Oracle® Database

Advanced Security Guide

19c

E96301-06

September 2020

Oracle Database Advanced Security Guide, 19c

E96301-06

Primary Author: Patricia Huey

Contributors: Sudha Duraiswamy , Michael Hwa, Sudha Iyer, Supriya Kalyanasundaram, Lakshmi Kethana, Peter Knaggs, Andrew Koyfman, Dah-Yoh Lim, Adam Lee, Adam Lindsey, Rahil Mir, Gopal Mulagund, Andy Philips, Preetam Ramakrishna, Saikat Saha, Philip Thornton, Peter Wahl, Lixia Yuan, Paul Youn

Contents

3 Configuring Transparent Data Encryption

4 Managing the Keystore and the Master Encryption Key

5 Managing Keystores and TDE Master Encryption Keys in United Mode

6 Managing Keystores and TDE Master Encryption Keys in Isolated Mode

7 General Considerations of Using Transparent Data Encryption

8 Using Transparent Data Encryption with Other Oracle Features

13 Configuring Oracle Data Redaction Policies

14 Managing Oracle Data Redaction Policies in Oracle Enterprise Manager

15 Using Oracle Data Redaction with Oracle Database Features

16 Security Considerations for Oracle Data Redaction

Glossary

Index

Preface

Welcome to *Oracle Database Advanced Security Guide* for the 12*c* Release 2 (12.2) of Oracle Advanced Security. This guide describes how to implement, configure, and administer Oracle Advanced Security.

Audience

Oracle Database Advanced Security Guide is intended for users and systems professionals involved with the implementation, configuration, and administration of Oracle Advanced Security including:

- Implementation consultants
- System administrators
- Security administrators
- Database administrators (DBAs)

Documentation Accessibility

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

Access to Oracle Support

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

Related Documents

Before you configure Oracle Advanced Security features, you should be familiar with the following guides:

- *Oracle Database Administrator's Guide*
- *Oracle Database Security Guide*

- *Oracle Database SQL Language Reference*
- *Oracle Database PL/SQL Packages and Types Reference*
- *Oracle Multitenant Administrator's Guide*

Many books in the documentation set use the sample schemas of the default database. Refer to *Oracle Database Sample Schemas*for information about how these schemas were created and how you can use them.

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

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

If you already have a user name and password for OTN, then you can go directly to the documentation section of the OTN website at

`http://www.oracle.com/technetwork/documentation/index.html`

Conventions

The following text conventions are used in this document:

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

Changes in This Release for Oracle Database Advanced Security Guide

This preface contains:

Changes in Oracle Database Advanced Security 19c

The following are changes in *Oracle Database Advanced Security Guide* for Oracle Database 19c.

- Improved Key Management Support for Encrypting Oracle-Managed Tablespaces
  In this release, closing a TDE keystore is now allowed even when the Oracle-managed tablespaces (`SYSTEM`, `SYSAUX`, `TEMP`, and `UNDO` tablespaces) are encrypted.
- Transparent Online Conversion Support for Auto-Renaming in Non-Oracle-Managed Files Mode
  Starting with this release, in a Transparent Data Encryption online conversion in non-Oracle-managed files mode, you are no longer forced to include the `FILE_NAME_CONVERT` clause in the `ADMINISTER KEY MANAGEMENT` SQL statement. The file name will retain its original name.
- Support for More Algorithms for Offline Tablespace Encryption
  In previous releases, only the `AES128` encryption algorithm was supported for offline tablespace encryption. In addition to `AES128`, this release introduces support for the `AES192` and `AES256` encryption algorithms, as well as `ARIA`, `GOST`, and `3DES` encryption algorithms for offline tablespace encryption.

Improved Key Management Support for Encrypting Oracle-Managed Tablespaces

In this release, closing a TDE keystore is now allowed even when the Oracle-managed tablespaces (`SYSTEM`, `SYSAUX`, `TEMP`, and `UNDO` tablespaces) are encrypted.

Internal operations on these tablespaces when they are encrypted continue to be unaffected even when the TDE keystore is in the `CLOSED` state.

Closing the TDE keystore has no effect on queries of an encrypted `SYSTEM`, `SYSAUX`, `TEMP`, or `UNDO` tablespace, unlike queries of a user-created tablespace, which continue to return an `ORA-28365 wallet is not open` error when the TDE keystore is closed.

User-initiated operations such as decrypt on any encrypted Oracle-managed tablespace still require the TDE keystore to be in the `OPEN` state.

Related Topics

- Impact of a Closed TDE Keystore on Encrypted Tablespaces
  A TDE keystore can be closed or migrated when an Oracle-managed tablespace is encrypted, and the database system itself must be shut down to disallow operations on an Oracle-managed tablespace.

Transparent Online Conversion Support for Auto-Renaming in Non-Oracle-Managed Files Mode

Starting with this release, in a Transparent Data Encryption online conversion in non-Oracle-managed files mode, you are no longer forced to include the `FILE_NAME_CONVERT` clause in the `ADMINISTER KEY MANAGEMENT` SQL statement. The file name will retain its original name.

This enhancement helps to prevent you from having to rename files back to the original name afterward, sometimes missing files.

Related Topics

- Encrypting an Existing Tablespace with Online Conversion
  To encrypt an existing tablespace with online conversion, use `ALTER TABLESPACE` with the `ONLINE` and `ENCRYPT` clauses.

Support for More Algorithms for Offline Tablespace Encryption

In previous releases, only the `AES128` encryption algorithm was supported for offline tablespace encryption. In addition to `AES128`, this release introduces support for the `AES192` and `AES256` encryption algorithms, as well as `ARIA`, `GOST`, and `3DES` encryption algorithms for offline tablespace encryption.

This enhancement benefits scenarios in which you have concerns about auxiliary space usage required by online tablespace encryption.

Related Topics

- About Encryption Conversion for Tablespaces and Databases
  The `CREATE TABLESPACE` SQL statement can be used to encrypt new tablespaces. `ALTER TABLESPACE` can encrypt existing tablespaces.
- Supported Encryption and Integrity Algorithms
  The supported Advanced Encryption Standard cipher keys, including tablespace and database encryption keys, can be either 128, 192, or 256 bits long. Tablespace and database encryption use the 128–bit length cipher key.

Changes in Oracle Database Advanced Security 18c

The following are changes in *Oracle Database Advanced Security Guide* for Oracle Database 18c.

- Ability to Create a Keystore for Each Pluggable Database
  Each pluggable database (PDB) can now have its own keystore instead of there only being one keystore for the entire container database (CDB).

- Ability to Create a User-Defined Master Encryption Key
  This release introduces the ability to create a user-defined master encryption key, also known as “bring your own key.”
- Ability to Use Encrypted Passwords for Database Links with Oracle Data Pump
  The behavior for handling database link passwords has changed in this release.

Ability to Create a Keystore for Each Pluggable Database

Each pluggable database (PDB) can now have its own keystore instead of there only being one keystore for the entire container database (CDB).

In previous releases, PDBs shared the keystore with the entire container database (CDB), which included the CDB root and all the PDBs. This keystore stored the master encryption keys for the CDB as well as all the PDBs. In this release, a PDB can either continue to share the same keystore with the entire CDB as before, or have a separate keystore. This design offers greater isolation between PDBs, because each separate keystore can be administered independently. For example, each keystore can be protected by a different password

> **Note:**
>
> This feature can only be used in an Oracle Cloud environment. It does not apply to on-premise environments.

The additional advantage of this feature is that it enables independent key management operations to be performed by each tenant (PDB) in a multitenant environment rather than having to share a keystore at the CDB root level. This feature benefits both multitenant and non-multitenant environments because it provides parameters to facilitate the configuration of the keystore location and the keystore type, eliminating the need for editing the `sqlnet.ora` file.

This feature provides the following new functionality:

- For multitenant environments, the following two modes:
  - United mode, in which the keystores and master encryption keys are primarily managed from the CDB root, and can be accessed from the united mode PDB. Within the PDB, the keystore can be opened and closed just for that PDB. You also can create a PDB-specific master encryption key for this keystore.
  - Isolated mode, in which the keystore and encryption keys are managed in an individual PDB. This way, each PDB can configure its own keystore type independently, and create and manage this keystore after configuring it.

  You can mix these two modes. For example, suppose you have 1 CDB and 10 PDBs. You can run 3 of these PDBs in united mode and the remaining 7 in isolated mode. This design offers the highest flexibility depending on your environment and site requirements.

  To accommodate these modes, the `ADMINISTER KEY MANAGEMENT` SQL statement has been enhanced to behave differently in the two modes.
- For both non-multitenant and multitenant environments, the following new features:

  - Addition of the `WALLET_ROOT` static instance initialization parameter, to specify the keystore path. In this guide, `WALLET_ROOT` refers to the configuration of software keystores, hardware keystores, and Oracle Key Vault keystores, but this parameter can be used to designate the wallet location for other products as well: Enterprise User Security, Secure Sockets Layer, Oracle XML DB, and Secure External Password Store.
  - Addition of the `TDE_CONFIGURATION` dynamic instance initialization parameter, to specify the type of keystore to use. You can set this parameter for TDE software keystores, hardware security module keystores (HSMs), and Oracle Key Vault.
  - Modification to the behavior of the `SQLNET.ENCRYPTION_WALLET_LOCATION` parameter. When the `WALLET_ROOT` parameter has been set, then `WALLET_ROOT` overrides `SQLNET.ENCRYPTION_WALLET_LOCATION`. If `WALLET_ROOT` has not been set, then `SQLNET.ENCRYPTION_WALLET_LOCATION` is the default.

This design offers greater isolation between PDBs. The additional advantage of this feature is that it enables independent key management operations to be performed by each tenant (PDB) in a multitenant environment rather than having to share a keystore at the CDB root level. This feature benefits both multitenant and non-multitenant environments because it provides parameters to facilitate the configuration of the keystore location and the keystore type, eliminating the need for editing the `sqlnet.ora` file.

Related Topics

- How the Multitenant Option Affects Transparent Data Encryption
  In a multitenant environment, you can configure keystores for either the entire container database (CDB) or for individual pluggable databases (PDBs).

Ability to Create a User-Defined Master Encryption Key

This release introduces the ability to create a user-defined master encryption key, also known as “bring your own key.”

Instead of requiring that TDE master encryption keys always be generated in the database, Oracle Database now supports the use of master encryption keys that have been generated outside the database.

To create the user-defined key, you supply your own master key identification value when you create the master encryption key by using the `ADMINISTER KEY MANAGEMENT SET [ENCRYPTION] KEY` or `ADMINISTER KEY MANAGEMENT CREATE [ENCRYPTION] KEY` statements. This enhancement applies to master encryption keys that are used in software keystores only, not hardware keystores. It can be used in non-multitenant, standalone environments and in multitenant environments.

Related Topics

- Creating a User-Defined TDE Master Encryption Key
  To create a user-defined TDE master encryption key, use the `ADMINISTER KEY MANAGEMENT` statement with the `SET | CREATE [ENCRYPTION] KEY` clause.
- Creating a User-Defined TDE Master Encryption Key in United Mode
  To create a user-defined TDE master encryption key, use the `ADMINISTER KEY MANAGEMENT` statement with the `SET | CREATE [ENCRYPTION] KEY` clause.

- Creating a User-Defined TDE Master Encryption Key in Isolated Mode
  To create a user-defined TDE master encryption key, use the `ADMINISTER KEY MANAGEMENT` statement with the `SET | CREATE [ENCRYPTION] KEY` clause.

Ability to Use Encrypted Passwords for Database Links with Oracle Data Pump

The behavior for handling database link passwords has changed in this release.

Passwords in database links are now encrypted. Oracle Data Pump handles the export and import of these passwords. Import operations from older versions and export operations to older versions can still be used.

The benefit of this feature is that it prevents an intruder from decrypting an encrypted database link password.

Related Topics

- Using Oracle Data Pump with Encrypted Data Dictionary Data
  Oracle Data Pump operations provide protections for encrypted passwords and other encrypted data.

1

Introduction to Oracle Advanced Security

Two features comprise Oracle Advanced Security: Transparent Data Encryption and Oracle Data Redaction.

- Transparent Data Encryption
  Transparent Data Encryption (TDE) enables you to encrypt data so that only an authorized recipient can read it.
- Oracle Data Redaction
  Oracle Data Redaction enables you to redact (mask) column data using several redaction types.

Transparent Data Encryption

Transparent Data Encryption (TDE) enables you to encrypt data so that only an authorized recipient can read it.

Use encryption to protect sensitive data in a potentially unprotected environment, such as data you placed on backup media that is sent to an off-site storage location. You can encrypt individual columns in a database table, or you can encrypt an entire tablespace.

To use Transparent Data Encryption, you do not need to modify your applications. TDE enables your applications to continue working seamlessly as before. It automatically encrypts data when it is written to disk, and then automatically decrypts the data when your applications access it. Key management is built-in, eliminating the complex task of managing and securing encryption keys.

Oracle Data Redaction

Oracle Data Redaction enables you to redact (mask) column data using several redaction types.

The types of redaction that you can perform are as follows:

- **Full redaction.** You redact all of the contents of the column data. The redacted value that is returned to the querying user depends on the data type of the column. For example, columns of the `NUMBER` data type are redacted with a zero (`0`) and character data types are redacted with a blank space.
- **Partial redaction.** You redact a portion of the column data. For example, you can redact most of a Social Security number with asterisks (*), except for the last 4 digits.
- **Regular expressions.** You can use regular expressions in both full and partial redaction. This enables you to redact data based on a search pattern for the data. For example, you can use regular expressions to redact specific phone numbers or email addresses in your data.

