
   PROCEDURE set_cust_ctx(sec_level_attr IN VARCHAR2,
     sec_level_val IN VARCHAR2);
   PROCEDURE clear_session(session_id_p IN NUMBER);
   PROCEDURE clear_context;
  END;
 /
CREATE OR REPLACE PACKAGE BODY cust_ctx_pkg
  AS
  session_id_global NUMBER;

 PROCEDURE set_session_id(session_id_p IN NUMBER)
  AS
  BEGIN
   session_id_global := session_id_p;
   DBMS_SESSION.SET_IDENTIFIER(session_id_p);
 END set_session_id;

 PROCEDURE set_cust_ctx(sec_level_attr IN VARCHAR2, sec_level_val IN VARCHAR2)
  AS
  BEGIN
   DBMS_SESSION.SET_CONTEXT(
    namespace  => 'global_cust_ctx',
    attribute  => sec_level_attr,
    value      => sec_level_val,
    username   => USER, -- Retrieves the session user, in this case, apps_user
    client_id  => session_id_global);
  END set_cust_ctx;

  PROCEDURE clear_session(session_id_p IN NUMBER)
   AS
   BEGIN
     DBMS_SESSION.SET_IDENTIFIER(session_id_p);
     DBMS_SESSION.CLEAR_IDENTIFIER;
   END clear_session;

 PROCEDURE clear_context
  AS
  BEGIN
   DBMS_SESSION.CLEAR_CONTEXT('global_cust_ctx', session_id_global);
  END clear_context;
```

```
 END;
/
```

For a detailed explanation of how this type of package works, see Example 7–9 on page 7-26.

2. Grant EXECUTE privileges on the cust_ctx_pkg package to the connection pool owner, apps_user.

```
GRANT EXECUTE ON cust_ctx_pkg TO apps_user;
```

Step 4: Test the Global Application Context

At this stage, you are ready to explore how this global application context and session ID settings work.

1. Log on to SQL*Plus as the connection pool owner, user apps_user.

```
CONNECT apps_user
Enter password: ready2go
```

2. When the connection pool user logs on, the application sets the client session identifier as follows:

```
BEGIN
 sysadmin_ctx.cust_ctx_pkg.set_session_id(34256);
END;
```

You can test and check the value of the client session identifier as follows:

a. Log on as the connection pool user apps_user.

b. Set the session ID:

```
EXEC sysadmin_ctx.cust_ctx_pkg.set_session_id(34256);
```

c. Check the session ID:

```
SELECT SYS_CONTEXT('userenv', 'client_identifier') FROM dual;

SYS_CONTEXT('USERENV','CLIENT_IDENTIFIER')
--------------------------------------------------
34256
```

3. As user apps_user, set the global application context as follows:

```
EXEC sysadmin_ctx.cust_ctx_pkg.set_cust_ctx('Category', 'Gold Partner');
EXEC sysadmin_ctx.cust_ctx_pkg.set_cust_ctx('Benefit Level', 'Highest');
```

(In a real-world scenario, the middle-tier application would set the global application context values, similar to how the client session identifier was set in Step 2.)

4. Enter the following SELECT SYS_CONTEXT statement to check that the settings were successful:

```
col category format a13
col benefit_level format a14

SELECT SYS_CONTEXT('global_cust_ctx', 'Category') category, SYS_
CONTEXT('global_cust_ctx', 'Benefit Level') benefit_level FROM dual;

CATEGORY       BENEFIT_LEVEL
-------------  --------------
```

```
Gold Partner   Highest
```

What apps_user has done here, within the client session 34256, is set a global application context on behalf of a nondatabase user. This context sets the Category and Benefit Level DBMS_SESSION.SET_CONTEXT attributes to be Gold Partner and Highest, respectively. The context exists only for user apps_user with client ID 34256. When a nondatabase user logs in, behind the scenes, he or she is really logging on as the connection pool user apps_user. Hence, the Gold Partner and Highest context values are available to the nondatabase user.

Suppose the user had been a database user and could log in without using the intended application. (For example, the user logs in using SQL*Plus.) Because the user has not logged in through the connection pool user apps_user, the global application context appears empty to our errant user. This is because the context was created and set under the apps_user session. If the user runs the SELECT SYS_CONTEXT statement, the following output appears:

```
CATEGORY       BENEFIT_LEVEL
-------------  --------------
```

Next, try the following test:

1. As user apps_user, clear the session ID.

   ```
   EXEC sysadmin_ctx.cust_ctx_pkg.clear_session(34256);
   ```

2. Check the global application context settings again.

   ```
   SELECT SYS_CONTEXT('global_cust_ctx', 'Category') category, SYS_
   CONTEXT('global_cust_ctx', 'Benefit Level') benefit_level FROM dual;

   CATEGORY       BENEFIT_LEVEL
   -------------  --------------
   ```

   Because apps_user has cleared the session ID, the global application context settings are no longer available.

3. Restore the session ID to 34256, and then check the context values.

   ```
   EXEC sysadmin_ctx.cust_ctx_pkg.set_session_id(34256);

   SELECT SYS_CONTEXT('global_cust_ctx', 'Category') category, SYS_
   CONTEXT('global_cust_ctx', 'Benefit Level') benefit_level FROM dual;

   CATEGORY       BENEFIT_LEVEL
   -------------  --------------
   Gold Partner   Highest
   ```

   As you can see, resetting the session ID to 34256 brings the application context values back again. To summarize, the global application context needs to be set only *once* for this user, but the client session ID needs to be set *each time* the user logs on.

4. Now try clearing and then checking the global application context values.

   ```
   EXEC sysadmin_ctx.cust_ctx_pkg.clear_context;

   SELECT SYS_CONTEXT('global_cust_ctx', 'Category') category, SYS_
   CONTEXT('global_cust_ctx', 'Benefit Level') benefit_level FROM dual;

   CATEGORY       BENEFIT_LEVEL
   -------------  --------------
   ```

At this stage, the client session ID, 34256 is still in place, but the application context settings no longer exist. This enables you to continue the session for this user but without using the previously set application context values.

Step 5: Remove the Components for This Example

1. Log on as SYS and connect using AS SYSDBA.

```
CONNECT SYS/AS SYSDBA
Enter password: password
```

2. Drop the global application context.

```
DROP CONTEXT global_cust_ctx;
```

Remember that even though sysadmin_ctx created the global application context, it is owned by the SYS schema.

3. Drop the two sample users.

```
DROP USER sysadmin_ctx CASCADE;
DROP USER apps_user;
```

Global Application Context Processes

This section provides two examples of how a global application context can be processed.

- Simple Global Application Context Process
- Global Application Context Process for Lightweight Users

Simple Global Application Context Process

Consider the application server, AppSvr, that has assigned the client identifier 12345 to client scott. The AppSvr application uses the scott user to create a session (that is, it is not a connection pool.) The value assigned to the context attribute can come from anywhere, for example, from running a SELECT statement on a table that holds the responsibility codes for users. When the application context is populated, it is stored in memory. As a result, any action that needs the responsibility code can access it quickly with SYS_CONTEXT call, without the overhead of accessing a table. The only advantage of a global context over a local context in this case is if scott were changing applications frequently and used the same context in each application.

The following steps show how the global application context process sets the client identifier for scott.

1. The administrator creates a global context namespace by using the following statement:

```
CREATE OR REPLACE CONTEXT hr USING hr.init ACCESSED GLOBALLY;
```

2. The administrator creates a PL/SQL package for the HR application context to indicate that, for this client identifier, there is an application context called responsibility with a value of 13 in the HR namespace.:

```
CREATE OR REPLACE PROCEDURE hr.init
 AS
 BEGIN
  DBMS_SESSION.SET_CONTEXT(
    namespace => 'HR',
```

```
    attribute => 'RESPONSIBILITY',
    value     => '13',
    username  => 'SCOTT',
    client_id => '12345' );
END;
```

This PL/SQL procedure is stored in the HR database schema, but typically it is stored in the schema of the security administrator.

3. The AppSvr application issues the following command to indicate the connecting client identity each time scott uses AppSvr to connect to the database:

```
DBMS_SESSION.SET_IDENTIFIER('12345');
```

4. When there is a SYS_CONTEXT('HR','RESPONSIBILITY') call within the database session, the database matches the client identifier, 12345, to the global context, and then returns the value 13.

5. When exiting this database session, AppSvr clears the client identifier by issuing the following procedure:

```
DBMS_SESSION.CLEAR_IDENTIFIER( );
```

6. To release the memory used by the application context, AppSvr issues the following procedure:

```
DBMS_SESSION.CLEAR_CONTEXT( );
```

CLEAR_CONTEXT is needed when the user session is no longer active, either on an explicit logout, timeout, or other conditions determined by the AppSvr application.

Note: After a client identifier in a session is cleared, it becomes 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.

Global Application Context Process for Lightweight Users

The following steps show the global application context process for a lightweight user application. The lightweight user, robert, is not known to the database through the application.

1. The administrator creates the global context namespace by using the following statement:

```
CREATE CONTEXT hr USING hr.init ACCESSED GLOBALLY;
```

2. The HR application server, AppSvr, starts and then establishes multiple connections to the HR database as the appsmgr user.

3. User SCOTT logs in to the HR application server.

4. AppSvr authenticates robert to the application.

5. AppSvr assigns a temporary session ID (or uses the application user ID), 12345, for this connection.

6. The session ID is returned to the browser used by robert as part of a cookie or is maintained by AppSvr.

7. AppSvr initializes the application context for this client by calling the hr.init package, which issues the following statements:

```
DBMS_SESSION.SET_CONTEXT( 'hr', 'id', 'robert', '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 the following statement:

```
DBMS_SESSION.SET_IDENTIFIER( 12345 );
```

9. All SYS_CONTEXT calls within this database session return application context values that belong only to the client session.

   For example, SYS_CONTEXT('hr','id') returns the value robert.

10. When finished with the session, AppSvr issues the following statement to clean up the client identity:

```
DBMS_SESSION.CLEAR_IDENTIFIER ( );
```

Even if another database user logged in to the database, this user cannot access the global context set by AppSvr, because AppSvr specified that only the application with user APPSMGR logged in can see it. If AppSvr used the following, then any user session with client ID set to 12345 can see the global context:

```
DBMS_SESSION.SET_CONTEXT( 'hr', 'id', 'robert', NULL , 12345 );
DBMS_SESSION.SET_CONTEXT( 'hr', 'dept', 'sales', NULL , 12345 );
```

Setting USERNAME to NULL enables different users to share the same context.

> **Note:** Be aware of the security implication of different settings of the global context. NULL in the user name means that any user can access the global context. A NULL client ID in the global context means that a session with an uninitialized client ID can access the global context. To ensure that only the user who has logged on can access the session, specify USER instead of NULL.

You can query the client identifier set in the session as follows:

```
SELECT SYS_CONTEXT('USERENV','CLIENT_IDENTIFIER') FROM dual;

SYS_CONTEXT('USERENV','CLIENT_IDENTIFIER')
------------------------------------------------
12345
```

A security administrator can see which sessions have the client identifier set by querying the V$SESSION view for the CLIENT_IDENTIFIER and USERNAME, for example:

```
COL client_identifier format a18
SELECT CLIENT_IDENTIFIER, USERNAME from V$SESSION;

CLIENT_IDENTIFIER   USERNAME
------------------  --------
12345               APPSMGR
```

To check the amount of global context area (in bytes) being used, you can run the following query:

```
SELECT SYS_CONTEXT('USERENV','GLOBAL_CONTEXT_MEMORY') FROM dual;

SYS_CONTEXT('USERENV','GLOBAL_CONTEXT_MEMORY')
----------------------------------------------
584
```

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 3-38
- *Oracle Database SQL Language Reference*
- *Oracle Call Interface Programmer's Guide*

Using Client Session-Based Application Contexts

This section contains the following topics:

- About Client Session-Based Application Contexts
- Setting a Value in the CLIENTCONTEXT Namespace
- Retrieving the Client Session ID
- Clearing a Setting in the CLIENTCONTEXT Namespace
- Clearing All Settings in the CLIENTCONTEXT Namespace

About Client Session-Based Application Contexts

In a client session-based application context, you use Oracle Call Interface (OCI) functions to set and clear user session information, which is then stored in the User Global Area (UGA).

The advantage of this type of application context is that an individual application can check for specific user session data, rather than having the database perform this task. Another advantage is that the calls to set the application context value are included in the next call to the server, which improves performance.

However, be aware that application context security is compromised with a client session-based application context: any application user can set the client application context, and no check is performed in the database.

You configure the client session-based application context for the client application only. You do not configure any settings on the database server to which the client connects. Any application context settings in the database server do not affect the client session-based application context.

To configure a client session-based application context, use the `OCIAppCtxSet` OCI function. A client session-based application context uses the `CLIENTCONTEXT` namespace, updatable by any OCI client or by the existing `DBMS_SESSION` package for application context. Oracle Database performs no privilege or package security checks for this type.

The `CLIENTCONTEXT` namespace enables a single application transaction to both change the user context information and use the same user session handle to service the new user request. You can set or clear individual values for attributes in the `CLIENTCONTEXT` namespace, or clear all their values.

- An OCI client uses the `OCIAppCtx` function to set variable length data for the namespace, called `OCISessionHandle`. The OCI network single, round-trip transport sends all the information to the server in one round-trip. On the server side, you can query the application context information by using the `SYS_CONTEXT` SQL function on the namespace. For example:
- A JDBC client uses the `oracle.jdbc.internal.OracleConnection` function to achieve the same purposes.

Any user can set, clear, or collect the information in the `CLIENTCONTEXT` namespace, because it is not protected by package-based security.

> **See Also:** *Oracle Call Interface Programmer's Guide* for more information about client application contexts

Setting a Value in the CLIENTCONTEXT Namespace

For Oracle Call Interface, to set a value in the `CLIENTCONTEXT` namespace, use a command in the following syntax:

```
err = OCIAppCtxSet((void *) session_handle,(dvoid *)"CLIENTCONTEXT",(ub4) 13,
                   (dvoid *)attribute_name, length_of_attribute_name
                   (dvoid *)attribute_value, length_of_attribute_value, errhp,
                   OCI_DEFAULT);
```

In this specification:

- *`session_handle`*: Represents the `OCISessionHandle` namespace.
- *`attribute_name`*: Name of attribute. For example, `responsibility`, with a length of `14`.
- *`attribute_value`*: Value of attribute. For example, `manager`, with a length of `7`.

For JDBC, use a command of the following form:

```
public void setApplicationContext(
  string CLIENTCONTEXT,
  string attribute,
  string value)
throws SQLException;
```

In this specification:

- *`attribute`*: Represents the attribute whose value needs to be set.
- *`value`*: Represents the value to be assigned to the attribute.

> **See Also:** "Managing Scalable Platforms" in *Oracle Call Interface Programmer's Guide* for details about the `OCIAppCtx` function

Retrieving the Client Session ID

To retrieve the client session ID, you can use the `OCIStmtExecute` call with either of the following statements:

```
SELECT SYS_CONTEXT('CLIENTCONTEXT', attribute) FROM dual;

SELECT value FROM session_context WHERE namespace='CLIENTCONTEXT' AND
'attribute='CLIENT_IDENTIFIER' ;
```

The *`attribute`* value can be any attribute value that has already been set in the `CLIENTCONTEXT` namespace. Oracle Database only retrieves the set attribute;

otherwise, it returns NULL. Typically, you set the attribute by using the OCIAppCtxSet call. In addition, you can embed a DBMS_SESSION.SET_CONTEXT call in the OCI code to set the attribute value.