- **Random redaction.** The redacted data presented to the querying user appears as randomly generated values each time it is displayed, depending on the data type of the column.
- **No redaction.** This option enables you to test the internal operation of your redaction policies, with no effect on the results of queries against tables with policies defined on them. You can use this option to test the redaction policy definitions before applying them to a production environment.

Data Redaction performs the redaction at runtime, that is, the moment that the user tries to view the data. This functionality is ideally suited for dynamic production systems in which data constantly changes. While the data is being redacted, Oracle Database is able to process all of the data normally and to preserve the back-end referential integrity constraints. Data redaction can help you to comply with industry regulations such as Payment Card Industry Data Security Standard (PCI DSS) and the Sarbanes-Oxley Act.

Part I
Using Transparent Data Encryption

Part I describes how to use Transparent Data Encryption.

- Introduction to Transparent Data Encryption
  Transparent Data Encryption enables you to encrypt sensitive data, such as credit card numbers or Social Security numbers.
- Configuring Transparent Data Encryption
  You can configure software or hardware keystores, for use on both individual table columns or entire tablespaces.
- Managing the Keystore and the Master Encryption Key
  You can modify settings for the keystore and TDE master encryption key, and store Oracle Database and store Oracle GoldenGate secrets in a keystore.
- Managing Keystores and TDE Master Encryption Keys in United Mode
  United mode enables you to create a common keystore for the CDB and the PDBs for which the keystore is in united mode.
- Managing Keystores and TDE Master Encryption Keys in Isolated Mode
  In an Oracle Cloud database (but not an on-premises database), isolated mode enables you to create a keystore for each pluggable database (PDB).
- General Considerations of Using Transparent Data Encryption
  When you use Transparent Data Encryption, you should consider factors such as security, performance, and storage overheads.
- Using Transparent Data Encryption with Other Oracle Features
  You can use Oracle Data Encryption with other Oracle features, such as Oracle Data Guard or Oracle Real Application Clusters.
- Using sqlnet.ora to Configure Transparent Data Encryption Keystores
  If you do not want to use the `WALLET_ROOT` and `TDE_CONFIGURATION` parameters to configure the keystore location, then you can use the `sqlnet.ora` file.
- Frequently Asked Questions About Transparent Data Encryption
  Users frequently have questions about transparency and performance issues with Transparent Data Encryption.

2

Introduction to Transparent Data Encryption

Transparent Data Encryption enables you to encrypt sensitive data, such as credit card numbers or Social Security numbers.

- What Is Transparent Data Encryption?
  Transparent Data Encryption (TDE) enables you to encrypt sensitive data that you store in tables and tablespaces.
- Benefits of Using Transparent Data Encryption
  Transparent Data Encryption (TDE) ensures that sensitive data is encrypted, meets compliance requirements, and provides functionality that streamlines encryption operations.
- Who Can Configure Transparent Data Encryption?
  You must be granted the `ADMINISTER KEY MANAGEMENT` system privilege to configure Transparent Data Encryption (TDE).
- Types and Components of Transparent Data Encryption
  Transparent Data Encryption can be applied to individual columns or entire tablespaces.
- How the Multitenant Option Affects Transparent Data Encryption
  In a multitenant environment, you can configure keystores for either the entire container database (CDB) or for individual pluggable databases (PDBs).

What Is Transparent Data Encryption?

Transparent Data Encryption (TDE) enables you to encrypt sensitive data that you store in tables and tablespaces.

After the data is encrypted, this data is transparently decrypted for authorized users or applications when they access this data. TDE helps protect data stored on media (also called data at rest) in the event that the storage media or data file is stolen.

Oracle Database uses authentication, authorization, and auditing mechanisms to secure data in the database, but not in the operating system data files where data is stored. To protect these data files, Oracle Database provides Transparent Data Encryption (TDE). TDE encrypts sensitive data stored in data files. To prevent unauthorized decryption, TDE stores the encryption keys in a security module external to the database, called a keystore.

You can configure Oracle Key Vault as part of the TDE implementation. This enables you to centrally manage TDE keystores (called TDE wallets in Oracle Key Vault) in your enterprise. For example, you can upload a software keystore to Oracle Key Vault and then make the contents of this keystore available to other TDE-enabled databases.

Related Topics

- *Oracle Key Vault Administrator's Guide*

Benefits of Using Transparent Data Encryption

Transparent Data Encryption (TDE) ensures that sensitive data is encrypted, meets compliance requirements, and provides functionality that streamlines encryption operations.

Benefits are as follows:

- As a security administrator, you can be sure that sensitive data is encrypted and therefore safe in the event that the storage media or data file is stolen.
- Using TDE helps you address security-related regulatory compliance issues.
- You do not need to create auxiliary tables, triggers, or views to decrypt data for the authorized user or application. Data from tables is transparently decrypted for the database user and application. An application that processes sensitive data can use TDE to provide strong data encryption with little or no change to the application.
- Data is transparently decrypted for database users and applications that access this data. Database users and applications do not need to be aware that the data they are accessing is stored in encrypted form.
- You can encrypt data with zero downtime on production systems by using online table redefinition or you can encrypt it offline during maintenance periods. (See *Oracle Database Administrator's Guide* for more information about online table redefinition.)
- You do not need to modify your applications to handle the encrypted data. The database manages the data encryption and decryption.
- Oracle Database automates TDE master encryption key and keystore management operations. The user or application does not need to manage TDE master encryption keys.

Who Can Configure Transparent Data Encryption?

You must be granted the `ADMINISTER KEY MANAGEMENT` system privilege to configure Transparent Data Encryption (TDE).

If you must open the keystore at the mount stage, then you must be granted the `SYSKM` administrative privilege, which includes the `ADMINISTER KEY MANAGEMENT` system privilege and other necessary privileges.

When you grant the `SYSKM` administrative privilege to a user, ensure that you create a password file for it so that the user can connect to the database as `SYSKM` using a password. This enables the user to perform actions such as querying the `V$DATABASE` view.

To configure TDE column or tablespace encryption, you do not need the `SYSKM` or `ADMINISTER KEY MANAGEMENT` privileges. You must have the following additional privileges to encrypt table columns and tablespaces:

- `CREATE TABLE`
- `ALTER TABLE`
- `CREATE TABLESPACE`

- `ALTER TABLESPACE` (for online and offline tablespace encryption)
- `ALTER DATABASE` (for fast offline tablespace encryption)

Types and Components of Transparent Data Encryption

Transparent Data Encryption can be applied to individual columns or entire tablespaces.

About Transparent Data Encryption Types and Components

You can encrypt sensitive data at the column level or the tablespace level.

At the column level, you can encrypt data using selected table columns. TDE tablespace encryption enables you to encrypt all of the data that is stored in a tablespace.

Both TDE column encryption and TDE tablespace encryption use a two-tiered key-based architecture. Unauthorized users, such as intruders who are attempting security attacks, cannot read the data from storage and back up media unless they have the TDE master encryption key to decrypt it.

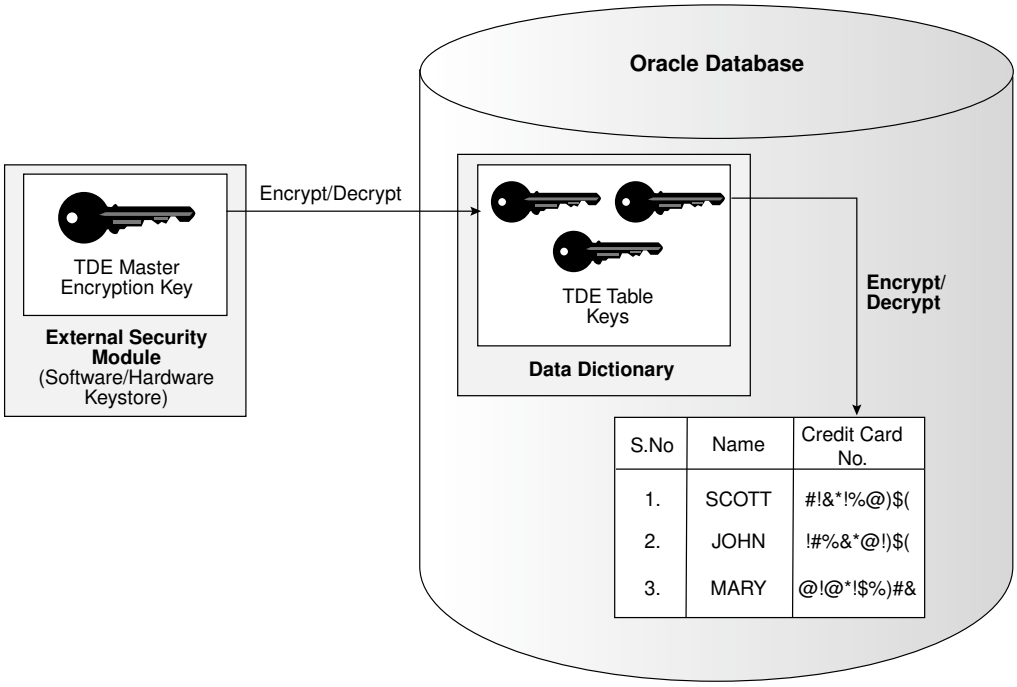
How Transparent Data Encryption Column Encryption Works

Transparent Data Encryption (TDE) column encryption protects confidential data, such as credit card and Social Security numbers, that is stored in table columns.

TDE column encryption uses the two-tiered key-based architecture to transparently encrypt and decrypt sensitive table columns. The TDE master encryption key is stored in an external security module, which can be an Oracle software keystore or hardware keystore. This TDE master encryption key encrypts and decrypts the TDE table key, which in turn encrypts and decrypts data in the table column.

Figure 2-1 an overview of the TDE column encryption process.

Figure 2-1 TDE Column Encryption Overview



As shown in Figure 2-1, the TDE master encryption key is stored in an external security module that is outside of the database and accessible only to a user who was granted the appropriate privileges. For this external security module, Oracle Database uses an Oracle software keystore (wallet, in previous releases) or hardware security module (HSM) keystore. Storing the TDE master encryption key in this way prevents its unauthorized use.

Using an external security module separates ordinary program functions from encryption operations, making it possible to assign separate, distinct duties to database administrators and security administrators. Security is enhanced because the keystore password can be unknown to the database administrator, requiring the security administrator to provide the password.

When a table contains encrypted columns, TDE uses a single TDE table key regardless of the number of encrypted columns. Each TDE table key is individually encrypted with the TDE master encryption key. All of the TDE table keys are located together in the `colklc` column of the `ENC$` data dictionary table. No keys are stored in plaintext.

How Transparent Data Encryption Tablespace Encryption Works

Transparent Data Encryption (TDE) tablespace encryption enables you to encrypt an entire tablespace.

All of the objects that are created in the encrypted tablespace are automatically encrypted. TDE tablespace encryption is useful if your tables contain sensitive data in multiple columns, or if you want to protect the entire table and not just individual columns. You do not need to perform a granular analysis of each table column to determine the columns that need encryption.

In addition, TDE tablespace encryption takes advantage of bulk encryption and caching to provide enhanced performance. The actual performance impact on applications can vary.

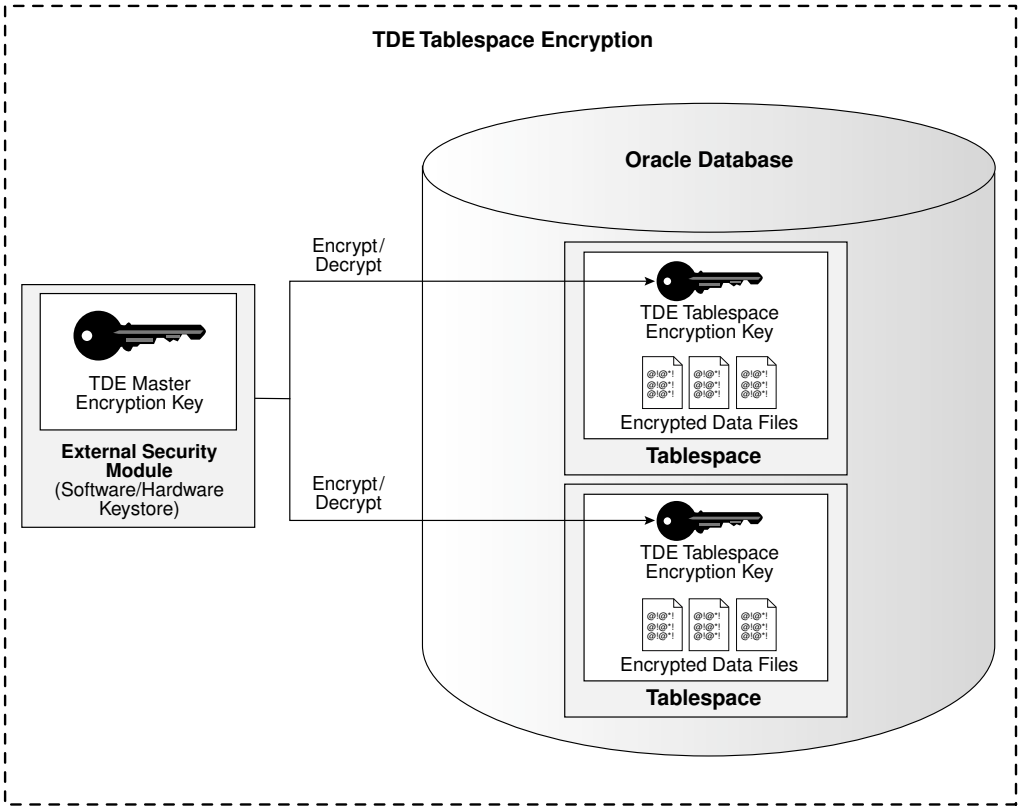
TDE tablespace encryption encrypts all of the data stored in an encrypted tablespace including its redo data. TDE tablespace encryption does not encrypt data that is stored outside of the tablespace. For example, `BFILE` data is not encrypted because it is stored outside the database. If you create a table with a `BFILE` column in an encrypted tablespace, then this particular column will not be encrypted.

All of the data in an encrypted tablespace is stored in encrypted format on the disk. Data is transparently decrypted for an authorized user having the necessary privileges to view or modify the data. A database user or application does not need to know if the data in a particular table is encrypted on the disk. In the event that the data files on a disk or backup media is stolen, the data is not compromised.

TDE tablespace encryption uses the two-tiered, key-based architecture to transparently encrypt (and decrypt) tablespaces. The TDE master encryption key is stored in an external security module (software or hardware keystore). This TDE master encryption key is used to encrypt the TDE tablespace encryption key, which in turn is used to encrypt and decrypt data in the tablespace.

Figure 2-2 shows an overview of the TDE tablespace encryption process.

Figure 2-2 TDE Tablespace Encryption



> **Note:**
>
> The encrypted data is protected during operations such as `JOIN` and `SORT`. This means that the data is safe when it is moved to temporary tablespaces. Data in undo and redo logs is also protected.

TDE tablespace encryption also allows index range scans on data in encrypted tablespaces. This is not possible with TDE column encryption.

Oracle Database implements the following features to TDE tablespace encryption:

- It uses a unified TDE master encryption key for both TDE column encryption and TDE tablespace encryption.
- You can reset the unified TDE master encryption key. This provides enhanced security and helps meet security and compliance requirements.

How the Keystore for the Storage of TDE Master Encryption Keys Works

To control the encryption, you use a keystore and a TDE master encryption key.

- About the Keystore Storage of TDE Master Encryption Keys
  Oracle Database provides a key management framework for Transparent Data Encryption that stores and manages keys and credentials.
- Benefits of the Keystore Storage Framework
  The key management framework provides several benefits for Transparent Data Encryption.
- Types of Keystores
  Oracle Database supports software keystores, Oracle Key Vault, and other PKCS#11 compatible key management devices.

About the Keystore Storage of TDE Master Encryption Keys

Oracle Database provides a key management framework for Transparent Data Encryption that stores and manages keys and credentials.

The key management framework includes the keystore to securely store the TDE master encryption keys and the management framework to securely and efficiently manage keystore and key operations for various database components.

The Oracle keystore stores a history of retired TDE master encryption keys, which enables you to change them and still be able to decrypt data that was encrypted under an earlier TDE master encryption key.

Benefits of the Keystore Storage Framework

The key management framework provides several benefits for Transparent Data Encryption.

- Enables separation of duty between the database administrator and the security administrator who manages the keys. You can grant the `ADMINISTER KEY`

`MANAGEMENT` or `SYSKM` privilege to users who are responsible for managing the keystore and key operations.

- Facilitates compliance, because it helps you to track encryption keys and implement requirements such as keystore password rotation and TDE master encryption key reset or rekey operations.
- Facilitates and helps enforce keystore backup requirements. A backup is a copy of the password-protected software keystore that is created for all of the critical keystore operations.

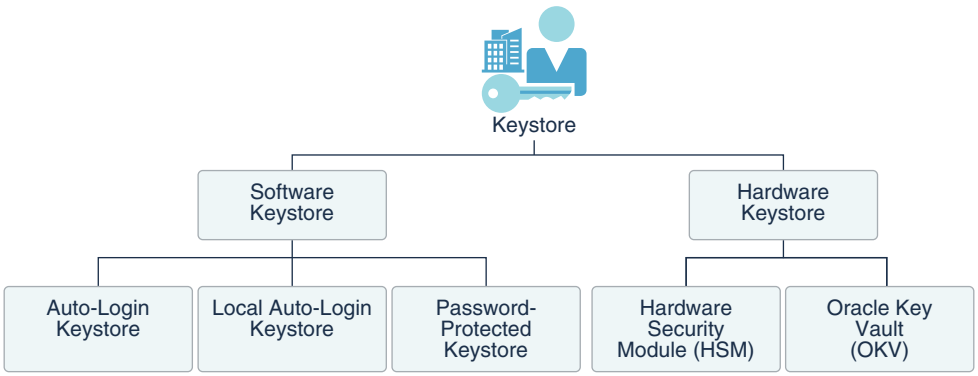
  You must make a backup of the keystore for all of the critical keystore operations. You must also make a backup of the TDE master encryption key before you reset or rekey this TDE master encryption key.
- Enables the keystore to be stored on an ASM file system. This is particularly useful for Oracle Real Application Clusters (Oracle RAC) environments where database instances share a unified file system view.
- Enables reverse migration from a hardware keystore to a file system-based software keystore. This option is useful if you must migrate back to a software keystore.

Types of Keystores

Oracle Database supports software keystores, Oracle Key Vault, and other PKCS#11 compatible key management devices.

Figure 2-3 illustrates the types of keystores that Oracle Database supports.

Figure 2-3 Oracle Database Supported Keystores



These keystores are as follows:

- **Auto-login software keystores:** Auto-login software keystores are protected by a system-generated password, and do not need to be explicitly opened by a security administrator. Auto-login software keystores are automatically opened when accessed. Auto-login software keystores can be used across different systems. If your environment does not require the extra security provided by

a keystore that must be explicitly opened for use, then you can use an auto-login software keystore. Auto-login software keystores are ideal for unattended scenarios.

- **Local auto-login software keystores:** Local auto-login software keystores are auto-login software keystores that are local to the computer on which they are created. Local auto-login keystores cannot be opened on any computer other than the one on which they are created. This type of keystore is typically used for scenarios where additional security is required (that is, to limit the use of the auto-login for that computer) while supporting an unattended operation.
- **Password-protected software keystores:** Password-protected software keystores are protected by using a password that you create. You must open this type of keystore before the keys can be retrieved or used.

Software keystores can be stored in Oracle Automatic Storage Management (Oracle ASM), Oracle Automatic Storage Management Cluster File System (Oracle ACFS), or regular file systems.

Hardware keystores are in the following categories:

- **Hardware security modules (HSMs):** Hardware security modules are third-party physical devices that provide secure storage for encryption keys, in hardware keystores. HSMs also provide secure computational space (memory) to perform encryption and decryption operations. If you are using an HSM, data encryption keys must decrypted by the TDE master encryption key inside the FIPS-boundary of the HSM. This means that the TDE master encryption key never leaves the internal FIPS boundary of the HSM, which explains the high (network) load when a database with high transactional load (for example, Exadata with hundreds of encrypted PDBs) continues to send data encryption keys to the HSM.
- **Oracle Key Vault (OKV) keystores:** Oracle Key Vault keystores (wallets) are Oracle supplied hardware keystores. Oracle Key Vault enables you to centrally manage encryption keys, Oracle keystores, Java kesytores, and credential files.

Related Topics

- *Oracle Key Vault Administrator's Guide*

Supported Encryption and Integrity Algorithms

The supported Advanced Encryption Standard cipher keys, including tablespace and database encryption keys, can be either 128, 192, or 256 bits long. Tablespace and database encryption use the 128–bit length cipher key.

for TDE column encryption, salt is added by default to plaintext before encryption unless specified otherwise. You cannot add salt to indexed columns that you want to encrypt. For indexed columns, choose the `NO SALT` parameter for the SQL `ENCRYPT` clause.

For TDE tablespace encryption and database encryption, the default is to use the Advanced Encryption Standard with a 128-bit length cipher key (AES128). By default, Transparent Data Encryption (TDE) Column encryption uses the Advanced Encryption Standard (AES).

You can change encryption algorithms and encryption keys on existing encrypted columns by setting a different algorithm with the SQL `ENCRYPT` clause.

Table 2-1 lists the supported encryption algorithms.

Table 2-1 Supported Encryption Algorithms for Transparent Data Encryption

Algorithm	Key Size	Parameter Name
Advanced Encryption Standard (AES)	• 128 bits (default for tablespace encryption) • 192 bits (default for column encryption) • 256 bits	• AES192 • AES128 • AES256
ARIA	• 128 bits • 192 bits • 256 bits	• ARIA128 • ARIA192 • ARIA256
GOST	256 bits	GOST256
SEED	128 bits	SEED128
Triple Encryption Standard (DES)	168 bits	3DES168

For integrity protection of TDE column encryption, the `SHA-1` hashing algorithm is used. If you have storage restrictions, then use the `NOMAC` option.

Related Topics

•

How the Multitenant Option Affects Transparent Data Encryption

In a multitenant environment, you can configure keystores for either the entire container database (CDB) or for individual pluggable databases (PDBs).

Oracle Database supports the following multitenant modes for the management of keystores:

- United mode enables you to configure one keystore for the CDB root and any associated united mode PDBs. United mode operates much the same as how TDE was managed in an multitenant environment in previous releases.
- Isolated mode enables you to create and manage both keystores and TDE master encryption keys in an individual PDB. Different isolated mode PDBs can have different keystore types.

You can use these modes to configure software keystores, hardware keystores, and Oracle Key Vault keystores.

Depending on your site's needs, you can use a mixture of both united mode and isolated mode. For example, if you want most of the PDBs to use one type of a keystore, then you can configure the keystore type in the CDB root (united mode). For the PDBs in this CDB that must use a different type of keystore, then you can configure the PDB itself to use the keystore it needs (isolated mode). The isolated mode setting for the PDB will override the united mode setting for the CDB.

Before you can configure keystores for use in united or isolated mode, you must perform a one-time configuration by using initialization parameters. To configure

keystores for united mode and isolated mode, you use the `ADMINISTER KEY MANAGEMENT` statement. After you restart the database, where you can use the `ADMINISTER KEY MANAGEMENT` statement commands will change. For example, before the configuration, you could not use the `EXTERNAL STORE` clause in the `ADMINISTER KEY MANAGEMENT` statement in the CDB root, but after the configuration, you can.

Related Topics

- Managing Keystores and TDE Master Encryption Keys in United Mode
  United mode enables you to create a common keystore for the CDB and the PDBs for which the keystore is in united mode.
- Managing Keystores and TDE Master Encryption Keys in Isolated Mode
  In an Oracle Cloud database (but not an on-premises database), isolated mode enables you to create a keystore for each pluggable database (PDB).
- Using sqlnet.ora to Configure Transparent Data Encryption Keystores
  If you do not want to use the `WALLET_ROOT` and `TDE_CONFIGURATION` parameters to configure the keystore location, then you can use the `sqlnet.ora` file.

3
Configuring Transparent Data Encryption

You can configure software or hardware keystores, for use on both individual table columns or entire tablespaces.

- About Configuring Transparent Data Encryption
  To configure Transparent Data Encryption, you must perform a one-time setup before you create keystores and encrypt data.
- Transparent Data Encryption Keystore Search Order
  The search order for the TDE keystore depends on how you have set either the instance initialization parameters, the `sqlnet.ora` parameters, or the environment variables.
- Configuring a Software Keystore
  A software keystore is a container for the master encryption key, and it resides in the software file system.
- Configuring a Hardware Keystore
  A hardware keystore resides in a hardware security module (HSM), which is designed to store encryption keys.
- Encrypting Columns in Tables
  You can use Transparent Data Encryption to encrypt individual columns in database tables.
- Encryption Conversions for Tablespaces and Databases
  You can perform encryption operations on both offline and online tablespaces and databases.
- Transparent Data Encryption Data Dynamic and Data Dictionary Views
  You can query a set of dynamic and data dictionary views to find more information about Transparent Data Encryption (TDE) data.

About Configuring Transparent Data Encryption

To configure Transparent Data Encryption, you must perform a one-time setup before you create keystores and encrypt data.

Before you can configure keystores and begin to encrypt data, you must perform a one-time configuration using the `WALLET_ROOT` and `TDE_CONFIGURATION` parameters to designate the location and type of keystores that you plan to create.

The `WALLET_ROOT` parameter specifies the keystore directory location. Before you set `WALLET_ROOT`, ensure that you have an existing directory that you can use to store keystores. (Typically, this directory is called `wallet`.)

The `TDE_CONFIGURATION` parameter specifies the type of keystore (software keystore, hardware keystore, or Oracle Key Vault keystore). If you omit the `TDE_CONFIGURATION` parameter, then Oracle Database uses the `sqlnet.ora` file settings. After you set the type of keystore using `TDE_CONFIGURATION`, when you create the keystore, Oracle Database creates a directory within the `WALLET_ROOT` location for the keystore type. For example, if you set `TDE_CONFIGURATION` to `FILE`, for Transparent Data Encryption

keystores, then Oracle Database creates a directory named `tde` (lower case) within the wallet directory. If you want to migrate from one keystore type to another, then you must first set `TDE_CONFIGURATION` parameter to the keystore type that you want to use, and then use the `ADMINISTER KEY MANAGEMENT` statement to perform the migration. For example, you can migrate from a hardware security module (HSM) keystore to a TDE keystore.