Example 7–10 shows how to use the OCIStmtExecute call to retrieve a client session ID value.

Example 7–11 Retrieving a Client Session ID Value for Client Session-Based Contexts

```
 1 oratext    clientid[31];
 2 OCIDefine  *defnp1 = (OCIDefine *) 0;
 3 OCIStmt    *statementhndle;
 4 oratext    *selcid = (oratext *)"SELECT SYS_CONTEXT('CLIENTCONTEXT',
 5             attribute) FROM  DUAL";
 6
 7 OCIStmtPrepare(statementhndle, errhp, selcid, (ub4) strlen((char *) selcid),
 8   (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT);
 9
10 OCIDefineByPos(statementhndle, &defnp1, errhp, 1, (dvoid *)clientid, 31,
11   SQLT_STR, (dvoid *) 0, (ub2 *) 0, (ub2 *) 0, OCI_DEFAULT);
12
13 OCIStmtExecute(servhndle, statementhndle, errhp, (ub4) 1, (ub4) 0,
14   (CONST OCISnapshot *) NULL, (OCISnapshot *) NULL, OCI_DEFAULT);
15
16 printf("CLIENT_IDENTIFIER = %s \n", clientid);
```

In this example:

- **Lines 1–5**: Create variables to store the client session ID, reference call for OCIDefine, the statement handle, and the SELECT statement to use.
- **Lines 7–8**: Prepare the statement selcid for execution.
- **Lines 10–11**: Define the output variable clientid for client session ID.
- **Lines 1 3–14**: Execute the statement in the selcid variable.
- **Line 16**: Prints the formatted output for the retrieved client session ID.

Clearing a Setting in the CLIENTCONTEXT Namespace

For Oracle Call Interface, to clear a setting in CLIENTCONTEXT, set the value to NULL or to an empty string by using one of the following commands:

```
(void) OCIAppCtxSet((void *) session_handle, (dvoid *)"CLIENTCONTEXT", 13,
                    (dvoid *)attribute_name, length_of_attribute_name,
                    (dvoid *)0, 0,errhp
                    OCI_DEFAULT);
```

or

```
(void) OCIAppCtxSet((void *) session_handle, (dvoid *)"CLIENTCONTEXT", 13
                    (dvoid *)attribute_name, length_of_attribute_name,
                    (dvoid *)"", 0,errhp,
                    OCI_DEFAULT);
```

For JDBC, use the following command:

```
public void setApplicationContext(String CLIENTCONTEXT,
                    string attribute,
                    string value)
                    throws SQLException;
```

In this specification:

- *attribute*: Represents the attribute whose value needs to be cleared.
- *value*: Either 0 or the null string ("").

Clearing All Settings in the CLIENTCONTEXT Namespace

For Oracle Call Interface (OCI), use a command of the following form:

```
err = OCIAppCtxClearAll((void *) session_handle,
                        (dvoid *)"CLIENTCONTEXT", 13,
                         errhp,
                         OCI_DEFAULT);
```

For JDBC, use a command of the following form:

```
public void clearAllApplicationContext(
                         String CLIENTCONTEXT)
                         throws SQLException;
```

Finding Information About Application Contexts

Table 7–3 lists data dictionary views that you can query to find information about application contexts. For detailed information about these views, see *Oracle Database Reference.*

Table 7–3 Application Context Views

View	Description
ALL_CONTEXT	Describes all context namespaces in the current session for which attributes and values were specified using the DBMS_SESSION.SET_CONTEXT procedure. It lists the namespace and its associated schema and PL/SQL package.
ALL_POLICY_CONTEXTS	Describes the driving contexts defined for the synonyms, tables, and views accessible to the current user. (A driving context is a context used in a Virtual Private Database policy.)
DBA_CONTEXT	Provides all context namespace information in the database. Its columns are the same as those in the ALL_CONTEXT view, except that it includes the TYPE column. The TYPE column describes how the application context is accessed or initialized.
DBA_POLICY_CONTEXTS	Describes all driving contexts in the database. Its columns are the same as those in ALL_POLICY_CONTEXTS.
SESSION_CONTEXT	Describes the context attributes and their values set for the current session.
USER_POLICY_CONTEXTS	Describes the driving contexts defined for the synonyms, tables, and views owned by the current user. Its columns (except for OBJECT_OWNER) are the same as those in ALL_POLICY_CONTEXTS.
V$CONTEXT	Lists set attributes in the current session. Users do not have access to this view unless you grant the user EXECUTE privileges on it.
V$SESSION	Lists detailed information about each current session. Users do not have access to this view unless you grant the user EXECUTE privileges on it.

Tip: In addition to these views, check the database trace file if you find errors when running applications that use application contexts. See *Oracle Database Performance Tuning Guide* for more information about trace files. The `USER_DUMP_DEST` initialization parameter specifies the current location of the trace files. You can find the value of this parameter by issuing `SHOW PARAMETER USER_DUMP_DEST` in SQL*Plus.

8

Using Oracle Virtual Private Database to Control Data Access

Oracle Virtual Private Database (VPD) enables you to create security policies to control database access at the row and column level. Essentially, Oracle Virtual Private Database adds a dynamic `WHERE` clause to a SQL statement that is issued against the table, view, or synonym to which an Oracle Virtual Private Database security policy was applied.

This chapter discusses the following topics:

- About Oracle Virtual Private Database
- Components of an Oracle Virtual Private Database Policy
- Configuring an Oracle Virtual Private Database Policy
- Examples: Creating Oracle Virtual Private Database Policies
- How Oracle Virtual Private Database Works with Other Oracle Features
- Finding Information About Oracle Virtual Private Database Policies

About Oracle Virtual Private Database

This section introduces Oracle Virtual Private Database.

- What Is Oracle Virtual Private Database?
- Benefits of Using Oracle Virtual Private Database Policies
- Components of an Oracle Virtual Private Database Policy
- Using Oracle Virtual Private Database with an Application Context

What Is Oracle Virtual Private Database?

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

When a user directly or indirectly accesses a table, view, or synonym that is protected with an Oracle Virtual Private Database policy, Oracle Database dynamically modifies the SQL statement of the user. This modification creates a `WHERE` condition (called a predicate) returned by a function implementing the security policy. Oracle Database modifies the statement dynamically, transparently to the user, using any condition that

can be expressed in or returned by a function. You can apply Oracle Virtual Private Database policies to SELECT, INSERT, UPDATE, INDEX, and DELETE statements.

For example, suppose a user performs the following query:

```
SELECT * FROM oe.orders;
```

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

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

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

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

```
SELECT * FROM oe.orders
 WHERE sales_rep_id = SYS_CONTEXT('userenv','session_user');
```

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

Benefits of Using Oracle Virtual Private Database Policies

Oracle Virtual Private Database policies provide the following benefits:

- Basing Security Policies on Database Objects Rather Than Applications
- Controlling How Oracle Database Evaluates Policy Functions

Basing Security Policies on Database Objects Rather Than Applications

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

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

Controlling How Oracle Database Evaluates Policy Functions

Running policy functions multiple times can affect performance. You can control the performance of policy functions by configuring how Oracle Database caches the Oracle Virtual Private Database predicates. The following options are available:

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

See "Optimizing Performance by Using Oracle Virtual Private Database Policy Types" on page 8-16 for information configuring these policy types.

Using Oracle Virtual Private Database with an Application Context

You can use application contexts with Oracle Virtual Private Database policies. When you create an application context, it securely caches user information. Only the designated application package can set the cached environment. It cannot be changed by the user or outside the package. In addition, because the data is cached, performance is increased. Chapter 7, "Using Application Contexts to Retrieve User Information" describes application contexts in detail.

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

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

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

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

```
SELECT * FROM orders_tab
```

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

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

Continuing with the preceding example, suppose you have 50,000 customers, and you do not want to have a different predicate returned for each customer. Customers all

share the same WHERE predicate, which prescribes that they can only see their own orders. It is merely their customer numbers that are different.

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

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

Components of an Oracle Virtual Private Database Policy

To implement Oracle Virtual Private Database, you need to create a function to generate the dynamic WHERE clause, and a policy to attach this function to the objects that you want to protect.

- Creating a Function to Generate the Dynamic WHERE Clause
- Creating a Policy to Attach the Function to the Objects You Want to Protect

> **See Also:** "Examples: Creating Oracle Virtual Private Database Policies" on page 8-21

Creating a Function to Generate the Dynamic WHERE Clause

To generate the dynamic WHERE clause (predicate), you need to create a function (not a procedure) that defines the restrictions that you want to enforce. Usually, the security administrator creates this function in his or her own schema. For more complex behavior, such as including calls to other functions or adding checks to track failed logon attempts, create these functions within a package.

The function must have the following components:

- **It must take as arguments a schema name and an object (table, view, or synonym) name as inputs.** Define input parameters to hold this information, but do not specify the schema and object name themselves within the function. The policy that you create with the DBMS_RLS package (described in "Creating a Policy to Attach the Function to the Objects You Want to Protect" on page 8-5) provides the names of the schema, and object to which the policy will apply. You must create the parameter for the schema first, followed by the parameter for the object.
- **It must provide a return value for the WHERE clause predicate that will be generated.** The return value for the WHERE clause is always a VARCHAR2 data type.
- **It must generate a valid WHERE clause.** This code can be as basic as the example in "Simple Example of Creating an Oracle Virtual Private Database Policy" on page 8-21, in that its WHERE clause is the same for all users who log on.

  But in most cases, you may want to design the WHERE clause to be different for each user, each group of users, or each application that accesses the objects you want to protect. For example, if a manager logs in, the WHERE clause can be specific to the rights of that particular manager. You can do this by incorporating an application context, which accesses user session information, into the WHERE clause generation code. "Example of Implementing a Policy with a Database Session-Based Application Context" on page 8-23 demonstrates how to create an Oracle Virtual Private Database policy that uses an application context.

You can create Oracle Virtual Private Database functions that do not use an application context, but an application context creates a much stronger Oracle Virtual Private Database policy, by securely basing user access on the session attributes of that user, such as the user ID. Chapter 7, "Using Application Contexts to Retrieve User Information" discusses different types of application contexts in detail.

In addition, you can embed C or Java calls to access operating system information or to return `WHERE` clauses from an operating system file or other source.

Creating a Policy to Attach the Function to the Objects You Want to Protect

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

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

"Examples: Creating Oracle Virtual Private Database Policies" on page 8-21 provides examples of how to create Virtual Private Database policies. See "Configuring an Oracle Virtual Private Database Policy" on page 8-5 for detailed information.

Configuring an Oracle Virtual Private Database Policy

This section describes how to configure Oracle Virtual Private Database policies.

- About Oracle Virtual Private Database Policies
- Attaching a Policy a Database Table, View, or Synonym
- Enforcing Policies on Specific SQL Statement Types
- Controlling the Display of Column Data with Policies
- Working with Policy Groups
- Optimizing Performance by Using Oracle Virtual Private Database Policy Types

About Oracle Virtual Private Database Policies

After you create a function that defines the actions of the Oracle Virtual Private Database `WHERE` clause, you need to associate this function with the database table to which the VPD action applies. You can do this by configuring an Oracle Virtual Private Database policy. The policy itself is a mechanism for managing the Virtual Private Database function. The policy also enables you to add fine-grained access control, such as specifying the types of SQL statements or particular table columns the policy affects. When a user tries to access the data in this database object, the policy goes into effect automatically.

This section describes commonly used ways of attaching policies to tables, views, and synonyms. To manage an Oracle Virtual Private Database policy, you use the `DBMS_RLS` package, which is described in detail in *Oracle Database PL/SQL Packages and Types Reference*.

Table 8–1 lists the procedures in the DBMS_RLS package.

Table 8–1 DBMS_RLS Procedures

Procedure	Description
For Handling Individual Policies	
DBMS_RLS.ADD_POLICY	Adds a policy to a table, view, or synonym
DBMS_RLS.ENABLE_POLICY	Enables (or disables) a policy you previously added to a table, view, or synonym
DBMS_RLS.REFRESH_POLICY	Invalidates cursors associated with nonstatic policies
DBMS_RLS.DROP_POLICY	To drop a policy from a table, view, or synonym
For Handling Grouped Policies	
DBMS_RLS.CREATE_POLICY_GROUP	Creates a policy group
DBMS_RLS.DELETE_POLICY_GROUP	Drops a policy group
DBMS_RLS.ADD_GROUPED_POLICY	Adds a policy to the specified policy group
DBMS_RLS.ENABLE_GROUPED_POLICY	Enables a policy within a group
DBMS_RLS.REFRESH_GROUPED_POLICY	Parses again the SQL statements associated with a refreshed policy
DBMS_RLS.DISABLE_GROUPED_POLICY	Disables a policy within a group
DBMS_RLS.DROP_GROUPED_POLICY	Drops a policy that is a member of the specified group
For Handling Application Contexts	
DBMS_RLS.ADD_POLICY_CONTEXT	Adds the context for the active application
DBMS_RLS.DROP_POLICY_CONTEXT	Drops the context for the application

See Also:

- "Components of an Oracle Virtual Private Database Policy" on page 8-4 for a description of the type of function that you need to create to control user access to a database table, view, or synonym
- Chapter 7, "Using Application Contexts to Retrieve User Information" if you plan to use application contexts in the Oracle Virtual Private Database policy (which in most cases, you would)
- "Examples: Creating Oracle Virtual Private Database Policies" on page 8-21 for examples of using application contexts in sample Oracle Virtual Private Database functions

Attaching a Policy a Database Table, View, or Synonym

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

Example 8–1 shows how to use DBMS_RLS.ADD_POLICY to attach an Oracle Virtual Private Database policy called secure_update to the HR.EMPLOYEES table. The function attached to the policy is check_updates.

Example 8–1 Attaching a Simple Oracle Virtual Private Database Policy to a Table

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

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

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

> **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.

Enforcing Policies on Specific SQL Statement Types

You can enforce Oracle Virtual Private Database policies for SELECT, INSERT, UPDATE, INDEX, and DELETE statements. If you do not specify a statement type, by default, Oracle Database specifies SELECT, INSERT, UPDATE, and DELETE, but not INDEX. Enter any combination of these statement types by using the statement_types parameter in the DBMS_RLS.ADD_POLICY procedure. Enclose the list in a pair of single quotation marks.

Example 8–2 shows an how to specify the SELECT and INDEX statements for a policy.

Example 8–2 Specifying SQL Statement Types with DBMS_RLS.ADD_POLICY

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

> **Important:** Be aware that a user who has privileges to maintain an index can see all the row data, even if the user does not have full table access under a regular query such as SELECT. For example, a user can create a function-based index that contains a user-defined function with column values as its arguments. During index creation, Oracle Database passes column values of every row into the user function, making the row data available to the user who creates the index. You can enforce Oracle Virtual Private Database policies on index maintenance operations by specifying INDEX with the statement_types parameter.

Controlling the Display of Column Data with Policies

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

- Adding Policies for Column-Level Oracle Virtual Private Database
- Displaying Only the Column Rows Relevant to the Query
- Using Column Masking to Display Sensitive Columns as NULL Values

Adding Policies for Column-Level Oracle Virtual Private Database

Column-level policies enforce row-level security when a query references a security-relevant column. You can apply a column-level Oracle Virtual Private Database policy to tables and views, but not to synonyms.