The `KEYSTORE_MODE` column of the `V$ENCRYPTION_WALLET` dynamic view shows whether united mode or isolated mode has been configured.

> **Note:**
>
> In previous releases, the `SQLNET.ENCRYPTION_WALLET_LOCATION` parameter was used to define the keystore directory location. This parameter has been deprecated. Oracle recommends that you use the `WALLET_ROOT` static initialization parameter and `TDE_CONFIGURATION` dynamic initialization parameter instead.

Related Topics

- Transparent Data Encryption Keystore Search Order
  The search order for the TDE keystore depends on how you have set either the instance initialization parameters, the `sqlnet.ora` parameters, or the environment variables.

Transparent Data Encryption Keystore Search Order

The search order for the TDE keystore depends on how you have set either the instance initialization parameters, the `sqlnet.ora` parameters, or the environment variables.

Oracle Database retrieves the keystore by searching in these locations, in the following order:

1. The location set by the `WALLET_ROOT` instance initialization parameter, when the `KEYSTORE_CONFIGURATION` attribute of the `TDE_CONFIGURATION` initialization parameter is set to `FILE`. Oracle recommends that you use this parameter to configure the keystore location.
2. If the `KEYSTORE_CONFIGURATION` attribute of the `TDE_CONFIGURATION` initialization parameter is not set to `FILE` or `WALLET_ROOT` is not set, then the location specified by the `WALLET_LOCATION` setting in the `sqlnet.ora` file.
3. If `WALLET_ROOT` and `WALLET_LOCATION` are not set, then the location specified by the `ENCRYPTION_WALLET_LOCATION` parameter (now deprecated in favor of `WALLET_ROOT`) in the `sqlnet.ora` file.
4. If none of these parameters are set, and if the `ORACLE_BASE` environment variable is set, then the `$ORACLE_BASE/admin/`*`db_unique_name`*`/wallet` directory. If `ORACLE_BASE` is not set, then `$ORACLE_HOME/admin/`*`db_unique_name`*`/wallet`.

Configuring a Software Keystore

A software keystore is a container for the master encryption key, and it resides in the software file system.

- About Configuring a Software Keystore
  A software keystore is a container that stores the TDE master encryption key.
- Step 1: Configure the Software Keystore Location and Type
  You must configure the keystore location and type by setting `WALLET_ROOT` in `init.ora` and `TDE_CONFIGURATION` in the database instance.
- Step 2: Create the Software Keystore
  After you have specified a directory location for the software keystore, you can create the keystore.
- Step 3: Open the Software Keystore
  Depending on the type of keystore you create, you must manually open the keystore before you can use it.
- Step 4: Set the TDE Master Encryption Key in the Software Keystore
  Once the keystore is open, you can set a TDE master encryption key for it.
- Step 5: Encrypt Your Data
  Now that you have completed the keystore configuration, you can begin to encrypt data in the PDB.

About Configuring a Software Keystore

A software keystore is a container that stores the TDE master encryption key.

Before you can configure the keystore, you first must define a location for it by setting the `WALLET_ROOT` parameter in the `init.ora` file. The database locates this keystore by first checking the `WALLET_ROOT` setting. If this setting has not been created, then the database checks the `sqlnet.ora` file. You can create other keystores, such as copies of the keystore and export files that contain keys, depending on your needs. If you must remove or delete the keystore that you configured in the `WALLET_ROOT` location, then you must do so only after you have moved the TDE master encryption key in this keystore to another keystore. Then you must reset `WALLET_ROOT` to point to the new location of the keystore.

After you configure the software keystore location using the `WALLET_ROOT` parameter, you can log in to the database instance to create and open the keystore, and then set the TDE master encryption key. After you complete these steps, you can begin to encrypt data.

Step 1: Configure the Software Keystore Location and Type

You must configure the keystore location and type by setting `WALLET_ROOT` in `init.ora` and `TDE_CONFIGURATION` in the database instance.

1. Log in to the server where the Oracle database resides.
2. If necessary, create a wallet directory.

Typically, the wallet directory is located in the `$ORACLE_BASE/admin/db_unique_name` directory, and it is named `wallet`. Preferably, this directory should be empty.

3. Edit the `init.ora` initialization file for the database instance to include the `WALLET_ROOT` static initialization parameter for the wallet directory.

   By default, the `init.ora` file is located in the `$ORACLE_HOME/dbs` directory.

   For example, for a database instance named `orcl`:

   ```
   WALLET_ROOT=$ORACLE_BASE/admin/orcl/wallet
   ```

4. Log in to the database instance as a user who has been granted the `SYSDBA` administrative privilege.

   For example:

   ```
   sqlplus sec_admin as sysdba
   Enter password: password
   ```

5. Set the `TDE_CONFIGURATION` dynamic initialization parameter to specify the keystore type.

   For example:

   ```
   ALTER SYSTEM SET TDE_CONFIGURATION="KEYSTORE_CONFIGURATION=keystore_type"
   SCOPE=scope_type ;
   ```

   In this specification:

   - *keystore_type* must be set to `FILE` to configure a TDE keystore.
   - *scope_type* sets the type of scope (for example, `both`, `memory`, or `spfile`.

   For example, to configure a TDE keystore if the parameter file (`pfile`) is in use, set `scope` to `memory`:

   ```
   ALTER SYSTEM SET TDE_CONFIGURATION="KEYSTORE_CONFIGURATION=FILE"
   scope=memory;
   ```

   To configure a TDE keystore if the server parameter file (`spfile`) is in use, set `scope` to `both`:

   ```
   ALTER SYSTEM SET TDE_CONFIGURATION="KEYSTORE_CONFIGURATION=FILE" scope=both;
   ```

6. Restart the database.

   ```
   SHUTDOWN IMMEDIATE
   STARTUP
   ```

Step 2: Create the Software Keystore

After you have specified a directory location for the software keystore, you can create the keystore.

- About Creating Software Keystores
  There are three different types of software keystores.
- Creating a Password-Protected Software Keystore
  A password-protected software keystore requires a user password, which is used to protect the keys and credentials stored in the keystore.

- Creating an Auto-Login or a Local Auto-Login Software Keystore
  As an alternative to password-protected keystores, you can create either an auto-login or local auto-login software keystore.

About Creating Software Keystores

There are three different types of software keystores.

You can create password-protected software keystores, auto-login software keystores, and local auto-login software keystores.

Be aware that executing the query `SELECT * FROM V$ENCRYPTION_WALLET` will automatically open an auto-login software keystore. For example, suppose you have a password-protected keystore and an auto-login keystore. If the password-protected keystore is open and you close the password-protected keystore and then query the `V$ENCRYPTION_WALLET` view, then the output will indicate that a keystore is open. However, this is because `V$ENCRYPTION_WALLET` opened up the auto-login software keystore and then displayed the status of the auto-login keystore.

Related Topics

- Types of Keystores
  Oracle Database supports software keystores, Oracle Key Vault, and other PKCS#11 compatible key management devices.

Creating a Password-Protected Software Keystore

A password-protected software keystore requires a user password, which is used to protect the keys and credentials stored in the keystore.

1. Log in to the database instance as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.

   For example:

   ```
   sqlplus sec_admin as syskm
   Enter password: password
   Connected.
   ```

   If SQL*Plus is already open and you had modified the `init.ora` file to set the `WALLET_ROOT` parameter during this time, then reconnect to SQL*Plus. The database session must be changed before the `init.ora` changes can take effect.

2. Create the keystore by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT CREATE KEYSTORE 'keystore_location'
   IDENTIFIED BY software_keystore_password;
   ```

   In this specification:

   - *keystore_location* is the path to the keystore directory location of the password-protected keystore for which you want to create the auto-login keystore (for example, `/etc/ORACLE/WALLETS/orcl`). If the path that is set by the `WALLET_ROOT` parameter is the path that you want to use, then you can omit the *keystore_location* setting.

     If you specify the *keystore_location*, then enclose it in single quotation marks (`' '`). To find this location, you can query the `WRL_PARAMETER` column of the `V$ENCRYPTION_WALLET` view. (If the keystore was not created in the

default location, then the STATUS column of the V$ENCRYPTION_WALLET view is NOT_AVAILABLE.) If WALLET_ROOT is set and you want to use the directory that WALLET_ROOT points to, then you can omit the *keystore_location* setting.

- *software_keystore_password* is the password of the keystore that you, the security administrator, creates.

For example, to create the keystore in the /etc/ORACLE/WALLETS/orcl directory:

```
ADMINISTER KEY MANAGEMENT CREATE KEYSTORE '/etc/ORACLE/WALLETS/orcl'
IDENTIFIED BY password;

keystore altered.
```

After you run this statement, the ewallet.p12 file, which is the keystore, appears in the keystore location.

Creating an Auto-Login or a Local Auto-Login Software Keystore

As an alternative to password-protected keystores, you can create either an auto-login or local auto-login software keystore.

Both of these keystores have system-generated passwords. They are also PKCS#12-based files. The auto-login software keystore can be opened from different computers from the computer where this keystore resides, but the local auto-login software keystore can only be opened from the computer on which it was created. Both the auto-login and local auto-login keystores are created from the password-protected software keystores.

1. Log in to the database instance as a user who has been granted the ADMINISTER KEY MANAGEMENT or SYSKM privilege.

   For example:

   ```
   sqlplus sec_admin as syskm
   Enter password: password
   Connected.
   ```

   If SQL*Plus is already open and you had modified the init.ora file to set the WALLET_ROOT parameter during this time, then reconnect to SQL*Plus. The database session must be changed before the init.ora changes can take effect.

2. Create a password-protected software keystore.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT CREATE KEYSTORE '/etc/ORACLE/WALLETS/orcl'
   IDENTIFIED BY keystore_password;
   ```

3. Create the auto-login or local auto-login keystore by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT CREATE [LOCAL] AUTO_LOGIN KEYSTORE
   FROM KEYSTORE 'keystore_location'
   IDENTIFIED BY software_keystore_password;
   ```

   In this specification:

   - LOCAL enables you to create a local auto-login software keystore. Otherwise, omit this clause if you want the keystore to be accessible by other computers.
   - *keystore_location* is the path to the directory location of the password-protected keystore for which you want to create the auto-login keystore (for

example, /etc/ORACLE/WALLETS/orcl). Enclose this setting in single quotation marks (' '). To find this location, query the WRL_PARAMETER column of the V$ENCRYPTION_WALLET view.

- *software_keystore_password* is the password-protected keystore for which you want to create the auto-login keystore.

For example, to create an auto-login software keystore of the password-protected keystore that is located in the/etc/ORACLE/WALLETS/orcl directory:

```
ADMINISTER KEY MANAGEMENT CREATE AUTO_LOGIN KEYSTORE
FROM KEYSTORE '/etc/ORACLE/WALLETS/orcl'
IDENTIFIED BY password;

keystore altered.
```

After you run this statement, the cwallet.sso file appears in the keystore location. The ewallet.p12 file is the password-protected wallet.

Follow these guidelines:

- Do not remove the PKCS#12 wallet (ewallet.p12 file) after you create the auto-login keystore (.sso file). You must have the PKCS#12 wallet to regenerate or rekey the TDE master encryption key in the future. By default, this file is located in the admin/*db_unique_name*/wallet directory of the $ORACLE_BASE or $ORACLE_HOME location.
- Remember that Transparent Data Encryption uses an auto login keystore only if it is available at the correct location (WALLET_ROOT, WALLET_LOCATION, ENCRYPTION_WALLET_LOCATION, or the default keystore location), and the SQL statement to open an encrypted keystore has not already been executed. Note that auto-login keystores are encrypted, because they have system-generated passwords. If you have the ENCRYPTION_WALLET_LOCATION parameter set, then be aware this parameter is deprecated. Oracle recommends that you use the WALLET_ROOT static initialization parameter and TDE_CONFIGURATION dynamic initialization parameter instead.

Related Topics

- Creating a Password-Protected Software Keystore
  A password-protected software keystore requires a user password, which is used to protect the keys and credentials stored in the keystore.

Step 3: Open the Software Keystore

Depending on the type of keystore you create, you must manually open the keystore before you can use it.

- About Opening Software Keystores
  A password-protected software keystore must be open before any TDE master encryption keys can be created or accessed in the keystore.
- Opening a Software Keystore
  To open a software keystore, you must use the ADMINISTER KEY MANAGEMENT statement with the SET KEYSTORE OPEN clause.

About Opening Software Keystores

A password-protected software keystore must be open before any TDE master encryption keys can be created or accessed in the keystore.

Many Transparent Data Encryption operations require the software keystore to be open. There are two ways that you can open the software keystore:

- Manually open the keystore by issuing the `ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN` statement. Afterward, you can perform the operation.
- Include the `FORCE KEYSTORE` clause in the `ADMINISTER KEY MANAGEMENT` statement that is used to perform the operation. `FORCE KEYSTORE` temporarily opens the keystore for the duration of the operation, and when the operation completes, the keystore is closed again. `FORCE KEYSTORE` is useful for situations when the database is heavily loaded. In this scenario, because of concurrent access to encrypted objects in the database, the auto-login keystore continues to open immediately after it has been closed but before a user has had chance to open the password-based keystore.

Keystores can be in the following states: open, closed, open but with no master encryption key, open but with an unknown master encryption key, undefined, or not available (that is, not present in the `WALLET_ROOT` location).

After you manually open a keystore, it remains open until you manually close it. Each time you restart a database instance, you must manually open the password keystore to reenable encryption and decryption operations.

You can check the status of whether a keystore is open or not by querying the `STATUS` column of the `V$ENCRYPTION_WALLET` view.

Related Topics

- Performing Operations That Require a Keystore Password
  Many `ADMINISTER KEY MANAGEMENT` operations require access to a keystore password, for both software and hardware keystores.
- How Keystore Open and Close Operations Work in United Mode
  You should be aware of how keystore open and close operations work in united mode.

Opening a Software Keystore

To open a software keystore, you must use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEYSTORE OPEN` clause.

1. Log in to the database instance as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.

   For example:

   ```
   sqlplus sec_admin as syskm
   Enter password: password
   Connected.
   ```

2. Run the `ADMINISTER KEY MANAGEMENT` statement to open the keystore.

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
IDENTIFIED BY password;
keystore altered.
```

To switch over to opening the password-protected software keystore when an auto-login keystore is configured and is currently open, specify the `FORCE KEYSTORE` clause as follows.

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
FORCE KEYSTORE
IDENTIFIED BY EXTERNAL STORE;
keystore altered.
```

Here, the `IDENTIFIED BY EXTERNAL STORE` clause is included in the statement because the keystore credentials exist in an external store. This enables the password-protected keystore to be opened without specifying the keystore password within the statement itself.

If the `WALLET_ROOT` parameter has been set, then Oracle Database finds the external store by searching in this path: *`WALLET_ROOT`*`/`*`PDB_GUID`*`/tde_seps`.

3. Confirm that the keystore is open.

```
SELECT STATUS FROM V$ENCRYPTION_WALLET;
```

Note that if the keystore is open but you have not created a TDE master encryption key yet, the `STATUS` column of the `V$ENCRYPTION_WALLET` view reminds you with an `OPEN_NO_MASTER_KEY` status.

Step 4: Set the TDE Master Encryption Key in the Software Keystore

Once the keystore is open, you can set a TDE master encryption key for it.

- About Setting the Software Keystore TDE Master Encryption Key
  The TDE master encryption key is stored in the keystore.
- Setting the TDE Master Encryption Key in the Software Keystore
  To set the TDE master encryption key in a software keystore, use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEY` clause.

About Setting the Software Keystore TDE Master Encryption Key

The TDE master encryption key is stored in the keystore.

The TDE master encryption key protects the TDE table keys and tablespace encryption keys. By default, the TDE master encryption key is a key that TDE generates. You can find if a keystore has no TDE master encryption key set or an unknown TDE master encryption key by querying the `STATUS` column of the `V$ENCRYPTION_WALLET` view.

You can manually create a master encryption ID outside the database, which is useful for Cloud environments. You also can create TDE master encryption keys for use later on, and then manually activate them.

Related Topics

- Creating User-Defined TDE Master Encryption Keys
  You can create a user-defined TDE master encryption key outside the database by generating a TDE master encryption key ID.

- Creating TDE Master Encryption Keys for Later Use
  You can create a TDE master encryption key that can be activated at a later date.

Setting the TDE Master Encryption Key in the Software Keystore

To set the TDE master encryption key in a software keystore, use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEY` clause.

1. Log in to the database instance as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.

   For example:

   ```
   sqlplus sec_admin as syskm
   Enter password: password
   Connected.
   ```

2. Ensure that the database is open in `READ WRITE` mode.

   You can set the TDE master encryption key if `OPEN_MODE` is set to `READ WRITE`. To find the status, query the `OPEN_MODE` column of the `V$DATABASE` dynamic view. (If you cannot access this view, then connect as `SYSDBA` and try the query again. In order to connect as `SYSKM` for this type of query, you must create a password file for it.)

3. Set the TDE master encryption key in the software by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT SET KEY
   [USING TAG 'tag']
   [FORCE KEYSTORE]
   IDENTIFIED BY [EXTERNAL STORE] | keystore_password
   [WITH BACKUP [USING 'backup_identifier']];
   ```

   In this specification:

   - *tag* is the associated attributes and information that you define. Enclose this setting in single quotation marks (`' '`).
   - `FORCE KEYSTORE` temporarily opens the password-protected keystore for this operation. You must open the keystore for this operation.
   - `FORCE KEYSTORE` should be included if the keystore is closed. This automatically opens the keystore before setting the TDE master encryption key. The `FORCE KEYSTORE` clause also switches over to opening the password-protected software keystore when an auto-login keystore is configured and is currently open.
   - `IDENTIFIED BY` specifies the keystore password. Alternatively, if the keystore password is in an external store, you can use the `IDENTIFIED BY EXTERNAL STORE` clause.
   - `WITH BACKUP` creates a backup of the keystore. You must use this option for password-protected keystores. Optionally, you can use the `USING` clause to add a brief description of the backup. Enclose this description in single quotation marks (`' '`). This identifier is appended to the named keystore file (for example, `ewallet_`*time_stamp*`_emp_key_backup.p12`, with `emp_key_backup` being the backup identifier). Follow the file naming conventions that your operating system uses.

   For example:

```
ADMINISTER KEY MANAGEMENT SET KEY
FORCE KEYSTORE
IDENTIFIED BY keystore_password
WITH BACKUP USING 'emp_key_backup';

keystore altered.
```

Step 5: Encrypt Your Data

Now that you have completed the keystore configuration, you can begin to encrypt data in the PDB.

You can encrypt data in individual table columns or in entire tablespaces or databases.

Related Topics

- Encrypting Columns in Tables
  You can use Transparent Data Encryption to encrypt individual columns in database tables.
- Encryption Conversions for Tablespaces and Databases
  You can perform encryption operations on both offline and online tablespaces and databases.

Configuring a Hardware Keystore

A hardware keystore resides in a hardware security module (HSM), which is designed to store encryption keys.

- About Configuring a Hardware (External) Keystore
  A hardware keystore, also called an external keystore, is a separate server or device that provides security storage for encryption keys.
- Step 1: Configure the Hardware Keystore Type
  You can configure the hardware keystore type by setting the `TDE_CONFIGURATION` parameter.
- Step 2: Configure the Hardware Security Module
  To configure a third-party hardware security module, you must copy the PKCS#11 library to the correct location and follow your vendor's instructions.
- Step 3: Open the Hardware Keystore
  After you have configured the hardware security module, you must open the hardware keystore before it can be used.
- Step 4: Set the Hardware Keystore TDE Master Encryption Key
  After you have opened the hardware keystore, you are ready to set the hardware keystore TDE master encryption key.
- Step 5: Encrypt Your Data
  Now that you have completed the configuration for a hardware keystore or for an Oracle Key Vault keystore, you can begin to encrypt data.

About Configuring a Hardware (External) Keystore

A hardware keystore, also called an external keystore, is a separate server or device that provides security storage for encryption keys.

External keystores are external to an Oracle database. Oracle Database can interface with external keystores but cannot manipulate them outside of the Oracle interface. The Oracle database can request the external keystore to create a key but it cannot define how this key is stored in an external database. (Conversely, for software keystores that are created using TDE, Oracle Database has full control: that is, you can use SQL statements to manipulate this type of keystore.) Examples of external keystores are hardware security modules or Oracle Key Vault keystores. External keystores among multiple databases can be managed centrally, such as with Oracle Key Vault.

To configure a keystore for a hardware security module (hardware keystore), you must first include the keystore type in the `TDE_CONFIGURATION` parameter setting, configure and open the hardware keystore, and then set the hardware keystore TDE master encryption key. In short, there is one hardware keystore per database, and the database locates this keystore by checking the keystore type that you define in the `TDE_CONFIGURATION` parameter.

How you specify the `IDENTIFIED BY` clause when you run the `ADMINISTER KEY MANAGEMENT` statement depends on the type of hardware keystore. In most cases, and in the examples throughout this guide, you would use the following syntax for a hardware security module (`HSM`) or Cloud Key Management Service (`KMS`) keystore:

```
IDENTIFIED BY "user_name:password"
```

However, depending on your site's configuration of HSMs, the syntax for the credential may be *"password:user_name"*.

For an Oracle Key Vault keystore, you can omit the *user_name* and colon, but you must include the quotation marks:

```
IDENTIFIED BY "password"
```

After you configure the hardware keystore, you are ready to begin encrypting your data.

Step 1: Configure the Hardware Keystore Type

You can configure the hardware keystore type by setting the `TDE_CONFIGURATION` parameter.

1. Log in to the database instance as a user who has been granted the `SYSDBA` administrative privilege.

   For example:

   ```
   sqlplus sec_admin as sysdba
   Enter password: password
   ```

2. Set the `TDE_CONFIGURATION` dynamic initialization parameter to specify the keystore type by using the following syntax:

   ```
   ALTER SYSTEM SET TDE_CONFIGURATION="KEYSTORE_CONFIGURATION=keystore_type"
   SCOPE=scope_type ;
   ```

   In this specification:

   - *keystore_type* can be one of the following types:
     - `HSM` to configure a hardware security module (HSM) keystore

  - `OKV` to configure an Oracle Key Vault keystore
- *`scope_type`* sets the type of scope (for example, `both`, `memory`, or `spfile`.

For example, to configure an HSM keystore if the parameter file (`pfile`) is in use, set `scope` to `memory`:

```
ALTER SYSTEM SET TDE_CONFIGURATION="KEYSTORE_CONFIGURATION=HSM"
scope=memory;
```

To configure an Oracle Key Vault keystore if the server parameter file (`spfile`) is in use, set `scope` to `both`:

```
ALTER SYSTEM SET TDE_CONFIGURATION="KEYSTORE_CONFIGURATION=OKV"
scope=both;
```

3. Restart the database.

```
SHUTDOWN IMMEDIATE
STARTUP
```

Step 2: Configure the Hardware Security Module

To configure a third-party hardware security module, you must copy the PKCS#11 library to the correct location and follow your vendor's instructions.

If you are using Oracle Key Vault, then you can bypass this section.

1. Copy the PKCS#11 library to its correct path.

   Your hardware keystore vendor should provide you with an associated PKCS#11 library. Only one PKCS#11 library is supported at a time. If you want to use a hardware keystore from a new vendor, then you must replace the PKCS#11 library from the earlier vendor with the library from the new vendor.

   Copy this library to the appropriate location to ensure that Oracle Database can find this library:

   - **UNIX systems:** Use the following syntax to copy the library to this directory:

     ```
     /opt/oracle/extapi/[32,64]/hsm/{VENDOR}/{VERSION}/libapiname.so
     ```

   - **Windows systems:** Use the following syntax to copy the library to this directory:

     ```
     %SYSTEM_DRIVE%\oracle\extapi\[32,64]\hsm\{VENDOR}\{VERSION}
     \libapiname.dll
     ```

   In this specification:

   - `[32,64]` specifies whether the supplied binary is 32 bits or 64 bits.
   - *`VENDOR`* stands for the name of the vendor supplying the library
   - *`VERSION`* refers to the version of the library. This should preferably be in the format, *`number.number.number`*
   - *`apiname`* requires no special format. However, the *`apiname`* must be prefixed with the word `lib`, as illustrated in the syntax.

2. Follow your vendor's instructions to set up the hardware keystore.

   Use your hardware keystore management interface and the instructions provided by your HSM vendor to set up the hardware keystore. Create the user account and password that must be used by the database to interact with the

hardware keystore. This process creates and configures a hardware keystore that communicates with your Oracle database.

Step 3: Open the Hardware Keystore

After you have configured the hardware security module, you must open the hardware keystore before it can be used.

- About Opening Hardware Keystores
  You must open the hardware keystore so that it is accessible to the database before you can perform any encryption or decryption.
- Opening a Hardware Keystore
  To open a hardware keystore, use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEYSTORE OPEN` clause.

About Opening Hardware Keystores

You must open the hardware keystore so that it is accessible to the database before you can perform any encryption or decryption.

If a recovery operation is needed on your database (for example, if the database was not cleanly shut down, and has an encrypted tablespace that needs recovery), then you must open the hardware keystore before you can open the database itself.

There are two ways that you can open the hardware keystore:

- Manually open the keystore by issuing the `ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN` statement. Afterward, you can perform the operation.
- Include the `FORCE KEYSTORE` clause in the `ADMINISTER KEY MANAGEMENT` statement. `FORCE KEYSTORE` temporarily opens the keystore for the duration of the operation, and when the operation completes, the keystore is closed again. `FORCE KEYSTORE` is useful for situations when the database is heavily loaded. In this scenario, because of concurrent access to encrypted objects in the database, the auto-login keystore continues to open immediately after it has been closed but before a user has had a chance to open the password-based keystore.

To check the status of the keystore, query the `STATUS` column of the `V$ENCRYPTION_WALLET` view. Keystores can be in the following states: `CLOSED`, `NOT_AVAILABLE` (that is, not present in the `WALLET_ROOT` location), `OPEN`, `OPEN_NO_MASTER_KEY`, `OPEN_UNKNOWN_MASTER_KEY_STATUS`.