To apply the policy to a column, specify the security-relevant column by using the `sec_relevant_columns` parameter of the `DBMS_RLS.ADD_POLICY` procedure. This parameter applies the security policy whenever the column is referenced, explicitly or implicitly, in a query.

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

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

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

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

Example 8–3 shows a Oracle Virtual Private Database policy in which sales department users cannot see the salaries of people outside the department (department number 30) of the sales department users. The relevant columns for this policy are `sal` and `comm`. First, the Oracle Virtual Private Database policy function is created, and then it is added by using the `DBMS_RLS` PL/SQL package.

Example 8–3 Creating a Column-Level Oracle Virtual Private Database Policy

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

RETURN VARCHAR2 AS
con VARCHAR2 (200);

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

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

```
BEGIN
 DBMS_RLS.ADD_POLICY (
  object_schema     => 'scott',
  object_name       => 'emp',
  policy_name       => 'hide_sal_policy',
```

```
  policy_function   => 'hide_sal_comm',
  sec_relevant_cols => 'sal,comm');
END;
```

Displaying Only the Column Rows Relevant to the Query

The default behavior for column-level Oracle Virtual Private Database is to restrict the number of rows returned for a query that references columns containing sensitive information. You specify these security-relevant columns by using the `sec_relevant_columns` parameter of the `DBMS_RLS.ADD_POLICY` procedure, as shown in Example 8–3 on page 8-8.

For example, consider sales department users with the `SELECT` privilege on the `emp` table, which is protected with the column-level Oracle Virtual Private Database policy created in Example 8–3. The user (for example, user `SCOTT`) runs the following query:

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

The database returns the following rows:

```
ENAME      DNAME          JOB              SAL       COMM
---------- -------------- --------- ---------- ----------
ALLEN      SALES          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

6 rows selected.
```

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

Using Column Masking to Display Sensitive Columns as NULL Values

If a query references a sensitive column, then the default action of column-level Oracle Virtual Private Database restricts the number of rows returned. With column-masking behavior, all rows display, even those that reference sensitive columns. However, the sensitive columns display as `NULL` values. To enable column-masking, set the `sec_relevant_cols_opt` parameter of the `DBMS_RLS.ADD_POLICY` procedure.

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

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

Example 8–4 shows column-level Oracle Virtual Private Database column-masking. It uses the same VPD policy as Example 8–3 on page 8-8, but with `sec_relevant_cols_opt` specified as `dbms_rls.ALL_ROWS`.

Example 8–4 Adding a Column Masking to an Oracle Virtual Private Database Policy

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

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

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

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

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

14 rows selected.
```

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

The following considerations apply to column-masking:

- Column-masking applies only to `SELECT` statements.
- Column-masking conditions generated by the policy function must be simple Boolean expressions, unlike regular Oracle Virtual Private Database predicates.
- For applications that perform calculations, or do not expect `NULL` values, use standard column-level Oracle Virtual Private Database, specifying `sec_relevant_cols` rather than the `sec_relevant_cols_opt` column-masking option.
- Column-masking used with `UPDATE AS SELECT` updates only the columns that users are allowed to see.
- For some queries, column-masking may prevent some rows from displaying. For example:

  ```
  SELECT * FROM emp
  ```

```
WHERE sal = 10;
```

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

Working with Policy Groups

You can group multiple security policies together, and apply them to an application. This section describes the following topics:

- About Policy Groups
- Creating a New Policy Group
- Example of Implementing a Policy Group
- Designating a Default Policy Group with the SYS_DEFAULT Policy Group
- Establishing Multiple Policies for Each Table, View, or Synonym
- Validating the Application Used to Connect to the Database

About Policy Groups

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

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

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

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

Creating a New Policy Group

To add a policy to a table, view, or synonym, use the `DBMS_RLS.ADD_GROUPED_POLICY` procedure to specify the group to which the policy belongs. To specify which policies will be effective, you can add a driving context using the `DBMS_RLS.ADD_POLICY_CONTEXT` procedure. If the driving context returns an unknown policy group, then an error is returned.

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

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

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

Example of Implementing a Policy Group

To create a policy group, you must first create a driving context to identify the effective policy group. Then, you can add policies to the policy groups as required.

The following steps show how to implement a policy group:

- Step 1: Create the Components for This Example
- Step 2: Create the Driving Application Context
- Step 3: Add a Policy to the Default Policy Group
- Step 4: Add a Policy to the HR Policy Group
- Step 5: Add a Policy to the FINANCE Policy Group

The following example shows how to perform these tasks.

Step 1: Create the Components for This Example

In SQL*Plus, run the following statements:

```
DROP USER finance CASCADE;
CREATE USER finance IDENTIFIED BY beancounter4u;
GRANT RESOURCE TO apps;
DROP TABLE apps.benefit;
CREATE TABLE apps.benefit (c NUMBER);
```

Step 2: Create the Driving Application Context

To create the driving application context, you create a namespace by using the CREATE CONTEXT SQL statement. Remember that you need CREATE ANY CONTEXT privileges to use this statement.

1. Create the namespace for the driving application context:

```
CREATE OR REPLACE CONTEXT appsctx USING apps.apps_security_init;
```

2. Create the package that administers the driving context:

```
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;
/
```

3. Define the driving context for the table `APPS.BENEFIT`:

```
BEGIN
DBMS_RLS.ADD_POLICY_CONTEXT('apps','benefit','APPSCTX','ACTIVE_APPS');
END;
/
```

Step 3: Add a Policy to the Default Policy Group

1. 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;
/
```

   Because policies in the `SYS_DEFAULT` policy group are executed (except for `SYS`, or users with the `EXEMPT ACCESS POLICY` system privilege), the `SECURITY_BY_COMPANY` security policy is enforced, regardless of the application running. This achieves the universal security requirement on the table: that each company should see its own data regardless of the application that is running. The function `APPS.APPS_SECURITY_INIT.BY_COMPANY` returns the predicate to ensure that users can only see data related to their own company.

2. Run the `DBMS_RLS.ADD_GROUPED_POLICY` procedure:

```
BEGIN
DBMS_RLS.ADD_GROUPED_POLICY('apps','benefit','SYS_DEFAULT',
'security_by_company',
'apps','by_company');
END;
/
```

Step 4: Add a Policy to the HR Policy Group

1. Create a function for the `HR` policy group:

```
CREATE OR REPLACE FUNCTION hr.security_policy
RETURN VARCHAR2
AS
BEGIN
 RETURN 'SYS_CONTEXT(''ID'',''TITLE'') = ''MANAGER'' ';
END;
/
```

2. Create the `HR` policy group and then add the `HR_SECURITY` policy to the `HR` policy group:

```
BEGIN
 DBMS_RLS.CREATE_POLICY_GROUP('apps','benefit','HR');
```

```
 DBMS_RLS.ADD_GROUPED_POLICY('apps','benefit','HR',
 'hr_security','hr','security_policy');
END;
/
```

The function `HR.SECURITY_POLICY` returns the predicate to enforce security on the `APPS.BENEFIT` table.

Step 5: Add a Policy to the FINANCE Policy Group

1. Create a function for the `FINANCE` policy group:

```
CREATE OR REPLACE FUNCTION finance.security_policy
RETURN VARCHAR2
AS
BEGIN
 RETURN ('SYS_CONTEXT(''ID'',''DEPT'') = ''FINANCE'' ');
END;
```

2. 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;
```

When a user accesses the database, the application initializes the driving context after authentication. For example, `HR` application initializes it as follows:

```
EXECUTE apps.security_init.setctx('HR');
```

Designating a Default Policy Group with the SYS_DEFAULT Policy Group

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

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

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

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

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

Establishing Multiple Policies for Each Table, View, or Synonym

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

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

Validating the Application Used to Connect to the Database

The package implementing the driving context must correctly validate the application that is being used to connect to the database. Although Oracle Database checks the call stack to ensure that the package implementing the driving context sets context attributes, inadequate validation can still occur within the package.

For example, in applications where database users or enterprise users are known to the database, the user needs the `EXECUTE` privilege on the package that sets the driving context. Consider a user who knows that:

- The `BENEFITS` application enables more liberal access than the `HR` application.
- The `setctx` procedure (which sets the correct policy group within the driving context) does not perform any validation to determine which application is actually connecting. That is, the procedure does not check either the IP address of the incoming connection (for a three-tier system) or the `proxy_user` attribute of the user session.

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

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

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

In this case, the following query is executed:

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

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

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

Optimizing Performance by Using Oracle Virtual Private Database Policy Types

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

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

This section describes the following topics:

- Using the Dynamic Policy Type to Automatically Rerun Policy Functions
- Using a Static Policy to Prevent Policy Functions from Rerunning for Each Query
- Using a Shared Static Policy to Share a Policy with Multiple Objects
- When to Use Static and Shared Static Policies
- Using a Context-Sensitive Policy for Predicates That Do Not Change After Parsing
- Using a Shared Context Sensitive Policy to Share a Policy with Multiple Objects
- When to Use Context-Sensitive and Shared Context-Sensitive Policies
- Summary of the Five Oracle Virtual Private Database Policy Types

Using the Dynamic Policy Type to Automatically Rerun Policy Functions

The DYNAMIC policy type runs the policy function each time a user accesses the Virtual Private Database-protected database objects. If you do not specify a policy type in the DBMS_RLS.ADD_POLICY procedure, then, by default, your policy will be dynamic. You can specifically configure a policy to be dynamic by setting the policy_type parameter of the DBMS_RLS.ADD_POLICY procedure to DYNAMIC.

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

You can use the DBMS_UTILITY.GET_TIME procedure to measure the start and end times for a statement to execute. For example:

```
SQL> SELECT DBMS_UTILITY.GET_TIME FROM dual;

  GET_TIME
----------
   2312721
```

```
SQL> SELECT COUNT(*) FROM hr.employees;

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

SQL> SELECT DBMS_UTILITY.GET_TIME FROM dual;

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

Example 8–5 shows how to create the DYNAMIC policy type.

Example 8–5 Creating a DYNAMIC Policy with DBMS_RLS.ADD_POLICY

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

Using a Static Policy to Prevent Policy Functions from Rerunning for Each Query

The static policy type enforces the same predicate for all users in the instance. Oracle Database stores static policy predicates in SGA, so policy functions do not rerun for each query. This results in faster performance.

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

Example 8–6 shows how to create the STATIC policy type.

Example 8–6 Creating a STATIC Policy with DBMS_RLS.ADD_POLICY

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

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

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

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

Although the predicate does not change for each query, it applies to the query based on session attributes of the SYS_CONTEXT. In the case of the preceding example, the

predicate returns only those rows where the department number matches the deptno attribute of the SYS_CONTEXT, which is the department number of the user who is querying the policy-protected database object.

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

Using a Shared Static Policy to Share a Policy with Multiple Objects

If, for example, you wanted to apply the policy in Example 8–6 to a second table in the HR schema that may contain financial data that you want to side, you would use the SHARED_STATIC setting for both tables.

Example 8–7 shows how to set the SHARED_STATIC policy type for two tables that share the same policy.

Example 8–7 Creating a SHARED_STATIC Policy with DBMS_RLS.ADD_POLICY

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

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

When to Use Static and Shared 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, rerunning the policy function each time adds unnecessary overhead to the system. For example, consider a data warehouse that contains market research data for customer organizations that are competitors. The warehouse must enforce the policy that each organization can see only their own market research, which is expressed by the following predicate:

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

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

Using a Context-Sensitive Policy for Predicates That Do Not Change After Parsing

In contrast to static policies, context-sensitive policies do not always cache the predicate. With context-sensitive policies, the database assumes that the predicate will change after statement parse time. But if there is no change in local application context, Oracle Database does not rerun the policy function within the user session. If there was a change in context, then the database reruns the policy function to ensure that it captures any changes to the predicate since the initial parsing.

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`.

Example 8–8 shows how to create the `CONTEXT_SENSITIVE` policy type.

Example 8–8 Creating a CONTEXT_SENSITIVE Policy with DBMS_RLS.ADD_POLICY

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

Context-sensitive policies are useful when different predicates should apply depending on which user is executing the query. For example, consider the case where managers should have the predicate `WHERE group` set to `managers`, and employees should have the predicate `WHERE empno` set to `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.

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

Using a Shared Context Sensitive Policy to Share a Policy with Multiple Objects

Example 8–9 Ishows how to create two shared context sensitive policies that share a policy with multiple tables.

Example 8–9 Creating a SHARED_CONTEXT_SENSITIVE Policy with DBMS_RLS.ADD_POLICY

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

-- Now create a policy for the second table, fin_data:
BEGIN
```

```
 DBMS_RLS.ADD_POLICY(
  object_schema   => 'hr',
  object_name     => 'fin_data',
  policy_name     => 'secure_update',
  policy_function => 'hide_fin',
  policy_type     => dbms_rls.SHARED_CONTEXT_SENSITIVE);
END;
```

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

Context-sensitive policies are useful when a predicate does not need to change for a user 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. This policy states that analysts can see only their own products and regional employees can see only their own region. In this case, the database must rerun the policy function each time the type of user changes. The performance gain is realized when a user can log in and issue several DML statements against the protected object without causing the server to rerun the policy function.

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

Summary of the Five Oracle Virtual Private Database Policy Types

Table 8–2 summarizes the types of policy types available.

Table 8–2 DBMS_RLS.ADD_POLICY Policy Types

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

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

Examples: Creating Oracle Virtual Private Database Policies

This section provides the following examples of creating Oracle Virtual Private Database policies.

- Simple Example of Creating an Oracle Virtual Private Database Policy
- Example of Implementing a Policy with a Database Session-Based Application Context

Simple Example of Creating an Oracle Virtual Private Database Policy

This example shows how to create a simple Oracle Virtual Private Database policy that limits access to all orders in the `OE.ORDERS` table that were created by Sales Representative 159. In essence, the policy translates the following statement:

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

To the following:

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

Follow these steps:

- Step 1: Ensure That the OE User Account Is Active
- Step 2: Create a Policy Function
- Step 3: Create the Oracle Virtual Private Database Policy
- Step 4: Test the Policy
- Step 5: Remove the Components for This Example

Step 1: Ensure That the OE User Account Is Active

1. Log on to SQL*Plus as `SYS` and connect using the `AS SYSDBA` privilege.

   ```
   sqlplus "SYS/AS SYSDBA"
   Enter password: password
   ```

2. Run the following `SELECT` statement on the `DBA_USERS` data dictionary view:

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

   If the `DBA_USERS` view lists user `OE` as locked and expired, then enter the following statement to unlock the `OE` account and create a new password for him:

   ```
   ALTER USER OE ACCOUNT UNLOCK IDENTIFIED BY ready2go;
   ```

   The password `ready2go` is offered as an example of a valid password, but you can create any password that is secure, according to the requirements described in "How Oracle Database Checks the Complexity of Passwords" on page 3-7.

Step 2: Create a Policy Function

Create the following function, which will append the `WHERE SALES_REP_ID = 159` clause to any `SELECT` statement on the `OE.ORDERS` table.