Be aware that for hardware keystores, if the database is in the mounted state, then it cannot check if the master key is set because the data dictionary is not available. In this situation, the status will be `OPEN_UNKNOWN_MASTER_KEY_STATUS`.

Related Topics

- How Keystore Open and Close Operations Work in United Mode
  You should be aware of how keystore open and close operations work in united mode.

Opening a Hardware Keystore

To open a hardware keystore, use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEYSTORE OPEN` clause.

1. Log in to the database instance as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.

   For example:

   ```
   sqlplus sec_admin as syskm
   Enter password: password
   Connected.
   ```

2. Open the hardware keystore by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
   [FORCE KEYSTORE]
   IDENTIFIED BY "hardware_keystore_password";
   ```

   In this specification:

   - `FORCE KEYSTORE` enables the keystore operation if the keystore is closed.
   - `IDENTIFIED BY` can be one of the following settings:
     - `EXTERNAL STORE` uses the keystore password stored in the external store to perform the keystore operation.
     - *`hardware_keystore_credentials`* refers to the credentials for either an HSM or an Oracle Key Vault hardware keystore. For an HSM, specify the credentials using this format, enclosed in quotation marks and separating the components with a colon: *"`user_name`:`password`"*, with *`user_name`* being the user who created the HSM and password being this user's password. For Oracle Key Vault, enter only the password of the user who created the keystore. Enclose this password with quotation marks.

   For example, for an HSM:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
   IDENTIFIED BY "psmith:password";

   keystore altered.
   ```

3. Repeat this procedure each time you restart the database instance.

Step 4: Set the Hardware Keystore TDE Master Encryption Key

After you have opened the hardware keystore, you are ready to set the hardware keystore TDE master encryption key.

- About Setting the Hardware Keystore TDE Master Encryption Key
  You must create a TDE master encryption key that is stored inside the hardware keystore.
- Setting a New TDE Master Encryption Key
  You should complete this procedure if you have not previously configured a hardware keystore for Transparent Data Encryption.

- Migration of a Previously Configured TDE Master Encryption Key
  You must migrate the previously configured TDE master encryption key if you previously configured a software keystore.

About Setting the Hardware Keystore TDE Master Encryption Key

You must create a TDE master encryption key that is stored inside the hardware keystore.

Oracle Database uses the master encryption key to encrypt or decrypt TDE table keys or tablespace encryption keys inside the hardware security module.

If you have not previously configured a software keystore for TDE, then you must set the master encryption key. If you have already configured a software keystore for TDE, then you must migrate it to the hardware security module.

Along with the current master encryption key, Oracle wallets maintain historical master encryption keys that are generated after every re-key operation that rekeys the master encryption key. These historical master keys help to restore Oracle database backups that were taken previously using one of the historical master encryption keys.

Related Topics

- Setting a New TDE Master Encryption Key
  You should complete this procedure if you have not previously configured a hardware keystore for Transparent Data Encryption.
- Migration of a Previously Configured TDE Master Encryption Key
  You must migrate the previously configured TDE master encryption key if you previously configured a software keystore.

Setting a New TDE Master Encryption Key

You should complete this procedure if you have not previously configured a hardware keystore for Transparent Data Encryption.

1. Log in to the database instance as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.

   For example:

   ```
   sqlplus sec_admin as syskm
   Enter password: password
   Connected.
   ```

2. Ensure that the database is open in `READ WRITE` mode.

   You can set the master encryption key if `OPEN_MODE` is set to `READ WRITE`. To find the status, for a non-multitenant environment, query the `OPEN_MODE` column of the `V$DATABASE` dynamic view. If you are in a multitenant environment, then run the `show_pdbs` command.

3. Set the new master encryption key by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT SET KEY
   [USING TAG 'tag']
   [FORCE KEYSTORE]
   IDENTIFIED BY [EXTERNAL STORE | "hardware_keystore_credentials"];
   ```

   In this specification:

- *tag* is the associated attributes and information that you define. Enclose this setting in single quotation marks (' ').
- FORCE KEYSTORE temporarily opens the password-protected keystore for this operation. You must open the keystore for this operation.
- IDENTIFIED BY can be one of the following settings:
  - EXTERNAL STORE uses the keystore password stored in the external store to perform the keystore operation.
  - *hardware_keystore_credentials* refers to the credentials for either an HSM or an Oracle Key Vault hardware keystore. For an HSM, specify the credentials using this format, enclosed in quotation marks and separating the components with a colon: "*user_name*:*password*", with *user_name* being the user who created the HSM and password being this user's password. For Oracle Key Vault, enter only the password of the user who created the keystore. Enclose this password with quotation marks.

For example:

```
ADMINISTER KEY MANAGEMENT SET KEY
FORCE KEYSTORE
IDENTIFIED BY "psmith:password";

keystore altered.
```

Related Topics

- Creating a TDE Master Encryption Key for Later Use
  A keystore must be opened before you can create a TDE master encryption key for use later on.
- *Oracle Database Administrator's Guide*

Migration of a Previously Configured TDE Master Encryption Key

You must migrate the previously configured TDE master encryption key if you previously configured a software keystore.

Tools such as Oracle Data Pump and Oracle Recovery Manager require access to the old software keystore to perform decryption and encryption operations on data exported or backed up using the software keystore. You can migrate from the software to the hardware keystore.

Along with the current master encryption key, Oracle keystores maintain historical master encryption keys that are generated after every re-key operation that rotates the master encryption key. These historical master encryption keys help to restore Oracle database backups that were taken previously using one of the historical master encryption keys.

Related Topics

- Migrating Between a Software Password Keystore and a Hardware Keystore
  You can migrate between password-protected software keystores and hardware keystores.

Step 5: Encrypt Your Data

Now that you have completed the configuration for a hardware keystore or for an Oracle Key Vault keystore, you can begin to encrypt data.

Oracle Key Vault Administrator's Guide describes how to configure Oracle Key Vault keystores.

You can encrypt individual columns in a table or entire tablespaces.

Related Topics

- Encrypting Columns in Tables
  You can use Transparent Data Encryption to encrypt individual columns in database tables.
- Encryption Conversions for Tablespaces and Databases
  You can perform encryption operations on both offline and online tablespaces and databases.

Encrypting Columns in Tables

You can use Transparent Data Encryption to encrypt individual columns in database tables.

- About Encrypting Columns in Tables
  You can encrypt individual columns in tables.
- Data Types That Can Be Encrypted with TDE Column Encryption
  Oracle Database supports a specific set of data types that can be used with TDE column encryption.
- Restrictions on Using TDE Column Encryption
  TDE column encryption is performed at the SQL layer. Oracle Database utilities that bypass the SQL layer cannot use TDE column encryption services.
- Creating Tables with Encrypted Columns
  Oracle Database provides a selection of different algorithms that you can use to define the encryption used in encrypted columns.
- Encrypting Columns in Existing Tables
  You can encrypt columns in existing tables. As with new tables, you have a choice of different algorithms to use to definite the encryption.
- Creating an Index on an Encrypted Column
  You can create an index on an encrypted column.
- Adding Salt to an Encrypted Column
  Salt, which is a random string added to data before encryption, is a way to strengthen the security of encrypted data. .
- Removing Salt from an Encrypted Column
  You can use the `ALTER TABLE` SQL statement to remove salt from an encrypted column.
- Changing the Encryption Key or Algorithm for Tables with Encrypted Columns
  You can use the `ALTER TABLE` SQL statement to change the encryption key or algorithm used in encrypted columns.

About Encrypting Columns in Tables

You can encrypt individual columns in tables.

Whether you choose to encrypt individual columns or entire tablespaces depends on the data types that the table has. There are also several features that do not support TDE column encryption.

Related Topics

- Data Types That Can Be Encrypted with TDE Column Encryption
  Oracle Database supports a specific set of data types that can be used with TDE column encryption.
- Restrictions on Using TDE Column Encryption
  TDE column encryption is performed at the SQL layer. Oracle Database utilities that bypass the SQL layer cannot use TDE column encryption services.

Data Types That Can Be Encrypted with TDE Column Encryption

Oracle Database supports a specific set of data types that can be used with TDE column encryption.

You can encrypt data columns that use a variety of different data types.

Supported data types are as follows:

- `BINARY_DOUBLE`
- `BINARY_FLOAT`
- `CHAR`
- `DATE`
- `INTERVAL DAY TO SECOND`
- `INTERVAL YEAR TO MONTH`
- `NCHAR`
- `NUMBER`
- `NVARCHAR2`
- `RAW` (legacy or extended)
- `TIMESTAMP` (includes `TIMESTAMP WITH TIME ZONE` and `TIMESTAMP WITH LOCAL TIME ZONE`)
- `VARCHAR2` (legacy or extended)

If you want to encrypt large binary objects (LOBs), then you can use Oracle SecureFiles. Oracle SecureFiles enables you to store LOB data securely. To encrypt a LOB using SecureFiles, you use the `CREATE TABLE` or `ALTER TABLE` statements.

You cannot encrypt a column if the encrypted column size is greater than the size allowed by the data type of the column.

Table 3-1 shows the maximum allowable sizes for various data types.

Table 3-1 Maximum Allowable Size for Data Types

Data Type	Maximum Size
CHAR	1932 bytes
VARCHAR2 (legacy)	3932 bytes
VARCHAR2 (extended)	32,699 bytes
NVARCHAR2 (legacy)	1966 bytes
NVARCHAR2 (extended)	16,315 bytes
NCHAR	966 bytes
RAW (extended)	32,699 bytes

> **Note:**
>
> TDE tablespace encryption does not have these data type restrictions.

Related Topics

- Restrictions on Using Transparent Data Encryption Tablespace Encryption
  You should be aware of restrictions on using Transparent Data Encryption when you encrypt a tablespace.
- *Oracle Database SecureFiles and Large Objects Developer's Guide*

Restrictions on Using TDE Column Encryption

TDE column encryption is performed at the SQL layer. Oracle Database utilities that bypass the SQL layer cannot use TDE column encryption services.

Do not use TDE column encryption with the following database features:

- Index types other than B-tree
- Range scan search through an index
- Synchronous change data capture
- Transportable tablespaces
- Columns that have been created as identity columns

In addition, you cannot use TDE column encryption to encrypt columns used in foreign key constraints.

Applications that must use these unsupported features can use the DBMS_CRYPTO PL/SQL package for their encryption needs.

Transparent Data Encryption protects data stored on a disk or other media. It does not protect data in transit. Use the network encryption solutions discussed in *Oracle Database Security Guide* to encrypt data over the network.

Related Topics

- How Transparent Data Encryption Works with Export and Import Operations
  Oracle Data Pump can export and import tables that contain encrypted columns, as well as encrypt entire dump sets.
- Data Types That Can Be Encrypted with TDE Column Encryption
  Oracle Database supports a specific set of data types that can be used with TDE column encryption.

Creating Tables with Encrypted Columns

Oracle Database provides a selection of different algorithms that you can use to define the encryption used in encrypted columns.

- About Creating Tables with Encrypted Columns
  You can use the `CREATE TABLE` SQL statement to create a table with an encrypted column.
- Creating a Table with an Encrypted Column Using the Default Algorithm
  By default, TDE uses the `AES` encryption algorithm with a 192-bit key length (`AES192`).
- Creating a Table with an Encrypted Column Using No Algorithm or a Non-Default Algorithm
  You an use the `CREATE TABLE` SQL statement to create a table with an encrypted column.
- Using the NOMAC Parameter to Save Disk Space and Improve Performance
  You can bypass checks that TDE performs. This can save up to 20 bytes of disk space per encrypted value.
- Example: Using the NOMAC Parameter in a CREATE TABLE Statement
  You can use the `CREATE TABLE` SQL statement to encrypt a table column using the `NOMAC` parameter.
- Example: Changing the Integrity Algorithm for a Table
  You can use the `ALTER TABLE` SQL statement in different foregrounds to convert different offline tablespaces in parallel.
- Creating an Encrypted Column in an External Table
  The external table feature enables you to access data in external sources as if the data were in a database table.

About Creating Tables with Encrypted Columns

You can use the `CREATE TABLE` SQL statement to create a table with an encrypted column.

To create relational tables with encrypted columns, you can specify the SQL `ENCRYPT` clause when you define database columns with the `CREATE TABLE` SQL statement.

Creating a Table with an Encrypted Column Using the Default Algorithm

By default, TDE uses the `AES` encryption algorithm with a 192-bit key length (`AES192`).

If you encrypt a table column without specifying an algorithm, then the column is encrypted using the `AES192` algorithm.

TDE adds salt to plaintext before encrypting it. Adding salt makes it harder for attackers to steal data through a brute force attack. TDE also adds a Message Authentication Code (MAC) to the data for integrity checking. The SHA-1 integrity algorithm is used by default.

- To create a table that encrypts a column, use the CREATE TABLE SQL statement with the ENCRYPT clause.

  For example, to encrypt a table column using the default algorithm:

  ```
  CREATE TABLE employee (
       first_name VARCHAR2(128),
       last_name VARCHAR2(128),
       empID NUMBER,
       salary NUMBER(6) ENCRYPT);
  ```

  This example creates a new table with an encrypted column (salary). The column is encrypted using the default encryption algorithm (AES192). Salt and MAC are added by default. This example assumes that the keystore is open and a master encryption key is set.

> **Note:**
>
> If there are multiple encrypted columns in a table, then all of these columns must use the same pair of encryption and integrity algorithms.
>
> Salt is specified at the column level. This means that an encrypted column in a table can choose not to use salt irrespective of whether or not other encrypted columns in the table use salt.

Creating a Table with an Encrypted Column Using No Algorithm or a Non-Default Algorithm

You an use the CREATE TABLE SQL statement to create a table with an encrypted column.

By default, TDE adds salt to plaintext before encrypting it. Adding salt makes it harder for attackers to steal data through a brute force attack. However, if you plan to index the encrypted column, then you must use the NO SALT parameter.

- To create a table that uses an encrypted column that is a non-default algorithm or no algorithm, run the CREATE TABLE SQL statement as follows:
  - If you do not want to use any algorithm, then include the ENCRYPT NO SALT clause.
  - If you want to use a non-default algorithm, then use the ENCRYPT USING clause, followed by one of the following algorithms enclosed in single quotation marks:
    - 3DES168
    - AES128
    - AES192 (default)
    - AES256

The following example shows how to specify encryption settings for the `empID` and `salary` columns.