```
SQL> CREATE OR REPLACE FUNCTION auth_orders(
  2  schema_var IN VARCHAR2,
  3  table_var  IN VARCHAR2
  4 )
```

```
 5 RETURN VARCHAR2
 6 IS
 7  return_val VARCHAR2 (400);
 8 BEGIN
 9  return_val := 'SALES_REP_ID = 159';
10  RETURN return_val;
11 END auth_orders;
```

In this example:

- **Lines 2–3**: Create input parameters to specify to store the schema name, `OE`, and table name, `ORDERS`. First, define the parameter for the schema, and then define the parameter for the object, in this case, a table. Always create them in this order. The Virtual Private Database policy you create will need these parameters to specify the `OE.ORDERS` table.
- **Line 5**: Returns the string that will be used for the `WHERE` predicate clause. Remember that return value is always a `VARCHAR2` data type.
- **Lines 6–10**: Encompass the creation of the `WHERE SALES_REP_ID = 159` predicate.

Step 3: Create the Oracle Virtual Private Database Policy

Next, create the following policy by using the `ADD_POLICY` procedure in the `DBMS_RLS` package.

```
SQL> BEGIN
  2   DBMS_RLS.ADD_POLICY (
  3     object_schema    => 'oe',
  4     object_name      => 'orders',
  5     policy_name      => 'orders_policy',
  6     function_schema  => 'sys',
  7     policy_function  => 'auth_orders',
  8     statement_types  => 'select, insert, update, delete'
  9   );
 10  END;
```

In this example:

- **Line 3**: Specifies the schema that you want to protect, that is, `OE`.
- **Line 4**: Specifies the object within the schema to protect, that is, the `ORDERS` table.
- **Line 5**: Names this policy `orders_policy`.
- **Line 6**: Specifies the schema in which the `auth_orders` function was created. In this example, `auth_orders` was created in the `SYS` schema. But typically, it should be created in the schema of a security administrator.
- **Line 7**: Specifies a function to enforce the policy. Here, you specify the `auth_orders` function that you created in Step 2: Create a Policy Function.
- **Line 8**: Specifies the operations to which the policy applies. In this example, the policy applies to all `SELECT`, `INSERT`, `UPDATE`, and `DELETE` statements the user may perform.

Step 4: Test the Policy

After you create the Oracle Virtual Private Database policy, it goes into effect immediately. The next time a user, including the owner of the schema, performs a `SELECT` on `OE.ORDERS`, only the orders by Sales Representative 159 will be accessed.

1. Log on as user OE.

```
CONNECT oe
Enter password: password
Connected.
```

2. Enter the following SELECT statement:

```
SELECT COUNT(*) FROM ORDERS;

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

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

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

3. Log back on as user SYS.

```
CONNECT SYS/AS SYSDBA
Enter password: password
Connected.
```

4. Enter the follow SELECT statement:

```
SELECT COUNT(*) FROM ORDERS;

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

Step 5: Remove the Components for This Example

1. As user SYS, remove the function and policy as follows:

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

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

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

Example of Implementing a Policy with a Database Session-Based Application Context

This example uses a database session-based application context to implement a policy in which customers can see only their own orders. This example creates the following layers of security:

1. When a user logs on, database session-based application context permits only users who are customers to log on.
2. After a customer has logged on, an Oracle Virtual Private Database policy restricts this user to see only his orders.
3. As a further restriction, Oracle Virtual Private Database policy permits the user to only view his orders. He cannot add, modify, or remove orders.

The steps to create this example are as follows:

- Step 1: Create User Accounts and Sample Tables
- Step 2: Create a Database Session-Based Application Context

- Step 3: Create a PL/SQL Package to Set the Application Context
- Step 4: Create a Logon Trigger for the Application Context PL/SQL Package
- Step 5: Create a PL/SQL Policy Function to Limit User Access to Their Orders
- Step 6: Create the New Security Policy
- Step 7: Test the New Policy
- Step 8: Remove the Components for This Example

Step 1: Create User Accounts and Sample Tables

1. Start SQL*Plus and log on as a user who has administrative privileges.

```
sqlplus "SYS/AS SYSDBA"
Enter password: password
Connected.
```

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

```
GRANT CREATE SESSION, CREATE ANY CONTEXT, CREATE PROCEDURE, CREATE TRIGGER,
ADMINISTER DATABASE TRIGGER TO sysadmin_vpd IDENTIFIED BY omni2all;
GRANT EXECUTE ON DBMS_SESSION TO sysadmin_vpd;
GRANT EXECUTE ON DBMS_RLS TO sysadmin_vpd;
```

3. Create the following user accounts:

```
GRANT CREATE SESSION TO tbrooke IDENTIFIED BY shop2drop;
GRANT CREATE SESSION TO owoods IDENTIFIED BY loads4me;
```

4. Check the status of the sample user `scott`, who will be used for this example:

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

   If the `DBA_USERS` view lists user `scott` as locked and expired, then enter the following statement to unlock the `scott` account and create a new password for him:

```
ALTER USER SCOTT ACCOUNT UNLOCK IDENTIFIED BY tgris86d;
```

   The password `tgris86d` is offered as an example of a valid password, but you can create any password that is secure, according to the requirements described in "How Oracle Database Checks the Complexity of Passwords" on page 3-7.

5. Connect as user `SCOTT`, and then create and populate the `customers` table.

```
CONNECT scott
Enter password: tgris86d
Connected.

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

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

6. User `sysadmin_vpd` will need select privileges for the customers table, so as user `scott`, grant him this privilege.

```
GRANT SELECT ON customers TO sysadmin_vpd;
```

7. Create and populate the orders_tab table.

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

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

8. Users tbrooke and owoods need to query the orders_tab table, so grant them the SELECT privilege.

```
GRANT SELECT ON orders_tab TO tbrooke;
GRANT SELECT ON orders_tab TO owoods;
```

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

Step 2: Create a Database Session-Based Application Context

1. Connect as user sysadmin_vpd.

```
CONNECT sysadmin_vpd
Enter password: omni2all
Connected.
```

2. Enter the following statement:

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

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

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

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

```
SQL> CREATE OR REPLACE PACKAGE orders_ctx_pkg IS
  2   PROCEDURE set_custnum;
  3  END;
  4/
  5 CREATE OR REPLACE PACKAGE BODY orders_ctx_pkg IS
  6   PROCEDURE set_custnum
  7   AS
  8     custnum NUMBER;
  9   BEGIN
 10      SELECT cust_no INTO custnum FROM scott.customers
 11         WHERE cust_email = SYS_CONTEXT('USERENV', 'SESSION_USER');
 12      DBMS_SESSION.SET_CONTEXT('orders_ctx', 'cust_no', custnum);
 13   END set_custnum;
 14 END;
 17 /
```

In this example:

- **Line 8**: Creates the custnum variable, which will hold the customer ID.

- **Line 10**: Performs a `SELECT` statement to copy the customer ID that is stored in the `cust_no` column data from the `scott.customers` table into the `custnum` variable.
- **Line 11**: Uses a `WHERE` clause to find all the customer IDs that match the user name of the user who is logging on.
- **Line 12**: Sets the `order_entry` application context values by creating the `cust_no` attribute and then setting it to the value stored in the `custnum` variable.

To summarize, the `sysadmin_vpd.set_cust_num` procedure says, "Get the session user ID of the user, and then find the customer user name matches this session user ID. If they are the same, then let the user log on. If not, then deny the user access."

Step 4: Create a Logon Trigger for the Application Context PL/SQL Package

The logon trigger runs the procedure in the PL/SQL package that you created in Step 3: Create a PL/SQL Package to Set the Application Context the next time a user logs on.

As user `sysadmin_vpd`, create the following trigger:

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

Step 5: Create a PL/SQL Policy Function to Limit User Access to Their Orders

At this stage, only customers who are listed in the `sysadmin_vpd.customers` table can log in to the database. The next step is to create a PL/SQL function that, when the user who has logged in performs a `SELECT * FROM scott.orders_tab` query, displays only the orders of that user.

As user `sysadmin_vpd`, create the following function:

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

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

Step 6: Create the New Security Policy

As user `sysadmin_vpd`, create the policy as follows:

```
BEGIN
 DBMS_RLS.ADD_POLICY (
  object_schema    => 'scott',
```

```
  object_name      => 'orders_tab',
  policy_name      => 'orders_policy',
  function_schema  => 'sysadmin_vpd',
  policy_function  => 'get_user_orders',
  statement_types  => 'select');
END;
/
```

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

Step 7: Test the New Policy

1. Log on as user `tbrooke`.

   ```
   CONNECT tbrooke
   Enter password: shop2drop
   Connected.
   ```

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

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

   ```
   SELECT * FROM scott.orders_tab;

   VALUES
   ------
   9876
   ```

   User `tbrooke` has passed the second test. He can access his own orders in the `scott.orders_tab` table.

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

   ```
   CONNECT owoods
   Enter password: loads4me

   SELECT * FROM scott.orders_tab

   VALUES
   ------
   5432
   4592
   ```

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

Note the following about this example:

- You can create several predicates based on the position of a user. For example, a sales representative would be able to see records only for 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 position context value.
- The use of an application context in a fine-grained access control package effectively gives you a bind variable in a parsed statement. For example:

  ```
  SELECT * FROM orders_tab
  ```

```
WHERE custno = SYS_CONTEXT('order_entry', 'cust_num')
```

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

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

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

  See Also: *Oracle Database Advanced Application Developer's Guide* for more information about triggers

Compare and contrast this example, which uses an application context within the dynamically generated predicate, with "About Oracle Virtual Private Database Policies" on page 8-5, which uses a subquery in the predicate.

Step 8: Remove the Components for This Example

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

```
CONNECT SYS/AS SYSDBA
Enter password: password
Connected.
```

2. Run the following statements to drop the components for this example:

```
DROP CONTEXT orders_ctx;
DROP USER sysadmin_vpd CASCADE;
DROP USER tbrooke CASCADE;
DROP USER owoods CASCADE;
DROP TABLE scott.orders_tab;
DROP TABLE scott.customers;
```

How Oracle Virtual Private Database Works with Other Oracle Features

This section explains how Oracle Virtual Private Database works with other Oracle Database features.

- How Oracle Virtual Private Database Security Policies Work with Applications
- Using Automatic Reparsing for Fine-Grained Access Control Policy Functions
- Oracle Virtual Private Database Policies and Flashback Query
- Oracle Virtual Private Database and Oracle Label Security Exceptions
- User Models and Oracle Virtual Private Database

How Oracle Virtual Private Database Security Policies Work with Applications

An Oracle Virtual Private Database security policy is applied within the database itself, rather than within an application. Hence, a user trying to access data by using a different application cannot bypass the Oracle Virtual Private Database security policy. Another advantage of creating the security policy in the database is that you maintain it in one central place, rather than maintaining individual security policies in multiple

applications. Oracle Virtual Private Database provides stronger security than application-based security, at a lower cost of ownership.

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

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

Using Automatic Reparsing for Fine-Grained Access Control Policy Functions

By default, queries against objects enabled with fine-grained access control run the policy function to ensure that the most current predicate is used for each policy. For example, in the case of a time-based policy function, in which queries are only allowed between 8:00 a.m. and 5:00 p.m., a cursor execution parsed at noon runs the policy function at that time, ensuring that the policy is consulted again for the query.

Automatic reparsing does not occur when you set the `DBMS_RLS.ADD_POLICY` setting `STATIC_POLICY` to `TRUE` while adding the policy. This setting causes the policy function to return the same predicate.

Oracle Virtual Private Database Policies and Flashback Query

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

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

See Also:

- *Oracle Database Advanced Application Developer's Guide* for more information about the flashback query feature and how to write applications that use it
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_FLASHBACK` PL/SQL package

Oracle Virtual Private Database and Oracle Label Security Exceptions

Be aware of the following exceptions when you use Oracle Virtual Private Database and Oracle Label Security:

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

  However, you can audit SYSDBA actions by enabling auditing upon installation and specifying that this audit trail be stored in a secure location in the operating system. See "Auditing Administrative Users" on page 6-33 for more information. You can also closely monitor the SYS user by using Oracle Database Vault.
- **Database users who were granted the EXEMPT ACCESS POLICY privilege, either directly or through a database role, are exempt from Oracle Virtual Private Database enforcements.** The system privilege EXEMPT ACCESS POLICY allows a user to be exempted from all fine-grained access control policies on any SELECT or DML operation (INSERT, UPDATE, and DELETE). This provides ease of use for administrative activities, such as installation and import and export of the database, through a non-SYS schema.

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

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

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

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

Note:

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

See Also: *Oracle Label Security Administrator's Guide*

User Models and Oracle Virtual Private Database

You can use Oracle Virtual Private Database in the following types of user models:

- **Application users who are also database users.** Oracle Database enables applications to enforce fine-grained access control for each user, regardless of whether that user is a database user or an application user unknown to the database. When application users are also database users, Oracle Virtual Private Database enforcement works as follows: users connect to the database, and then the application sets up application contexts for each session. (You can use the default `USERENV` application context namespace, which provides many parameters for retrieve different types of user session data.) As each session is initiated under a different user name, it can enforce different fine-grained access control conditions for each user.
- **Proxy authentication using OCI or thick JDBC.** Proxy authentication permits different fine-grained access control for each user, because each session (OCI or thick JDBC) is a distinct database session with its own application context.
- **Proxy authentication integrated with Enterprise User Security.** If you have integrated proxy authentication by using Enterprise User Security, you can retrieve user roles and other attributes from Oracle Internet Directory to enforce Oracle Virtual Private Database policies. (In addition, globally initialized application context can also be retrieved from the directory.)
- **Users connecting as One Big Application User.** Applications connecting to the database as a single user on behalf of all users can have fine-grained access control for each user. The user for that single session is often called *One Big Application User*. Within the context of that session, however, an application developer can create a global application context attribute to represent the individual application user (for example, `REALUSER`). Although all database sessions and audit records are created for One Big Application User, the attributes for each session can vary, depending on who the end user is. This model works best for applications with a limited number of users and no reuse of sessions. The scope of roles and database auditing is diminished because each session is created as the same database user. For more information about global application contexts, see "Using Global Application Contexts" on page 7-20.
- **Web-based applications.** Web-based applications typically have hundreds of users. Even when there are persistent connections to the database, supporting data retrieval for many user requests, these connections are not specific to particular Web-based users. Instead, Web-based applications typically set up and reuse connections, to provide scalability, rather than having different sessions for each user. For example, when Web users Jane and Ajit connect to a middle tier application, it may establish a single database session that it uses on behalf of both users. Typically, neither Jane nor Ajit is known to the database. The application is responsible for switching the user name on the connection, so that, at any given time, it is either Jane or Ajit using the session.

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

Table 8–3 summarizes how Oracle Virtual Private Database applies to user models.

Table 8–3 Oracle Virtual Private Database in Different User Models

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

[1] User roles and other attributes, including globally initialized application context, can be retrieved from Oracle Internet Directory to enforce Oracle Virtual Private Database.

[2] Application developers can create a global application context attribute representing individual application users (for example, `REALUSER`), which can then be used for controlling each session attributes, or for auditing.

Finding Information About Oracle Virtual Private Database Policies

Table 8–4 lists data dictionary views that you can use to find information about Oracle Virtual Private Database policies. See *Oracle Database Reference* for more information about these views.

Table 8–4 Data Dictionary Views That Display Information About Virtual Private Database Policies

View	Description
`ALL_POLICIES`	Describes all Oracle Virtual Private Database security policies for objects accessible to the current user.
`ALL_POLICY_CONTEXTS`	Describes the driving contexts defined for the synonyms, tables, and views accessible to the current user. A driving context is an application context used in an Oracle Virtual Private Database policy.
`ALL_POLICY_GROUPS`	Describes the Oracle Virtual Private Database policy groups defined for the synonyms, tables, and views accessible to the current user
`DBA_POLICIES`	Describes all Oracle Virtual Private Database security policies in the database.
`DBA_POLICY_GROUPS`	Describes all policy groups in the database.
`DBA_POLICY_CONTEXTS`	Describes all driving contexts in the database. Its columns are the same as those in `ALL_POLICY_CONTEXTS`.
`USER_POLICIES`	Describes all Oracle Virtual Private Database security policies associated with objects owned by the current user. This view does not display the `OBJECT_OWNER` column.
`USER_POLICY_CONTEXTS`	Describes the driving contexts defined for the synonyms, tables, and views owned by the current user. Its columns (except for `OBJECT_OWNER`) are the same as those in `ALL_POLICY_CONTEXTS`.
`USER_POLICY_GROUPS`	Describes the policy groups defined for the synonyms, tables, and views owned by the current user. This view does not display the `OBJECT_OWNER` column.
`V$VPD_POLICY`	Displays all the fine-grained security policies and predicates associated with the cursors currently in the library cache. This view is useful for finding the policies that were applied to a SQL statement.

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

9

Developing Applications Using the Data Encryption API

This chapter describes how Oracle Database manages data encryption by using the `DBMS_CRYPTO` and `DBMS_SQLHASH` PL/SQL packages. It contains the following topics:

- Securing Sensitive Information
- Security Problems That Encryption Does Not Solve
- Data Encryption Challenges
- Storing Data Encryption by Using the DBMS_CRYPTO Package
- Verifying Data Integrity with the DBMS_SQLHASH Package
- Examples of Using the Data Encryption API
- Finding Information About Encrypted Data

See Also:

- *Oracle Database 2 Day + Security Guide* for an introduction to network encryption
- *Oracle Database Advanced Security Administrator's Guide* for information about using transparent data encryption and tablespace encryption

Securing Sensitive Information

While the Internet poses new challenges in information security, many of them can be addressed by traditional 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 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 Database provided network encryption between database clients and the Oracle database since Oracle Database version 7 . Oracle Advanced Security, an option to Oracle Database, 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.

Encryption is not a remedy for all security problems, but it is an important tool that addresses 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 (unencrypted). 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 secure through proper configuration, but they can also be vulnerable to host break-ins if the host is misconfigured. In well-publicized break-ins, an intruder obtained a large list of credit card numbers by breaking into a database. Had the data been encrypted, the stolen information would have been useless. Encrypting stored data is an important tool in limiting information loss in the rare occurrence that access controls are bypassed.

Security Problems That Encryption Does Not Solve

While there are many good reasons to encrypt data, there are many reasons not to encrypt data. Encryption does not solve all security problems, and may 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 Database Administrator
- 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 users who need to see this data. 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 resource specialists may also need to see employee records for multiple employees.

Typically, you can use access control mechanisms to address security policies that limit data access to those with a need to see it. Oracle Database has provided strong, independently evaluated access control mechanisms for many years. It enables access control enforcement to a fine level of granularity through Virtual Private Database.

Because human resource 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 hinder data access. For example, an employee, his manager, and a human resources clerk may all need to access an employee record. If all employee data is encrypted, then all three must be able to

access the data in unencrypted form. Therefore, the employee, the manager and the human resources 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 hinder the proper or efficient functioning of the application. An additional issue is that it is 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 the `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. In this case, encryption adds to the overhead of decrypting the data before users can read it. If access controls are implemented well, then encryption adds little additional security within the database itself. A user who has privileges to access data within the database has no more nor any less privileges as a result of encryption. Therefore, you should never use encryption to solve access control problems.

Principle 2: Encryption Does Not Protect Against a Malicious Database Administrator

Some organizations, concerned that a malicious user might gain elevated (database administrator) privileges 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 database administrator 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 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 the `DBA` privilege, encryption will not protect against many of them. Examples include corrupting or deleting data, exporting user data to the file system to e-mail the data back to himself to run a password cracker on it, and so on.

Some organizations are concerned that database administrators, typically having all privileges, are able to see all data in the database. These organizations feel that the database administrators should 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 Database supports limited partitioning of `DBA` privileges. Oracle Database 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, you 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 database administrator (such as `CREATE TABLE`, `CREATE USER`, and so on).

Oracle Database 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 database administrators accountable for their actions.

See "Auditing Administrative Users" on page 6-33 for information about ways to audit database administrators.

You can also fine-tune the access and control that database administrators have by using Oracle Database Vault. See *Oracle Database Vault Administrator's Guide* for more information.

The database administrator function is a trusted position. Even organizations with the most sensitive data, such as intelligence agencies, do not typically partition the database administrator function. Instead, they manage their database administrators 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, because otherwise, larger security issues can result. For example, if by encrypting data you corrupt the data, then you create a security problem, the data itself cannot be interpreted, and it may not be recoverable.

You can use encryption to limit the ability of a database administrator or other privileged user to see data in the database. However, it is not a substitute for managing the database administrator privileges properly, or for controlling the use of powerful system privileges. If untrustworthy users have significant privileges, then they can pose multiple threats to an organization, some of them far more significant than viewing unencrypted 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 previous 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 create 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 reencrypted with a new key or keys.

There may be advantages to encrypting data stored off-line. For example, an organization may store backups for a period of 6 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. Because it is not being accessed on-line, performance need not be a consideration. While an Oracle database does not provide this capability, there are vendors who provide 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.

Data Encryption Challenges

In cases where encryption can provide additional security, there are some associated technical challenges, as described in the following sections:

- Encrypting Indexed Data
- Generating Encryption Keys
- Transmitting Encryption Keys
- Storing Encryption Keys
- Changing Encryption Keys
- Encrypting Binary Large Objects

Encrypting Indexed Data

Special difficulties arise when encrypted data 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 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 an index can be used for equality checking (for example, `SELECT * FROM emp WHERE employee_number = '1232456789'`), if the index on that column contains encrypted values, then the index is essentially unusable for any other purpose. You should not encrypt indexed data.

Oracle recommends that you do not use national identity numbers as unique IDs. Instead, use the `CREATE SEQUENCE` statement to generate unique identity numbers. Reasons to avoid using national identity numbers are as follows:

- There are privacy issues associated with overuse of national identity numbers (for example, identity theft).
- Sometimes national identity numbers can have duplicates, as with U.S. social security numbers.

Generating Encryption Keys

Encrypted data is only as secure as the key used for encrypting it. An encryption key must be securely generated using secure cryptographic key generation. Oracle Database provides support for secure random number generation, with 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 Security.

Note: Do not use the DBMS_RANDOM package. The DBMS_RANDOM package generates pseudo-random numbers, which, as Randomness Recommendations for Security (RFC-1750) states that using pseudo-random processes to generate secret quantities can result in pseudo-security.

Be sure to provide the correct number of bytes when you encrypt a key value. For example, you must provide a 16-byte key for the ENCRYPT_AES128 encryption algorithm.

Transmitting Encryption Keys

If the encryption key is to be passed by the application to the database, then you must encrypt it. Otherwise, an intruder could get access to the key as it is being transmitted. Network encryption, such as that provided by Oracle Advanced Security, protects all data in transit from modification or interception, including cryptographic keys.

Storing Encryption Keys

Storing encryption keys 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 an authorized application or user seeking to decrypt the data. At the same time, the key must be inaccessible to someone who is maliciously trying to access encrypted data that he is not supposed to see.

The options available to a developer are:

- Storing the Encryption Keys in the Database
- Storing the Encryption Keys in the Operating System
- Users Managing Their Own Encryption Keys
- Using Transparent Database Encryption and Tablespace Encryption

Storing the Encryption Keys in the Database

Storing the keys in the database cannot always provide infallible security if you are trying to protect against the database administrator accessing encrypted data. An all-privileged database administrator could still access tables containing encryption keys. However, it can often provide good security against the casual curious user 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 the employee social security number (SSN) stored in one of the columns. You could encrypt employee SSN using a key that 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 as simple as using an XOR operation on the employee_number and the birth date of the employee to determine the validity of the values.

As additional protection, PL/SQL source code performing encryption can be wrapped, (using the WRAP utility) which obfuscates (scrambles) the code. The WRAP utility

processes an input SQL file and obfuscates the PL/SQL units in it. For example, the following command uses the `keymanage.sql` file as the input:

```
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.

Oracle Database enables you to obfuscate dynamically generated PL/SQL code. The `DBMS_DDL` package contains two subprograms that allow you to obfuscate dynamically generated PL/SQL program units. For example, the following block uses the `DBMS_DDL.CREATE_WRAPPED` procedure to wrap dynamically generated PL/SQL code.

```
BEGIN
......
SYS.DBMS_DDL.CREATE_WRAPPED(function_returning_PLSQL_code());
......
END;
```

While wrapping is not unbreakable, it makes it harder for an intruder to get access to the encryption key. Even in cases where a different key is supplied for each encrypted data value, you should not embed the key value within a package. Instead, wrap the package that performs the key management (that is, data transformation or padding).

> **See Also:** *Oracle Database PL/SQL Language Reference* for additional information about the `WRAP` command line utility and the `DBMS_DDL` subprograms for dynamic wrapping

An alternative to wrapping the data is 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 that stores employee information and the encrypted SSN. The `employee_number` column is a foreign key to the `ssn_keys` table that stores the encryption keys for the employee SSN. The key stored in the `ssn_keys` table can also be transformed before use (by using an `XOR` operation), so the key itself is not stored unencrypted. If you wrap the procedure, then that can hide the way in which the 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 to this approach is that a user who has `SELECT` access to both the key table and the data table, and who can derive the key transformation algorithm, can break the encryption scheme.

The preceding approach is not infallible, but it is adequate to protect against easy retrieval of sensitive information stored in clear text.

Storing the Encryption 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 is that the database can be broken into from the operating system, then storing the keys in the operating system makes it easier for an intruder to retrieve encrypted data than storing the keys in the database itself.

Users Managing Their Own Encryption Keys

Having the user supply the key assumes the user will be responsible with the key. Considering that 40 percent 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 not recoverable.

If you do decide to have user-supplied or user-managed keys, then you need to ensure you are using network encryption so that the key is not passed from the client to the server in the clear. You also must develop key archive mechanisms, which is also a difficult security problem. Key archives and backdoors create the security weaknesses that encryption is attempting to solve.

Using Transparent Database Encryption and Tablespace Encryption

Transparent database encryption and tablespace encryption provide secure encryption with automatic key management for the encrypted tables and tablespaces. If the application requires protection of sensitive column data stored on the media, then these two types of encryption are a simple and fast way of achieving this.

> **See Also:** *Oracle Database Advanced Security Administrator's Guide* for more information about transparent data encryption

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. You would most likely change the encryption key while the data is not being accessed, which creates another challenge. This is especially true for a Web-based application encrypting credit card numbers, because you do not want to shut down the entire application while you switch encryption keys.

Encrypting Binary Large Objects

Certain data types require more work to encrypt. For example, Oracle Database supports storage of binary large objects (BLOBs), which stores 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 Example of Encryption and Decryption Procedures for BLOB Data on page 9-14.

Storing Data Encryption by Using the DBMS_CRYPTO Package

The DBMS_CRYPTO package provides several ways to address the security issues that were discussed. (For backward compatibility, DBMS_OBFUSCATION_TOOLKIT is also provided.)

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

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 was 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 Database data types, including RAW and large objects (LOBs), such as images and sound. Specifically, it supports BLOBs and CLOBs. 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)
- SHA-1 Cryptographic Hash
- SHA-1 Message Authentication Code (MAC)

Block cipher modifiers are also provided with DBMS_CRYPTO. You can choose from several padding options, including Public Key Cryptographic Standard (PKCS) #5, and from four block cipher chaining modes, including Cipher Block Chaining (CBC). Padding must be done in multiples of eight bytes.

Note:

- DES is no longer recommended by the National Institute of Standards and Technology (NIST).
- Usage of SHA-1 is more secure than MD5.
- Keyed MD5 is not vulnerable.

Table 9–1 compares the DBMS_CRYPTO package features to the other PL/SQL encryption package, the DBMS_OBFUSCATION_TOOLKIT.

Table 9–1 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	SHA-1, SHA-1, MD4	MD5

Table 9–1 (Cont.) DBMS_CRYPTO and DBMS_OBFUSCATION_TOOLKIT Feature Comparison

Package Feature	DBMS_CRYPTO	DBMS_OBFUSCATION_TOOLKIT
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 package, because it is easier to use and supports a range of algorithms that accommodate both new and existing systems. Although 3DES_2KEY and MD4 are provided for backward compatibility, you achieve better security using 3DES, AES, or SHA-1. Therefore, 3DES_2KEY is not recommended.

The DBMS_CRYPTO package includes cryptographic checksum capabilities (MD5), which are useful for comparisons, 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, because it is unsuitable for cryptographic key generation.

Key management is programmatic. That is, the application (or caller of the function) must supply the encryption key. 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.

Note: The DBMS_OBFUSCATION_TOOLKIT is granted to PUBLIC by default. Oracle recommends that you revoke this grant.

While the DBMS_OBFUSCATION_TOOLKIT package can take either VARCHAR2 or RAW data types, it is preferable to use the RAW data type for keys and encrypted data. Storing encrypted data as VARCHAR2 can cause problems if it passes through Globalization Support routines. For example, when transferring a database to another database that uses another character set.

To convert between VARCHAR2 and RAW data types, use the CAST_TO_RAW and CAST_TO_VARCHAR2 functions of the UTL_RAW package.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the DBMS_CRYPTO package
- *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the OBFUSCATION_TOOLKIT package
- *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the UTL_RAW package

Verifying Data Integrity with the DBMS_SQLHASH Package

This section describes the following topics:

- About the DBMS_SQLHASH Package
- Using the DBMS_SQLHASH.GETHASH Function

About the DBMS_SQLHASH Package

The DBMS_SQLHASH package can check data integrity by using hash algorithms. It provides an interface to generate the hash value of the result set returned by a SQL query. Hash values are similar to data fingerprints and are used to ensure data integrity. DBMS_SQLHASH provides support for several industry-standard hashing algorithms, including MD4, MD5, and SHA-1 cryptographic hashes.

Oracle Database installs the DBMS_SQLHASH package in the SYS schema. You can then grant package access to existing users and roles as required.

DBMS_SQLHASH includes the GETHASH function that is used to retrieve the hash value of a query result set. The GETHASH function runs one of the supported cryptographic hash algorithms against the result set of the SQL statement to arrive at a hash value.

You can compare hash values to check whether data was altered. For example, before storing data, Jane runs the DBMS_SQLHASH.GETHASH function against the SQL statement to create a hash value of the SQL result set. When she retrieves the stored data at a later date, she reruns the hash function against the SQL statement using the same algorithm. If the second hash value is identical to the first one, then data was not altered. Any modification to the result set data causes the hash value to be different.

Using the DBMS_SQLHASH.GETHASH Function

The DBMS_SQLHASH.GETHASH function applies one of the supported cryptographic hash algorithms to the result set of the SQL statement.

Syntax