```
CREATE TABLE employee (
     first_name VARCHAR2(128),
     last_name VARCHAR2(128),
     empID NUMBER ENCRYPT NO SALT,
     salary NUMBER(6) ENCRYPT USING '3DES168');
```

In this example:

- The `empID` column is encrypted and does not use salt. Both the `empID` and `salary` columns will use the `3DES168` encryption algorithm, because all of the encrypted columns in a table must use the same encryption algorithm.
- The `salary` column is encrypted using the `3DES168` encryption algorithm. Note that the string that specifies the algorithm must be enclosed in single quotation marks (' '). The `salary` column uses salt by default.

Using the NOMAC Parameter to Save Disk Space and Improve Performance

You can bypass checks that TDE performs. This can save up to 20 bytes of disk space per encrypted value.

If the number of rows and encrypted columns in the table is large, then bypassing TDE checks can add up to a significant amount of disk space. In addition, this saves processing cycles and reduces the performance overhead associated with TDE.

TDE uses the `SHA-1` integrity algorithm by default. All of the encrypted columns in a table must use the same integrity algorithm. If you already have a table column using the `SHA-1` algorithm, then you cannot use the `NOMAC` parameter to encrypt another column in the same table.

- To bypass the integrity check during encryption and decryption operations, use the `NOMAC` parameter in the `CREATE TABLE` and `ALTER TABLE` statements.

Related Topics

- Performance and Storage Overhead of Transparent Data Encryption
  The performance of Transparent Data Encryption can vary.

Example: Using the NOMAC Parameter in a CREATE TABLE Statement

You can use the `CREATE TABLE` SQL statement to encrypt a table column using the `NOMAC` parameter.

Example 3-1 creates a table with an encrypted column. The `empID` column is encrypted using the `NOMAC` parameter.

Example 3-1 Using the NOMAC parameter in a CREATE TABLE statement

```
CREATE TABLE employee (
     first_name VARCHAR2(128),
     last_name VARCHAR2(128),
     empID NUMBER ENCRYPT 'NOMAC' ,
     salary NUMBER(6));
```

Example: Changing the Integrity Algorithm for a Table

You can use the ALTER TABLE SQL statement in different foregrounds to convert different offline tablespaces in parallel.

Example 3-2 shows how to change the integrity algorithm for encrypted columns in a table. The encryption algorithm is set to 3DES168 and the integrity algorithm is set to SHA-1. The second ALTER TABLE statement sets the integrity algorithm to NOMAC.

Example 3-2 Changing the Integrity Algorithm for a Table

```
ALTER TABLE EMPLOYEE REKEY USING '3DES168' 'SHA-1';

ALTER TABLE EMPLOYEE REKEY USING '3DES168' 'NOMAC';
```

Creating an Encrypted Column in an External Table

The external table feature enables you to access data in external sources as if the data were in a database table.

External tables can be updated using the ORACLE_DATAPUMP access driver.

- To encrypt specific columns in an external table, use the ENCRYPT clause when you define those columns:

  A system-generated key encrypts the columns. For example, the following CREATE TABLE SQL statement encrypts the ssn column using the 3DES168 algorithm:

  ```
  CREATE TABLE emp_ext (
      first_name,
      ....
      ssn ENCRYPT USING '3DES168',
      ....
  ```

If you plan to move an external table to a new location, then you cannot use a randomly generated key to encrypt the columns. This is because the randomly generated key will not be available at the new location.

For such scenarios, you should specify a password while you encrypt the columns. After you move the data, you can use the same password to regenerate the key required to access the encrypted column data at the new location.

Table partition exchange also requires a password-protected TDE table key.

Example 3-3 creates an external table using a password to create the TDE table key.

Example 3-3 Creating a New External Table with a Password-Generated TDE Table Key

```
CREATE TABLE emp_ext (
     first_name,
     last_name,
     empID,
     salary,
     ssn ENCRYPT IDENTIFIED BY password
)  ORGANIZATION EXTERNAL
   (
    TYPE ORACLE_DATAPUMP
    DEFAULT DIRECTORY "D_DIR"
```

```
    LOCATION('emp_ext.dat')
    )
    REJECT LIMIT UNLIMITED
AS SELECT * FROM EMPLOYEE;
```

Encrypting Columns in Existing Tables

You can encrypt columns in existing tables. As with new tables, you have a choice of different algorithms to use to definite the encryption.

- About Encrypting Columns in Existing Tables
  The `ALTER TABLE` SQL statement enables you to encrypt columns in an existing table.
- Adding an Encrypted Column to an Existing Table
  You can encrypt columns in existing tables, use a different algorithm, and use `NO SALT` to index the column.
- Encrypting an Unencrypted Column
  You can use the `ALTER TABLE MODIFY` statement to encrypt an existing unencrypted column.
- Disabling Encryption on a Column
  You may want to disable encryption for reasons of compatibility or performance.

About Encrypting Columns in Existing Tables

The `ALTER TABLE` SQL statement enables you to encrypt columns in an existing table.

To add an encrypted column to an existing table, or to encrypt or decrypt an existing column, you use the `ALTER TABLE` SQL statement with the `ADD` or `MODIFY` clause.

Adding an Encrypted Column to an Existing Table

You can encrypt columns in existing tables, use a different algorithm, and use `NO SALT` to index the column.

- To add an encrypted column to an existing table, use the `ALTER TABLE ADD` statement, specifying the new column with the `ENCRYPT` clause.

Example 3-4 adds an encrypted column, `ssn`, to an existing table, called `employee`. The `ssn` column is encrypted with the default `AES192` algorithm. Salt and MAC are added by default.

Example 3-4 Adding an Encrypted Column to an Existing Table

```
ALTER TABLE employee ADD (ssn VARCHAR2(11) ENCRYPT);
```

Encrypting an Unencrypted Column

You can use the `ALTER TABLE MODIFY` statement to encrypt an existing unencrypted column.

- To encrypt an existing unencrypted column, use the `ALTER TABLE MODIFY` statement, specifying the unencrypted column with the `ENCRYPT` clause.

The following example encrypts the `first_name` column in the `employee` table. The `first_name` column is encrypted with the default `AES192` algorithm. Salt is added to the

data, by default. You can encrypt the column using a different algorithm. If you want to index a column, then you must specify `NO SALT`. You can also bypass integrity checks by using the `NOMAC` parameter.

```
ALTER TABLE employee MODIFY (first_name ENCRYPT);
```

The following example encrypts the `first_name` column in the employee table using the `NOMAC` parameter.

```
ALTER TABLE employee MODIFY (first_name ENCRYPT 'NOMAC');
```

Disabling Encryption on a Column

You may want to disable encryption for reasons of compatibility or performance.

- To disable column encryption, use the `ALTER TABLE MODIFY` command with the `DECRYPT` clause.

Example 3-5 decrypts the `first_name` column in the `employee` table.

Example 3-5 Turning Off Column Encryption

```
ALTER TABLE employee MODIFY (first_name DECRYPT);
```

Creating an Index on an Encrypted Column

You can create an index on an encrypted column.

The column being indexed must be encrypted without salt. If the column is encrypted with salt, then the `ORA-28338: cannot encrypt indexed column(s) with salt` error is raised.

- To create an index on an encrypted column, use the `CREATE INDEX` statement with the `ENCRYPT NO SALT` clause.

Example 3-6 shows how to create an index on a column that has been encrypted without salt.

Example 3-6 Creating Index on a Column Encrypted Without Salt

```
CREATE TABLE employee (
   first_name VARCHAR2(128),
   last_name VARCHAR2(128),
   empID NUMBER ENCRYPT NO SALT,
   salary NUMBER(6) ENCRYPT USING '3DES168');

CREATE INDEX employee_idx on employee (empID);
```

Adding Salt to an Encrypted Column

Salt, which is a random string added to data before encryption, is a way to strengthen the security of encrypted data. .

Salt ensures that the same plaintext data does not always translate to the same encrypted text. Salt removes the one common method that intruders use to steal data, namely, matching patterns of encrypted text. Adding salt requires an additional 16 bytes of storage per encrypted data value.

- To add or remove salt from encrypted columns, use the `ALTER TABLE MODIFY` SQL statement.

For example, suppose you want to encrypt the `first_name` column using salt. If the `first_name` column was encrypted without salt earlier, then the `ALTER TABLE MODIFY` statement reencrypts it using salt.

```
ALTER TABLE employee MODIFY (first_name ENCRYPT SALT);
```

Removing Salt from an Encrypted Column

You can use the `ALTER TABLE` SQL statement to remove salt from an encrypted column.

- To remove salt from an encrypted column, use the `ENCRYPT NO SALT` clause in the `ALTER TABLE` SQL statement.

For example, suppose you wanted to remove salt from the `first_name` column. If you must index a column that was encrypted using salt, then you can use this statement to remove the salt before indexing

```
ALTER TABLE employee MODIFY (first_name ENCRYPT NO SALT);
```

Changing the Encryption Key or Algorithm for Tables with Encrypted Columns

You can use the `ALTER TABLE` SQL statement to change the encryption key or algorithm used in encrypted columns.

Each table can have only one TDE table key for its columns. You can regenerate the TDE table key with the `ALTER TABLE` statement. This process generates a new key, decrypts the data in the table using the previous key, reencrypts the data using the new key, and then updates the table metadata with the new key information. You can also use a different encryption algorithm for the new TDE table key.

- To change the encryption key or algorithm for tables that contain encrypted columns, use the `ALTER TABLE` SQL statement with the `REKEY` or `REKEY USING` clause.

For example:

```
ALTER TABLE employee REKEY;
```

Example 3-7 regenerates the TDE table key for the `employee` table by using the `3DES168` algorithm.

Example 3-7 Changing an Encrypted Table Column Encryption Key and Algorithm