```
DBMS_SQLHASH.GETHASH(
    sqltext IN varchar2,
    digest_type IN BINARY_INTEGER,
    chunk_size IN number DEFAULT 134217728)
   RETURN raw;
```

Parameters

Table 9–2 lists the GETHASH parameters and their descriptions.

Table 9–2 GETHASH Function Parameters

Parameter Name	Description
sqltext	The SQL statement whose result is hashed.
digest_type	Hash algorithm used: HASH_MD4, HASH_MD5, or HASH_SH1
chunk_size	Size of the result chunk when getting the hash When the result set size is large, the GETHASH function breaks it into chunks having a size equal to chunk_size. It generates the hash for each chunk and then uses hash chaining to calculate the final hash. The default chunk_size is 128 megabytes.

Examples of Using the Data Encryption API

This section provides the following examples:

- Example of a Data Encryption Procedure
- Example of AES 256-Bit Data Encryption and Decryption Procedures
- Example of Encryption and Decryption Procedures for BLOB Data

Example of a Data Encryption Procedure

The following sample PL/SQL program (dbms_crypto.sql) shows how to encrypt data. This example code performs the following actions:

- Encrypts a string (VARCHAR2 type) using DES after first converting it into the RAW data type.

  This step is necessary because encrypt and decrypt functions and procedures in DBMS_CRYPTO package work on the RAW data type only, unlike functions and packages in the DBMS_OBFUSCATION_TOOLKIT package.
- Shows how to create a 160-bit hash using SHA-1 algorithm.
- Demonstrates how MAC, a key-dependent one-way hash, can be computed using the 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;
/
```

Example of AES 256-Bit Data Encryption and Decryption Procedures

The following PL/SQL block shows how to encrypt and decrypt a predefined variable named `input_string` using the AES 256-bit algorithm with Cipher Block Chaining and PKCS #5 padding.

```
declare
   input_string       VARCHAR2 (200) := 'Secret Message';
   output_string      VARCHAR2 (200);
   encrypted_raw      RAW (2000);             -- stores encrypted binary text
   decrypted_raw      RAW (2000);             -- stores decrypted binary text
   num_key_bytes      NUMBER := 256/8;        -- key length 256 bits (32 bytes)
   key_bytes_raw      RAW (32);               -- stores 256-bit encryption key
   encryption_type    PLS_INTEGER :=          -- total encryption type
                         DBMS_CRYPTO.ENCRYPT_AES256
                       + DBMS_CRYPTO.CHAIN_CBC
                       + DBMS_CRYPTO.PAD_PKCS5;
begin
   DBMS_OUTPUT.PUT_LINE ('Original string: ' || input_string);
   key_bytes_raw := DBMS_CRYPTO.RANDOMBYTES (num_key_bytes);
   encrypted_raw := DBMS_CRYPTO.ENCRYPT
      (
         src => UTL_I18N.STRING_TO_RAW (input_string, 'AL32UTF8'),
         typ => encryption_type,
         key => key_bytes_raw
      );
    -- The encrypted value in the encrypted_raw variable can be used here
   decrypted_raw := DBMS_CRYPTO.DECRYPT
```

```
      (
         src => encrypted_raw,
         typ => encryption_type,
         key => key_bytes_raw
      );
   output_string := UTL_I18N.RAW_TO_CHAR (decrypted_raw, 'AL32UTF8');
   DBMS_OUTPUT.PUT_LINE ('Decrypted string: ' || output_string);
end;
```

Example of Encryption and Decryption Procedures for BLOB Data

The following sample PL/SQL program (`blob_test.sql`) shows how to encrypt and decrypt BLOB data. This example code does the following, and prints 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;
```

Finding Information About Encrypted Data

Table 9–3 lists data dictionary views that you can query to access information about encrypted data. See *Oracle Database Reference* for detailed information about these views.

Table 9–3 Views That Display Information about Encrypted Data

View	Description
ALL_ENCRYPTED_COLUMNS	Describes encryption algorithm information for all encrypted columns in all tables accessible to the user
DBA_ENCRYPTED_COLUMNS	Describes encryption algorithm information for all encrypted columns in the database
USER_ENCRYPTED_COLUMNS	Describes encryption algorithm information for all encrypted columns in all tables in the schema of the user
V$ENCRYPTED_TABLESPACES	Displays information about the tablespaces that are encrypted
V$ENCRYPTION_WALLET	Displays information on the status of the wallet and the wallet location for transparent data encryption
V$RMAN_ENCRYPTION_ALGORITHMS	Displays supported encryption algorithms.

10

Keeping Your Oracle Database Secure

This chapter provides a set of guidelines to keep your Oracle database secure. It includes the following topics:

- About the Security Guidelines in This Chapter
- Downloading Security Patches and Contacting Oracle Regarding Vulnerabilities
- Guidelines for Securing User Accounts and Privileges
- Guidelines for Securing Roles
- Guidelines for Securing Passwords
- Guidelines for Securing Data
- Guidelines for Securing a Database Installation and Configuration
- Guidelines for Securing the Network
- Guidelines for Auditing
- Addressing the CONNECT Role Change

About the Security Guidelines in This Chapter

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

Oracle Database leads the industry in security. To maximize the security features offered by Oracle Database in any business environment, it is imperative that the database itself be well protected.

Security guidelines provide advice about how to configure Oracle Database to be secure by adhering to and recommending industry-standard and advisable security practices for operational database deployments. Many of the guidelines described in this section address common regulatory requirements such as those described in the Sarbanes-Oxley Act. For more information about how Oracle Database addresses regulatory compliance, protection of personally identifiable information, and internal threats, visit:

```
http://www.oracle.com/technology/deploy/security/db_
security/index.html
```

Downloading Security Patches and Contacting Oracle Regarding Vulnerabilities

This section includes the following topics:

- Applying Security Patches and Workaround Solutions
- Contacting Oracle Security Regarding Vulnerabilities in Oracle Database

Applying Security Patches and Workaround Solutions

Always apply all relevant 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.

Periodically check the security site on Oracle Technology Network for details about security alerts released by Oracle at

`http://www.oracle.com/technology/deploy/security/alerts.htm`

Also check the Oracle Worldwide Support Service site, Oracle*MetaLink*, for details about available and upcoming security-related patches at

`http://metalink.oracle.com`

Contacting Oracle Security Regarding Vulnerabilities in Oracle Database

If you are an Oracle customer or an Oracle partner, use Oracle*MetaLink* to submit a Service Request on any potential Oracle product security vulnerability. Otherwise, send an e-mail to `secalert_us@oracle.com` with a complete description of the problem, including product version and platform, together with any scripts and examples. Oracle encourages those who want to contact Oracle Security to employ e-mail encryption, using our encryption key.

Guidelines for Securing User Accounts and Privileges

Follow these guidelines to secure user accounts and privileges:

1. **Practice the principle of least privilege.**

   Oracle recommends the following guidelines:

   a. **Grant necessary privileges only.**

      Do not provide database users more privileges than are necessary. In other words, the *principle of least privilege* is that users be given only those privileges that are actually required to efficiently perform their jobs.

      To implement this principle, restrict the following as much as possible:

      – The number of `SYSTEM` and `OBJECT` privileges granted to database users.

      – The number of people who are allowed to make `SYS`-privileged connections to the database.

      – The number of users who are granted the `ANY` privileges, such as the `DROP ANY TABLE` privilege. For example, there is generally no need to grant `CREATE ANY TABLE` privileges to a non-DBA-privileged user.

      – The number of users who are allowed to perform actions that create, modify, or drop database objects, such as the `TRUNCATE TABLE`, `DELETE TABLE`, `DROP TABLE` statements, and so on.

b. Do not allow non-administrative users access to objects owned by the SYS schema.

Do not allow users to alter table rows or schema objects in the SYS schema, because doing so can compromise data integrity. Limit the use of statements such as DROP TABLE, TRUNCATE TABLE, DELETE, INSERT, or similar object-modification statements on SYS objects only to highly privileged administrative users.

The SYS schema owns the data dictionary. You can protect the data dictionary by setting the 07_DICTIONARY_ACCESSIBILITY parameter to FALSE. See Guideline 1 under "Guidelines for Securing Data" on page 10-8 for more information.

c. Revoke unnecessary privileges from the PUBLIC user group.

The PUBLIC user group represents all users in the database. 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. These privileges include EXECUTE on various PL/SQL packages, potentially enabling someone with minimal privileges to access and execute functions that this user would not otherwise be permitted to access directly.

d. Restrict permissions on run-time facilities.

Many Oracle Database products use run-time facilities, such as Oracle Java Virtual Machine (OJVM). Do not assign all permissions to a database run-time facility. Instead, grant specific permissions to the explicit document root file paths for facilities that might run files and packages outside the database.

Here is an example of a vulnerable run-time call, which individual files are specified:

```
call dbms_java.grant_permission('wsmith',
'SYS:java.io.FilePermission','<<ALL FILES>>','read');
```

Here is an example of a better (more secure) run-time call, which specifies a directory path instead:

```
call dbms_java.grant_permission('wsmith',
'SYS:java.io.FilePermission','<<actual directory path>>','read');
```

2. Lock and expire predefined user accounts.

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

If a manual (without using Database Configuration Assistant) installation of Oracle Database is performed, then no default database users are locked upon successful installation of the database server. 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 use the Database Configuration Assistant, you should *lock* and *expire* all default database user accounts. Oracle Database provides SQL statements to perform these operations. For example:

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

See *Oracle Database SQL Language Reference* for more information about the `ALTER USER` statement.

Installing additional products and components after the initial installation also results in creating more default database accounts. Database Configuration Assistant automatically locks and expires all additionally created database user accounts. Unlock only those accounts that 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 these operations.

If any default database user account other than the ones left open is required for any reason, then a database administrator (DBA) needs to unlock and activate that account with a new, secure password.

See *Oracle Database 2 Day + Security Guide* for a description of the predefined user accounts that are created when you install Oracle Database.

If a default database user account, other than the ones left open, is required for any reason, then a database administrator (DBA) can unlock and activate that account with a new, secure password.

Oracle Enterprise Manager Accounts

The preceding list of accounts depends on whether or not you install Oracle Enterprise Manager. If you do, the `SYSMAN` and `DBSNMP` accounts are open, unless you configure Oracle Enterprise Manager for central administration. In this case, the `SYSMAN` account (if present) will be locked.

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

3. **Use the following views to ensure that access is granted. Only users and roles that need access should be granted access to them.**
   - `DBA_*`
   - `DBA_ROLES`
   - `DBA_SYS_PRIVS`
   - `DBA_ROLE_PRIVS`
   - `DBA_TAB_PRIVS`
   - `SYS.AUD$` (if auditing is enabled)
   - `SYS.FGA_LOG$`
4. **Monitor the granting of the following privileges only to users and roles who need these privileges.**

   By default, Oracle Database audits the following privileges:
   - `ALTER SYSTEM`
   - `AUDIT SYSTEM`
   - `CREATE EXTERNAL JOB`

   Oracle recommends that you also audit the following privileges:
   - `ALL PRIVILEGES`
   - `BECOME USER`
   - `CREATE LIBRARY`

- CREATE PROCEDURE
- DBMS_BACKUP_RESTORE package
- EXECUTE to DBMS_SYS_SQL
- SELECT ANY TABLE
- SELECT on PERFSTAT.STATS$SQLTEXT
- SELECT on PERFSTAT.STATS$SQL_SUMMARY
- SELECT on SYS.USER$
- SELECT on SYS.SOURCE$
- Privileges that have the WITH ADMIN clause
- Privileges that have the WITH GRANT clause
- Privileges that have the CREATE keyword

5. **Revoke access to the following:**
   - The SYS.USER_HISTORY$ view from all users except SYS and DBA accounts
   - The RESOURCE role from typical application accounts
   - The CONNECT role from typical application accounts
   - The DBA role from users who do not need this role
6. **Grant privileges only to roles.**
7. **Limit the proxy account (for proxy authorization) privileges to CREATE SESSION only.**
8. **Use secure application roles to protect roles that are enabled by application code.**

   Secure application roles allow you to define a set of conditions, within a PL/SQL package, that determine whether or not a user can log on to an application. Users do not need to use a password with secure application roles.

   Another approach to protecting roles from being enabled or disabled in an application is the use of role passwords. This approach prevents a user from directly accessing the database in SQL (rather than the application) to enable the privileges associated with the role. However, Oracle recommends that you use secure application roles instead, to avoid having to manage another set of passwords.
9. **Discourage users from using the NOLOGGING clause in SQL statements.**

   In some SQL statements, the user has the option of specifying the NOLOGGING clause, which indicates that the database operation is not logged in the online redo log file. Even though the user specifies the clause, a redo record is still written to the online redo log file. However, there is no data associated with this record. Because of this, using NOLOGGING has the potential for malicious code to be entered can be accomplished without an audit trail.

Guidelines for Securing Roles

Follow these guidelines when managing roles:

1. **Grant a role to users only if they need all privileges of the role.**

Roles (groups of privileges) are useful for quickly and easily granting permissions to users. Although you can use Oracle-defined roles, you have more control and continuity if you create your own roles containing only the privileges pertaining to your requirements. Oracle may change or remove the privileges in an Oracle Database-defined role, as it has with the `CONNECT` role, which now has only the `CREATE SESSION` privilege. Formerly, this role had eight other privileges. Both `CONNECT` and `RESOURCE` roles will be deprecated in future Oracle releases.

Ensure that the roles you define contain only the privileges that reflect job responsibility. If your application users do not need all the privileges encompassed by an existing role, then apply a different set of roles that supply just the correct privileges. Alternatively, create and assign a more restricted role.

For example, it is imperative to strictly limit the privileges of user `SCOTT`, because this is a well known account that may be vulnerable to intruders. Because the `CREATE DBLINK` privilege allows access from one database to another, drop its privilege for `SCOTT`. Then, drop the entire role for the user, because privileges acquired by means of a role cannot be dropped individually. Re-create your own role with only the privileges needed, and grant that new role to that user. Similarly, for better security, drop the `CREATE DBLINK` privilege from all users who do not require it.

2. **Do not grant user roles to application developers.**

   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. Remember that roles are not enabled within stored procedures except for invoker's right procedures. See "How Roles Work in PL/SQL Blocks" on page 4-8 for information about this topic.

3. **Create and assign roles specific to each Oracle Database installation.**

   This principle enables the organization to retain detailed control of its roles and privileges. This also avoids the necessity to adjust if Oracle Database changes or removes Oracle Database-defined roles, as it has with `CONNECT`, which now has only the `CREATE SESSION` privilege. Formerly, it also had eight other privileges. Both `CONNECT` and `RESOURCE` roles will be deprecated in future Oracle Database versions.

4. **For enterprise users, create global roles.**

   Global roles are managed by an enterprise directory service, such as Oracle Internet Directory. See the following sections for more information about global roles:

   - "Configuring Global User Authentication and Authorization" on page 3-25
   - "Global Role Authorization by an Enterprise Directory Service" on page 4-16
   - *Oracle Database Enterprise User Security Administrator's Guide*

Guidelines for Securing Passwords

When you create a user account, Oracle Database assigns a default password policy for that user. The password policy defines rules for how the password should be created, such as a minimum number of characters, when it expires, and so on. You can strengthen passwords by using password policies. See also "Configuring Password Protection" on page 3-1 for additional ways to protect passwords.

Follow these guidelines to further strengthen passwords:

1. Choose passwords carefully.

   "How Oracle Database Checks the Complexity of Passwords" on page 3-7 describes the minimum requirements for passwords. Follow these additional guidelines when you create or change passwords:

   - Make the password between 8 and 30 characters.
   - Use the database character set for the password's characters, which can include the underscore (_), dollar ($), and number sign (#) characters.
   - In addition to including at least 1 digit and 1 alphabetic character, include at least 1 punctuation mark in the password.
   - Do not start the password with a number.
   - Do not use Oracle reserved words in the password.

     See *Oracle Database SQL Language Reference* for a list of Oracle Database reserved words.
   - Do not include the password in a dictionary or in a name (for example, an object name).

   Oracle Database provides a password complexity verification routine, the PL/SQL script `UTLPWDMG.SQL`, that you can run to check whether or not passwords are sufficiently complex. Ideally, edit the `UTLPWDMG.SQL` script to provide stronger password protections. See also "Enforcing Password Complexity Verification" on page 3-7 for a sample routine that you can use to check passwords.

2. **Change default user passwords.**

   Oracle Database installs with a set of predefined, default user accounts. Security is most easily broken when a default database user account still has a default password *even after installation*. This is particularly true for the user account `SCOTT`, which is a well known account that may be vulnerable to intruders. In Oracle Database 11*g* Release 1 (11.1), default accounts are installed locked with the passwords expired, but if you have upgraded from a previous release, you may still have accounts that use default passwords.

   To find user accounts that have default passwords, query the `DBA_USERS_WITH_DEFPWD` data dictionary view. See "Finding User Accounts That Have Default Passwords" on page 3-3 for more information.

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

   You can use the same or different passwords for the `SYS`, `SYSTEM`, `SYSMAN`, and `DBSNMP` administrative accounts. Oracle recommends that you use different passwords for each. In any Oracle environment (production or test), assign strong, secure, and distinct passwords to these administrative accounts. If you use Database Configuration Assistant to create a new database, then it requires you to enter passwords for the `SYS` and `SYSTEM` accounts, disallowing the default passwords `CHANGE_ON_INSTALL` and `MANAGER`.

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

   See *Oracle Database 2 Day + Security Guide* for information about changing a default password.

4. **Enforce password management.**

   Apply basic password management rules (such as password length, history, complexity, and so forth) to all user passwords. Oracle Database has password

policies enabled for the default profile. Guideline 1 in this section lists these password policies. *Oracle Database 2 Day + Security Guide* lists initialization parameters that you can use to further secure user passwords.

You can find information about user accounts by querying the DBA_USERS view. This view contains a column for passwords, but for better security, Oracle Database encrypts (disguises) the data in this column. The DBA_USERS view provides useful information such as the user account status, whether the account is locked, and password versions. You can query DBA_USERS as follows:

```
sqlplus system
Enter password: password
Connected.
SQL> SELECT * FROM DBA_USERS;
```

Oracle also recommends, if possible, using Oracle Advanced Security (an option to Oracle Database Enterprise Edition) with network authentication services (such as Kerberos), token cards, smart cards, or X.509 certificates. These services provide strong authentication of users, and provide protection against unauthorized access to Oracle Database.

5. **Do not store user passwords in clear text in Oracle tables.**

   For better security, do not store passwords in clear text (that is, human readable) in Oracle tables. You can correct this problem by encrypting the table column that contains the password. See *Oracle Database 2 Day + Security Guide* for information about how to use transparent data encryption to encrypt a table column.

   When you create or modify a password for a user account, Oracle Database automatically encrypts it. If you query the DBA_USERS view to find information about a user account, the data in the PASSWORD column is encrypted.

Guidelines for Securing Data

Follow these guidelines to secure data on your system:

1. **Enable data dictionary protection.**

   Oracle recommends that you protect the data dictionary to prevent users that have the ANY system privilege from using those privileges on the data dictionary. Altering or manipulating the data in data dictionary tables can permanently and detrimentally affect the operation of a database.

   To enable data dictionary protection, set the following initialization parameter to FALSE (which is the default) in the init*sid*.ora control file:

   ```
   07_DICTIONARY_ACCESSIBILITY = FALSE
   ```

   You can set the 07_DICTIONARY_ACCESSIBILITY parameter in a server parameter file. For more information about server parameter files, see *Oracle Database Administrator's Guide*.

   After you set O7_DICTIONARY_ACCESSIBILTY to FALSE, only users who have the SELECT ANY DICTIONARY privilege and those authorized users making DBA-privileged (for example CONNECT / AS SYSDBA) connections can use the ANY system privilege on the data dictionary. If O7_DICTIONARY_ACCESSIBILITY parameter is not set to FALSE, then any user with the DROP ANY TABLE (for example) system privilege will be able to drop parts of the data dictionary. However, if a user *needs* view access to the data dictionary, then you can grant that user the SELECT ANY DICTIONARY system privilege.

Note:

- In a default installation, the `O7_DICTIONARY_ACCESSIBILITY` parameter is set to `FALSE`. However, in Oracle8*i*, this parameter is set to `TRUE` by default, and must be changed to `FALSE` to enable this security feature.
- The `SELECT ANY DICTIONARY` privilege is not included in the `GRANT ALL PRIVILEGES` statement, but you can grant it through a role. Chapter 4, "Configuring Privilege and Role Authorization" describes roles in detail.

2. **Restrict operating system access.**

   Follow these guidelines:

   - 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 computer to the least privileges required for a user to perform necessary tasks.
   - Restrict 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.
   - Restrict symbolic links. Ensure that when you provide 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 database administrator or trusted account, such as *root*.

     This recommendation applies to all types of files: data files, log files, trace files, external tables, BFILE data types, and so on.

Guidelines for Securing a Database Installation and Configuration

For this release, changes were made to the default configuration of Oracle Database to make it more secure. The recommendations in this section augment the new, secure default configuration.

Follow these guidelines to secure the database installation and configuration:

1. **Before you begin an Oracle Database installation on UNIX systems, ensure that the umask value is 022 for the Oracle owner account.**
2. **Install only what is required.**

   Options and Products: The Oracle Database CD pack contains products and options in addition to the database. Install additional products and options only as necessary. Use the Custom Installation feature to avoid installing unnecessary products, or perform a typical installation, and then deinstall options and products that are not required. There is no need to maintain additional products and options if they are not being used. They can always be properly installed, as required.

   Sample Schemas: Oracle Database provides sample schemas to provide a common platform for examples. If your database will be used in a production environment, then do not install the sample schema. If you have installed the sample schema on a test database, then before going to production, remove or

relock the sample schema accounts. See *Oracle Database Sample Schemas* for more information about the sample schemas.

3. **During installation, when you are prompted for a password, create a secure password.**

   Follow Guidelines 1, 2, and 3 in "Guidelines for Securing Passwords" on page 10-6.

4. **Immediately after installation, lock and expire default user accounts.**

   See Guideline 2 in "Guidelines for Securing User Accounts and Privileges" on page 10-2.

Guidelines for Securing the Network

Security for network communications is improved by using client, listener, and network guidelines to ensure thorough protection. Using SSL is an essential element in these lists, enabling top security for authentication and communications.

These guidelines are as follows:

- Securing the Client Connection
- Securing the Network Connection
- Securing a Secure Sockets Layer Connection

Securing the Client Connection

Because authenticating client computers is problematic over the Internet, typically, user authentication is performed 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 guidelines improve the security of client connections:

1. **Enforce access controls effectively and authenticate clients stringently.**

   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 database to accept the client operating system user name received over an unsecure connection and use it for account access. Because clients, such as PCs, are not trusted to perform operating system authentication properly, it is poor security practice to use 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 means that the database will not trust that the client has already authenticated, and will therefore apply its standard authentication processes.

2. **Configure the connection to use Secure Sockets Layer (SSL).**

Using SSL communication makes eavesdropping difficult and enables the use of certificates for user and server authentication. To learn how to configure SSL, see *Oracle Database Advanced Security Administrator's Guide*.

3. **Set up certificate authentication for clients and servers.**

   See *Oracle Database Advanced Security Administrator's Guide* for more information about ways to manage certificates.

4. **Monitor the users who access your systems.**

   Authenticating client computers over the Internet is problematic. Perform 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:

   a. Configure the connection to use Secure Sockets Layer (SSL). Using SSL communication makes eavesdropping unprofitable, and enables the use of certificates for user and server authentication. To learn how to configure SSL, see *Oracle Database Advanced Security Administrator's Guide*.

   b. Set up certificate authentication for clients and servers such that:

      – The organization is identified by unit and certificate issuer, and the user is identified by distinguished name and certificate issuer.

      – Applications test for expired certificates.

      – Certificate revocation lists are audited.

   See *Oracle Database Advanced Security Administrator's Guide* for more information about ways to manage certificates.

Securing the Network Connection

Protecting the network and its traffic from inappropriate access or modification is the essence of network security. You should 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 breach of security. In addition, monitor and audit to detect either increased threat levels or successful penetration.

To manage network connections, you can use Oracle Net Manager. For an introduction to using Oracle Net Manager, see *Oracle Database 2 Day DBA*. See also *Oracle Database Net Services Administrator's Guide*.

The following practices improve network security:

1. **Use Secure Sockets Layer (SSL) when administering the listener.**

   See "Securing a Secure Sockets Layer Connection" on page 10-14 for more information.

2. **Monitor listener activity.**

   You can monitor listener activity by using Enterprise Manager Database Control. In the Database Control home page, under General, click the link for your listener. The Listener page appears. This page provides detailed information, such as the category of alert generated, alert messages, when the alert was triggered, and so on. This page provides other information as well, such as performance statistics for the listener.

3. **Prevent online administration by requiring the administrator to have write privileges on the listener.ora file and the listener password.**

a. Add or alter this line in the `listener.ora` file:

```
ADMIN_RESTRICTIONS_LISTENER=ON
```

b. Use `RELOAD` to reload the configuration.

c. Use SSL when administering the listener, by making the TCPS protocol the first entry in the address list as follows:

```
LISTENER=
  (DESCRIPTION=
    (ADDRESS_LIST=
      (ADDRESS=
        (PROTOCOL=tcps)
        (HOST = ed-pdsun1.us.oracle.com)
        (PORT = 8281)))
```

To administer the listener remotely, you define the listener in the `listener.ora` file on the client computer. For example, to access listener USER281 remotely, use the following configuration:

```
user281 =
  (DESCRIPTION =
    (ADDRESS =
      (PROTOCOL = tcps)
      (HOST = ed-pdsun1.us.oracle.com)
      (PORT = 8281))
    )
  )
```

For more information about the parameters in `listener.ora`, see *Oracle Database Net Services Reference*.

4. **Do not set the listener password.**

   Ensure that the password has not been set in the `listener.ora` file. The local operating system authentication secures the listener administration. The remote listener administration is disabled when the password has not been set.

5. **When a host computer has multiple IP addresses associated with multiple network interface controller (NIC) cards, configure the listener to the specific IP address.**

   This allows the listener to listen on all the IP addresses. You can restrict the listener to listen on a specific IP address. Oracle recommends that you specify the specific IP addresses on these types of computers, rather than allowing the listener to listen on all IP addresses. Restricting the listener to specific IP addresses helps to prevent an intruder from stealing a TCP end point from under the listener process.

6. **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 an agent) from inheriting the ability to perform read or write operations. The owner of this separate listener process should not be the owner that installed Oracle Database or executes the Oracle Database instance (such as `ORACLE`, the default owner).

   For more information about configuring external procedures in the listener, see *Oracle Database Net Services Administrator's Guide*.

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

See *Oracle Database 2 Day + Security Guide* and *Oracle Database Advanced Security Administrator's Guide* for more information about network data encryption.

8. **Use a firewall.**

   Appropriately placed and configured firewalls can prevent outside access to your intranet when you allow internal users to have Internet access.

   - Keep the database server behind a firewall. Oracle Database network infrastructure, Oracle Net (formerly known as Net8 and SQL*Net), provides support for a variety of firewalls from various vendors. Supported proxy-enabled firewalls include Gauntlet from Network Associates and Raptor from Axent. Supported packet-filtering firewalls include PIX Firewall from Cisco, and supported stateful inspection firewalls (more sophisticated packet-filtered firewalls) include Firewall-1 from CheckPoint.
   - 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 such as Oracle Connection Manager to manage multiplex multiple client network sessions through a single network connection to the database. It can filter on source, destination, and host name. This product 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 falsified.)

9. **Prevent unauthorized administration of the Oracle listener.**

   Create a well-formed password for the Oracle listener to prevent remote configuration of the Oracle listener. See Guideline 1 in "Guidelines for Securing Passwords" on page 10-6 for advice on creating strong, secure passwords. For more information about the listener, see *Oracle Database Net Services Administrator's Guide*.

10. **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. To use this feature, set the following `sqlnet.ora` configuration file parameters:

    ```
    tcp.validnode_checking = YES

    tcp.excluded_nodes = {list of IP addresses}

    tcp.invited_nodes = {list of IP addresses}
    ```

    The `tcp.validnode_checking` parameter enables the feature. The the `tcp.excluded_nodes` and `tcp.invited_nodes` parameters deny and enable specific client IP addresses from making connections to the Oracle listener. This helps to prevent potential Denial of Service attacks.

    You can use Oracle Net Manager to configure these parameters. See *Oracle Database Net Services Administrator's Guide* for more information.

11. **Encrypt network traffic.**

    If possible, use Oracle Advanced Security to encrypt network traffic among clients, databases, and application servers. *Oracle Database 2 Day + Security Guide* provides an introduction to network encryption. For detailed information about network encryption, see *Oracle Database Advanced Security Administrator's Guide*.

12. **Secure the host operating system (the system on which Oracle Database is installed).**

    Secure the host operating system by disabling all unnecessary operating system services. Both UNIX and Windows provide a variety of operating system services, most of which are not necessary for typical deployments. These 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.

Securing a Secure Sockets Layer Connection

Secure Sockets Layer (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 through certificates and, if necessary, encryption. Good security practices maximize protection and minimize gaps or disclosures that threaten security. The following guidelines show the cautious attention to detail necessary for the successful use of SSL. For detailed information about Oracle SSL configuration, see *Oracle Database Advanced Security Administrator's Guide*.

1. **Ensure that configuration files (for example, 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. The port can also be specified in the URL, for example:

   ```
   https://secure.server.com:4445/
   ```

   If a firewall is in use, then it too must use the same ports 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).**

   An 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, the database (on one side) and the user or application (on the other) must have the same SSL mode.**

   The mode can specify either client or server authentication (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 in use.**

5. **Enable DN matching for both the server and client, to prevent the server from falsifying its identity to the client during connections.**

   This setting ensures that the server identity is correct by matching its global database name against the DN from the server certificate.

   You can enable DN matching in the `tnsnames.ora` file. For example:

   ```
   set:SSL_SERVER_CERT_DN="cn=finance,cn=OracleContext,c=us,o=acme"
   ```

   Otherwise, a client application would not check the server certificate, which could allow the server to falsify its identity.

6. **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.**

   Note: A server without SSL does not require a pass phrase.

   If you decide your server is secure enough, you could remove the encryption from the RSA private key while preserving the original file. This enables system boot scripts to start the database server, because no pass phrase is needed. Ideally, restrict permissions to the root user only, and have the Web server start as `root`, but then log on as another user. Otherwise, anyone who gets this key can impersonate you on the Internet, or decrypt the data that was sent to the server.

   See Also:

   - *Oracle Database Advanced Security Administrator's Guide* for general SSL information, including configuration
   - *Oracle Database Net Services Reference* for TCP-related parameters in `sqlnet.ora`

Guidelines for Auditing

This section describes the following guidelines for auditing:

- Enabling Default Auditing of SQL Statements and Privileges
- Keeping Audited Information Manageable
- Auditing Typical Database Activity
- Auditing Suspicious Database Activity

Enabling Default Auditing of SQL Statements and Privileges

When you create a new database, you have the option to enable the auditing of a select set of SQL statements and privileges. Oracle recommends that you enable default auditing. Auditing is an effective method of enforcing strong internal controls so that your site can meet its regulatory compliance requirements, as defined in the Sarbanes-Oxley Act.

See Also: "Using Default Auditing for Security-Relevant SQL Statements and Privileges" on page 6-10

Keeping Audited Information Manageable

Although auditing is relatively inexpensive, limit the number of audited events as much as possible. This minimizes the performance impact on the execution of audited statements and the size of the audit trail, making it easier to analyze and understand.

Follow these guidelines when devising an auditing strategy:

1. **Evaluate your reason 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

strategy 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.

2. **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 using 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, then determine exactly what types of activities you want to track, audit only the activities of interest, and audit only for the amount of time necessary to gather the information that you want. As another example, do not audit *objects* if you are only interested in logical I/O information for each session.

Auditing Typical Database Activity

When your purpose for auditing is to gather historical information about particular database activities, use the following guidelines:

1. **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. You can audit specific actions by using fine-grained auditing, which is described in "Using Fine-Grained Auditing to Monitor Specific Activities" on page 6-38.

2. **Archive audit records and purge the audit trail.**

   After you collect the required information, archive the audit records of interest and then purge the audit trail of this information.

   To archive audit records, you can copy the relevant records to a normal database table, for example, using `INSERT INTO` *`table`* `SELECT ... FROM SYS.AUD$ ...` for the standard audit trail. (For fine-grained audit records, you can find their records in the `SYS.FGA_LOG$` table.) Alternatively, you can export the audit trail table to an operating system file. *Oracle Database Utilities* explains how to export tables by using Data Pump.

   To purge audit records, you can delete standard audit records from the `SYS.AUD$` table and fine-grained audit records from the `SYS.FGA_LOG$` table. For example, to delete *all* audit records from the standard 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';
   ```

   See "Controlling the Growth and Size of the Standard Audit Trail" on page 6-17 for more information about managing the standard audit trail.

3. **Remember your company's 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 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:

1. **First audit generally, and then specifically.**

   When you start to audit for suspicious database activity, often not much information is available to target specific users or schema objects. Therefore, set audit options more generally at first, that is, by using the standard audit options described in Chapter 6, "Configuring Auditing" explains how you can use the standard audit options to audit SQL statements, schema objects, privileges, and so on.

   After you have recorded and analyzed the preliminary audit information, disable general auditing, and then audit specific actions. You can use fine-grained auditing, which is described in "Using Fine-Grained Auditing to Monitor Specific Activities" on page 6-38, to audit specific actions. Continue this process until you have gathered enough evidence to draw conclusions about the origin of the suspicious database activity.

2. **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. You can audit the standard audit trail by using the `AUDIT` SQL statement. For example:

   ```
   sqlplus "sys/as sysdba"
   Enter password: password
   SQL> AUDIT SELECT ON SYS.AUD$ BY ACCESS;
   ```

   See also "Auditing the Standard Audit Trail" on page 6-19.

   To audit the fine-grained audit trail, as user `SYS`, you would enter the following statement:

   ```
   AUDIT SELECT ON SYS.FGA_LOG$ BY ACCESS;
   ```

Addressing the CONNECT Role Change

The `CONNECT` role was introduced with Oracle Database version 7, which added new and robust support for database roles. The `CONNECT` role is used in sample code, applications, documentation, and technical papers.

This section discusses the effects of changed `CONNECT` privileges in the following sections:

- Why Was the CONNECT Role Changed?
- How the CONNNECT Role Change Affects Applications
- How the CONNECT Role Change Affects Users
- Approaches to Addressing the CONNECT Role Change

Why Was the CONNECT Role Changed?

The CONNECT role was originally established with the following privileges:

```
ALTER SESSION              CREATE SESSION
CREATE CLUSTER             CREATE SYNONYM
CREATE DATABASE LINK       CREATE TABLE
CREATE SEQUENCE            CREATE VIEW
```

Beginning in Oracle Database 10*g* Release 2, the CONNECT role has only the CREATE SESSION privilege, all other privileges are removed.

Although the CONNECT role was frequently used to provision new accounts in Oracle Database, connecting to the database does not require all those privileges. Making this change enables you to enforce good security practices more easily.

Each user should have only the privileges needed to perform his or her tasks, an idea called the principle of least privilege. Least privilege mitigates risk by limiting privileges, so that it remains easy to do what is needed while concurrently reducing the ability to do inappropriate things, either inadvertently or maliciously.

How the CONNNECT Role Change Affects Applications

The effects of the changes to the CONNECT role can be seen in database upgrades, account provisioning, and installation of applications using new databases.

How the CONNECT Role Change Affects Database Upgrades

Upgrading your existing Oracle database to Oracle Database 10gR2 automatically changes the CONNECT role to have only the CREATE SESSION privilege. Most applications are not affected because the applications objects already exist: no new tables, views, sequences, synonyms, clusters, or database links need to be created.

Applications that create tables, views, sequences, synonyms, clusters, or database links, or that use the ALTER SESSION command dynamically, may fail due to insufficient privileges.

How the CONNECT Role Change Affects Account Provisioning

If your application or DBA grants the CONNECT role as part of the account provisioning process, then only CREATE SESSION privileges are included. Any additional privileges must be granted either directly or through another role.

This issue can be addressed by creating a new customized database role.

> **See Also:** Approaches to Addressing the CONNECT Role Change on page 10-19

How the CONNECT Role Change Affects Applications Using New Databases

New databases created using the Oracle Database 10*g* Release 2 (10.2) Utility (DBCA), or using database creation templates generated from DBCA, define the CONNECT role with only the CREATE SESSION privilege. Installing an application to use a new database may fail if the database schema used for the application is granted privileges solely through the CONNECT role.

How the CONNECT Role Change Affects Users

The change to the CONNECT role affects three classes of users differently: general users, application developers, and client/server applications.

How the CONNECT Role Change Affects General Users

The new CONNECT role supplies only the CREATE SESSION privilege. Users who connect to the database to use an application are not affected, because the CONNECT role still has the CREATE SESSION privilege.

However, appropriate privileges will not be present for a certain set of users if they are provisioned solely with the CONNECT role. These are users who create tables, views, sequences, synonyms, clusters, or database links, or use the ALTER SESSION command. The privileges they need are no longer provided with the CONNECT role. To authorize the additional privileges needed, the database administrator must create and apply additional roles for the appropriate privileges, or grant them directly to the users who need them.

Note that the ALTER SESSION privilege is required for setting events. Few database users should require the alter session privilege.

```
SQL> ALTER SESSION SET EVENTS ........
```

The alter session privilege is *not* required for other alter session commands.

```
SQL> ALTER SESSION SET NLS_TERRITORY = FRANCE;
```

How the CONNECT Role Change Affects Application Developers

Application developers provisioned solely with the CONNECT role do not have appropriate privileges to create tables, views, sequences, synonyms, clusters, or database links, nor to use the ALTER SESSION statement. The database administrator must either create and apply additional roles for the appropriate privileges, or grant them directly to the application developers who need them.

How the CONNECT Role Change Affects Client Server Applications

Most client/server applications that use dedicated user accounts will not be affected by this change. However, applications that create private synonyms or temporary tables using dynamic SQL in the user schema during account provisioning or run-time operations will be affected. They will require additional roles or grants to acquire the system privileges appropriate to their activities.

Approaches to Addressing the CONNECT Role Change

Oracle recommends the following three approaches to address the impact of this change.

Approach 1: Create a New Database Role

The privileges removed from the CONNECT role can be managed by creating a new database role.

First, connect to the upgraded Oracle database and create a new database role. The following example uses a role called my_app_developer.

```
SQL> CREATE ROLE my_app_developer;
SQL> GRANT CREATE TABLE, CREATE VIEW, CREATE SEQUENCE, CREATE SYNONYM, CREATE
CLUSTER, CREATE DATABASE LINK, ALTER SESSION TO my_app_developer;
SQL>
```

Second, determine which users or database roles have the CONNECT role, and grant the new role to these users or roles.

```
SQL> SELECT user$.name, admin_option, default_role
     FROM user$, sysauth$, dba_role_privs
     WHERE privilege# =
     (SELECT user# from user$ WHERE name = 'CONNECT')
     AND user$.user# = grantee#
     AND grantee = user$.name
     AND granted_role = 'CONNECT';

NAME                           ADMIN_OPTI DEF
------------------------------ ---------- ---
R1                             YES        YES
R2                             NO         YES

SQL> GRANT my_app_developer TO R1 WITH ADMIN OPTION;
SQL> GRANT my_app_developer TO R2;
```

You can determine the privileges that users require by using Oracle Auditing. The audit information can then be analyzed and used to create additional database roles with finer granularity.

Privileges not used can then be revoked for specific users. Note that before auditing, the database initialization parameter AUDIT_TRAIL must be initialized and the database restarted.

```
SQL> AUDIT CREATE TABLE, CREATE SEQUENCE, CREATE SYNONYM, CREATE DATABASE LINK,
CREATE CLUSTER, CREATE VIEW, ALTER SESSION;
```

Database privilege usage can now be monitored periodically.

```
SQL> SELECT userid, name FROM aud$, system_privilege_map
WHERE - priv$used = privilege;
USERID                         NAME
------------------------------ ----------------
ACME                           CREATE TABLE
ACME                           CREATE SEQUENCE
ACME                           CREATE TABLE
ACME                           ALTER SESSION
APPS                           CREATE TABLE
APPS                           CREATE TABLE
APPS                           CREATE TABLE
APPS                           CREATE TABLE

8 rows selected.
```

Approach 2: Restore CONNECT Privileges

Starting with Oracle Database 11*g* Release 1 (11.1), Oracle provides a script called rstrconn.sql in the $ORACLE_HOME/rdbms/admin directory. After a database upgrade or new database creation, this script can be used to grant the privileges that were removed from the CONNECT role in Oracle Database 11*g* Release 1 (11.1).

If this approach is used, then privileges that are not used should be revoked from users who do not need them. To identify such privileges and users, the database must be restarted with the database initialization parameter AUDIT_TRAIL initialized, for example, AUDIT_TRAIL=DB. Oracle Database auditing should then be turned on to monitor what privileges are used, as follows:

```
SQL> AUDIT CREATE TABLE, CREATE SEQUENCE, CREATE SYNONYM, CREATE DATABASE LINK,
```

```
CREATE CLUSTER, CREATE VIEW, ALTER SESSION;
```

Database privilege usage can also be monitored periodically.

```
SQL> SELECT userid, name FROM aud$, system_privilege_map
WHERE - priv$used = privilege;
USERID                         NAME
------------------------------ ----------------
ACME                           CREATE TABLE
ACME                           CREATE SEQUENCE
ACME                           CREATE TABLE
ACME                           ALTER SESSION
APPS                           CREATE TABLE
APPS                           CREATE TABLE
APPS                           CREATE TABLE
APPS                           CREATE TABLE
8 rows selected.
SQL>
```

New View Showing CONNECT Grantees A new view enables administrators who continue using the old CONNECT role to see quickly which users have that role.

Table 10–1 shows the columns in the new DBA_CONNECT_ROLE_GRANTEES view.

Table 10–1 Columns and Contents for DBA_CONNECT_ROLE_GRANTEES

Column Name	Contents
Grantee	User granted the CONNECT role
Path_of_connect_role_grant	Role (or nested roles) by which the user is granted CONNECT
Admin_opt	VARCHAR2(3), YES if user has ADMIN OPTION on CONNECT; otherwise, NO

Approach 3: Conduct Least Privilege Analysis

Oracle partners and application providers should use this approach to deliver more secure products to the Oracle customer base. The principle of least privilege mitigates risk by limiting privileges to the minimum set required to perform a given function.

For each class of users that the analysis shows need the same set of privileges, create a role with only those privileges. Remove all other privileges from those users, and assign that role to those users. As needs change, you can grant additional privileges, either directly or through these new roles, or create new roles to meet new needs. This approach helps to ensure that inappropriate privileges have been limited, thereby reducing the risk of inadvertent or malicious harm.

Glossary

application context

A name-value pair that enables an application to access session information about a user, such as the user ID or other user-specific information, and then securely pass this data to the database.

See also **global application context**.

application role

A database role that is granted to application users and that is secured by embedding passwords inside the application.

See also **secure application role**.

certificate

An ITU x.509 v3 standard data structure that securely binds an identify to a public key.

A certificate is created when an entity's public key is signed by a trusted identity, a certificate authority. The certificate ensures that the entity's information is correct, and that the public key belongs to that entity.

A certificate contains the entity's name, identifying information, and public key. It is also likely to contain a serial number, expiration date, and information about the rights, uses, and privileges associated with the certificate. Finally, it contains information about the certificate authority that issued it.

certificate revocation list (CRL)

See **CRL**.

CRL

A set of signed data structures that contain a list of revoked **certificates**. The authenticity and integrity of the CRL is provided by a digital signature appended to it. Usually, the CRL signer is the same entity that signed the issued certificate.

definer's rights procedure

A procedure that executes with the privileges of its owner, not its current user. Definer's rights subprograms are bound to the schema in which they are located. 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.

decryption

Decoding an encyrpted message so that it is readable.

encryption

Disguising a message, rendering it unreadable to all but the intended recipient.

Forwardable Ticket Granting Ticket

A special Kerberos ticket that can be forwarded to proxies, permitting the proxy to obtain additional Kerberos tickets on behalf of the client for proxy authentication.

See also **Kerberos ticket**.

global application context

A name-value pair that enables application context values to be accessible across database sessions.

See also **application context**.

integrity

A guarantee that the contents of a message received were not altered from the contents of the original message sent.

invoker's rights procedures

Procedures that 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

A computer that issues Kerberos tickets.

See also **Kerberos ticket**.

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)

See **KDC**.

lightweight user session

A user session that contains only information pertinent to the application that the user is logging onto. The lightweight user session does not hold its own database resources, such as transactions and cursors; hence it is considered "lightweight." Lightweight user sessions consume far less system resources than traditional database session. Because lightweight user sessions consume much fewer server resources, a lightweight user session can be dedicated to each end user and can persist for as long as the application deems necessary.

mandatory auditing

Activities that are audited by default, regardless of whether or not auditing was enabled. These activities include connections to the instance with administrator privileges, database startups, and database shutdowns. Oracle Database writes these activities to the operating system audit file.

namespace

In Oracle Database security, the name of an application context. You create this name in a `CREATE CONTEXT` statement.

Oracle Virtual Private Database

A set of features that enables you to create security policies to control database access at the row and column level. Essentially, Oracle Virtual Private Database adds a dynamic `WHERE` clause to a SQL statement that is issued against the table, view, or synonym to which an Oracle Virtual Private Database security policy was applied.

salt

In cryptography, a way to strengthen the security of encrypted data. Salt is a random string that is added to the data before it is encrypted, making it more difficult for attackers to steal the data by matching patterns of ciphertext to known ciphertext samples. Salt is often also added to passwords, before the passwords are encrypted, to avoid dictionary attacks, a method that unethical hackers (attackers) use to steal passwords. The encrypted salted values make it difficult for attackers to match the hash value of encrypted passwords (sometimes called verifiers) with their dictionary lists of common password hash values.

secure application role

A database role that is granted to application users, but 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 role**.

service ticket

See **Kerberos ticket**.

wallet

A data structure used to store and manage security credentials for an individual entity.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X