```
ALTER TABLE employee REKEY USING '3DES168';
```

Encryption Conversions for Tablespaces and Databases

You can perform encryption operations on both offline and online tablespaces and databases.

About Encryption Conversion for Tablespaces and Databases

The CREATE TABLESPACE SQL statement can be used to encrypt new tablespaces. ALTER TABLESPACE can encrypt existing tablespaces.

In addition to encrypting new and existing tablespaces, you can encrypt full databases, which entails the encryption of the Oracle-managed tablespaces (in this release, the SYSTEM, SYSAUX, TEMP, and UNDO tablespaces). To encrypt a full database, you use the ALTER TABLESPACE statement, not ALTER DATABASE, to encrypt the Oracle-managed tablespaces.

The following table compares the differences between an offline and an online encryption conversion of tablespaces and databases.

Table 3-2 Offline and Online Tablespace and Database Encryption Conversions

Functionality	Offline Conversion	Online Conversion
Release with minimum conversion capability	Oracle Database 11g release 2 (11.2)	Oracle Database 12c release 2 (12.2) and later

Table 3-2 (Cont.) Offline and Online Tablespace and Database Encryption Conversions

Functionality	Offline Conversion	Online Conversion
What can be backported?	The ability to encrypt or decrypt a data file with the AES128 algorithm (using `ALTER DATABASE DATAFILE data_file ENCRYPT` or DECRYPT) can be used in Oracle Database releases 12.1.0.2 and 11.2.0.4.	No
Algorithms supported	All symmetric encryption algorithms that TDE supports. See Supported Encryption and Integrity Algorithms. See also About Encryption Conversion for Existing Offline Tablespaces.	All symmetric encryption algorithm that TDE supports. See Supported Encryption and Integrity Algorithms. See also About Encryption Conversion for Existing Online Tablespaces.
When can the conversion be run?	When the tablespace is offline or the database is in the mount stage.	When the tablespace is online and database is open in read/write mode.
Is auxiliary space required for the conversion?	No	Yes. See Encrypting an Existing Tablespace with Online Conversion for guidelines.
Oracle Data Guard conversion guidelines	Convert both the primary and standby manually. Convert the standby first and then switch over to minimum downtime	After you convert the primary, the standby conversion takes place automatically. You cannot perform an online conversion directly on the standby.
Encrypt the SYSTEM, SYSAUX, and UNDO tablespaces (database conversion)	Oracle Database 12c release 2 (12.2) and later only. You must set COMPATIBILITY to 12.2.0.0.	Oracle Database 12c release 2 (12.2) and later only. You must set COMPATIBILITY to 12.2.0.0.
Can an existing TEMP tablespace be converted?	No, but you can create an encrypted TEMP tablespace in Oracle Database 12c release 2 (12.2) and later, make it the default temporary tablespace, and then drop the original TEMP tablespace.	No, but you can create an encrypted TEMP tablespace in Oracle Database 12c release 2 (12.2) and later, make it the default temporary tablespace, and then drop the original TEMP tablespace.

Table 3-2 (Cont.) Offline and Online Tablespace and Database Encryption Conversions

Functionality	Offline Conversion	Online Conversion
Can an existing tablespace be decrypted?	You only can decrypt a tablespace or data file that was previously encrypted by an offline encrypt operation. Oracle does not recommend that you decrypt the UNDO tablespace once it is encrypted.	Yes, but Oracle does not recommend that you decrypt the UNDO tablespace once it is encrypted.
Can encryption keys be rekeyed?	No, but after the tablespace is encrypted, you can then use online conversion to rekey in Oracle Database 12c release 2 (12.2) compatibility.	Yes
Can encryption operations be run in parallel?	You can run parallel encryption conversions at the data file level with multiple user sessions running.	You can run parallel encryption conversions at the tablespace level with multiple user sessions running.
What to do if an encryption conversion SQL statement fails to complete?	Re-issue the encryption or decryption SQL statement to ensure that all the data files within the tablespace are consistently either encrypted or decrypted.	Rerun the SQL statement but use the FINISH clause.

Impact of a Closed TDE Keystore on Encrypted Tablespaces

A TDE keystore can be closed or migrated when an Oracle-managed tablespace is encrypted, and the database system itself must be shut down to disallow operations on an Oracle-managed tablespace.

A closed TDE keystore has no impact on operations that involve an encrypted Oracle-managed tablespace (in this release, the SYSTEM, SYSAUX, TEMP, and UNDO tablespaces). This enables operations that are performed by background processes (for example, the log writer) to continue to work on these tablespaces while the TDE keystore is closed. If you want to disallow operations on an encrypted Oracle-managed tablespace, then you must shut down the database.

With regard to user-created tablespaces, a closed TDE keystore causes operations such as rotating a key or decrypting the tablespace to fail with an `ORA-28365 wallet is not open` error, just as it did in earlier releases. If you want to disallow operations on the user-created tablespace, then close the TDE keystore (or shut down the database).

User-created data can be copied into an encrypted Oracle-managed tablespace (for example, by an internal process such as DBMS_STATS statistics gathering) from a user-created tablespace while the TDE keystore is open. Closing the keystore does not prevent a user from viewing this data afterward, when the TDE keystore is in the CLOSED state at the time that you query the V$ENCRYPTION_WALLET view. Access to the original data by attempting to query an encrypted user-created tablespace will fail, resulting in an `ORA-28365 wallet is not open` error.

Table 3-3 describes the operations that are necessary to disallow or allow operations on encrypted data in user-created tablespaces and Oracle-managed tablespaces. For example, in the first scenario, both the user-created tablespaces and the Oracle-managed tablespaces are encrypted. In this case, for the encrypted data in the encrypted user-created tablespace, an administrator can close or open keystores, and shut down and open a database with an encrypted user-created tablespace. When an encrypted Oracle-managed tablespace is configured, the administrator can disallow operations by shutting down the database, and can allow operations by starting up in mount mode, opening the TDE keystore, and then opening the database. (It is necessary to open the TDE keystore *before* opening the database because the system may need the TDE master encryption key to decrypt the bootstrap dictionary tables, which are located in the encrypted Oracle-managed tablespace.) The N/A flags in this table refer to non-encrypted data, which you can always operate on, unless the instance is shut down.

Table 3-3 Necessary Commands to Disallow or Allow Operations on Encrypted Data

Tablespace Encryption Scenarios	Commands to Disallow Operations on Encrypted User-Created Tablespace Data	Commands to Disallow Operations on Encrypted Oracle-Managed Tablespace Data	Commands to Allow Operations on Encrypted User-Created Tablespace Data	Commands to Allow Operations on Encrypted Oracle-Managed Tablespace Data
Both user-created and Oracle-managed tablespaces encrypted	• `ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE IDENTIFIED BY` *`password`*`;` • `SHUTDOWN`	`SHUTDOWN`	• `STARTUP MOUNT;` • `ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY` *`password`*`;` • `ALTER DATABASE OPEN`	• `STARTUP MOUNT;` • `ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY` *`password`*`;` • `ALTER DATABASE OPEN`
User tablespace encrypted; Oracle-managed tablespace not encrypted	• `ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE IDENTIFIED BY` *`password`*`;` • `SHUTDOWN`	N/A	• `STARTUP MOUNT;` • `ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY` *`password`*`;` • `ALTER DATABASE OPEN`	N/A

Table 3-3 (Cont.) Necessary Commands to Disallow or Allow Operations on Encrypted Data

Tablespace Encryption Scenarios	Commands to Disallow Operations on Encrypted User-Created Tablespace Data	Commands to Disallow Operations on Encrypted Oracle-Managed Tablespace Data	Commands to Allow Operations on Encrypted User-Created Tablespace Data	Commands to Allow Operations on Encrypted Oracle-Managed Tablespace Data
User tablespace not encrypted; Oracle-managed tablespace encrypted	N/A	`SHUTDOWN`	N/A	• `STARTUP MOUNT;` • `ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY` *`password`*`;` • `ALTER DATABASE OPEN`
Neither user nor Oracle-managed tablespaces encrypted	N/A	N/A	N/A	N/A

Restrictions on Using Transparent Data Encryption Tablespace Encryption

You should be aware of restrictions on using Transparent Data Encryption when you encrypt a tablespace.

Note the following restrictions:

- Transparent Data Encryption (TDE) tablespace encryption encrypts or decrypts data during read and write operations, as opposed to TDE column encryption, which encrypts and decrypts data at the SQL layer. This means that most restrictions that apply to TDE column encryption, such as data type restrictions and index type restrictions, do not apply to TDE tablespace encryption.
- To perform import and export operations, use Oracle Data Pump.

Related Topics

- Encrypting an Unencrypted Column
  You can use the `ALTER TABLE MODIFY` statement to encrypt an existing unencrypted column.
- Data Types That Can Be Encrypted with TDE Column Encryption
  Oracle Database supports a specific set of data types that can be used with TDE column encryption.
- *Oracle Database Utilities*

Creating an Encrypted New Tablespace

When you create a new tablespace, you can configure its encryption settings during the creation process.

- Step 1: Set the COMPATIBLE Initialization Parameter for Tablespace Encryption
  You must set the `COMPATIBLE` initialization parameter before creating an encrypted tablespace.
- Step 2: Set the Tablespace TDE Master Encryption Key
  You should ensure that you have configured the TDE master encryption key.
- Step 3: Create the Encrypted Tablespace
  After you have set the `COMPATIBLE` initialization parameter, you are ready to create the encrypted tablespace.

Step 1: Set the COMPATIBLE Initialization Parameter for Tablespace Encryption

You must set the `COMPATIBLE` initialization parameter before creating an encrypted tablespace.

- About Setting the COMPATIBLE Initialization Parameter for Tablespace Encryption
  A minimum `COMPATIBLE` initialization parameter setting of `11.2.0.0` enables the full set of tablespace encryption features.
- Setting the COMPATIBLE Initialization Parameter for Tablespace Encryption
  To set the `COMPATIBLE` initialization parameter, you must edit the initialization parameter file for the database instance.

About Setting the COMPATIBLE Initialization Parameter for Tablespace Encryption

A minimum `COMPATIBLE` initialization parameter setting of `11.2.0.0` enables the full set of tablespace encryption features.

Setting the compatibility to `11.2.0.0` enables the following functionality:

- The `11.2.0.0` setting enables the database to use any of the four supported algorithms for data encryption (`3DES168`, `AES128`, `AES192`, and `AES256`).
- The `11.2.0.0` setting enables the migration of a key from a software keystore to a hardware keystore (ensure that the TDE master encryption key was configured for the hardware keystore)
- The `11.2.0.0` setting enables rekeying the TDE master encryption key

Be aware that once you set the `COMPATIBLE` parameter to `11.2.0.0`, the change is irreversible. To use tablespace encryption, ensure that the compatibility setting is at the minimum, which is `11.2.0.0`.

> **See Also:**
>
> - *Oracle Database SQL Language Reference* for more information about the `COMPATIBLE` parameter
> - *Oracle Database Administrator's Guide* for more information about initialization parameter files

Setting the COMPATIBLE Initialization Parameter for Tablespace Encryption

To set the `COMPATIBLE` initialization parameter, you must edit the initialization parameter file for the database instance.

1. Log in to the database instance.

   In a multitenant environment, log in to the PDB. For example:

   ```
   sqlplus sec_admin@hrpdb
   Enter password: password
   Connected.
   ```

   To find the available PDBs, run the `show pdbs` command. To check the current PDB, run the `show con_name` command.

2. Check the current setting of the `COMPATIBLE` parameter.

   For example:

   ```
   SHOW PARAMETER COMPATIBLE

   NAME                                 TYPE        VALUE
   ------------------------------------ -----------
   ------------------------------
   compatible                           string      11.2.0.0
   noncdbcompatible                     BOOLEAN     FALSE
   ```

3. If you must change the `COMPATIBLE` parameter, then complete the remaining steps in this procedure.

   The value should be 11.2.0.0 or higher.

4. From the command line, locate the initialization parameter file for the database instance.

   - **UNIX systems:** This file is in the *`ORACLE_HOME`*`/dbs` directory and is named `init`*`ORACLE_SID`*`.ora` (for example, `initmydb.ora`).
   - **Windows systems:** This file is in the *`ORACLE_HOME`*`\database` directory and is named `init`*`ORACLE_SID`*`.ora` (for example, `initmydb.ora`).

5. Edit the initialization parameter file to use the new `COMPATIBLE` setting.

   For example:

   ```
   compatible=12.2.0.0.0
   ```

6. In SQL*Plus, connect as a user who has the `SYSDBA` administrative privilege, and then restart the database.

   For example:

```
CONNECT /AS SYSDBA
SHUTDOWN
STARTUP
```

If tablespace encryption is in use, then open the keystore at the database mount. The keystore must be open before you can access data in an encrypted tablespace.

For example:

```
STARTUP MOUNT;
ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY keystore_password;
ALTER DATABASE OPEN;
```

Step 2: Set the Tablespace TDE Master Encryption Key

You should ensure that you have configured the TDE master encryption key.

- Set the TDE master encryption key as follows:
  - For software TDE master encryption keys, see Step 4: Set the TDE Master Encryption Key in the Software Keystore.
  - For hardware TDE master encryption keys, see Step 4: Set the Hardware Keystore TDE Master Encryption Key.

Step 3: Create the Encrypted Tablespace

After you have set the `COMPATIBLE` initialization parameter, you are ready to create the encrypted tablespace.

- About Creating Encrypted Tablespaces
  To create an encrypted tablespace, you can use the `CREATE TABLESPACE` SQL statement.
- Creating an Encrypted Tablespace
  To create an encrypted tablespace, you must use the `CREATE TABLESPACE` statement with the `ENCRYPTION USING` clause.
- Example: Creating an Encrypted Tablespace That Uses AES192
  You can use the `CREATE TABLESPACE` SQL statement to create an encrypted tablespace.
- Example: Creating an Encrypted Tablespace That Uses the Default Algorithm
  You can use the `CREATE TABLESPACE` SQL statement to create an encrypted tablespace that uses the default algorithm.

About Creating Encrypted Tablespaces

To create an encrypted tablespace, you can use the `CREATE TABLESPACE` SQL statement.

You must have the `CREATE TABLESPACE` system privilege to create an encrypted tablespace.

You can import data into an encrypted tablespace by using Oracle Data Pump. You can also use a SQL statement such as `CREATE TABLE...AS SELECT...` or `ALTER TABLE...MOVE...` to move data into an encrypted tablespace. The `CREATE TABLE...AS SELECT...` statement creates a table from an existing table. The `ALTER TABLE...MOVE...` statement moves a table into the encrypted tablespace.

For security reasons, you cannot encrypt a tablespace with the `NO SALT` option.

You can query the `ENCRYPTED` column of the `DBA_TABLESPACES` and `USER_TABLESPACES` data dictionary views to verify if a tablespace was encrypted.

> **See Also:**
>
> *Oracle Database Reference* for more information about the `DBA_TABLESPACES` and `USER_TABLESPACES` data dictionary views

Creating an Encrypted Tablespace

To create an encrypted tablespace, you must use the `CREATE TABLESPACE` statement with the `ENCRYPTION USING` clause.

1. Log in to the database instance as a user who has been granted the `CREATE TABLESPACE` system privilege.

   In a multitenant environment, log in to the PDB. For example:

   ```
   sqlplus sec_admin@hrpdb as syskm
   Enter password: password
   Connected.
   ```

   To find the available PDBs, run the `show pdbs` command. To check the current PDB, run the `show con_name` command.

2. Run the `CREATE TABLESPACE` statement, using its encryption clauses.

   For example:

   ```
   CREATE TABLESPACE encrypt_ts
     DATAFILE '$ORACLE_HOME/dbs/encrypt_df.dbf' SIZE 1M
     ENCRYPTION USING 'AES256' ENCRYPT;
   ```

   In this specification:

   - `ENCRYPTION USING 'AES256' ENCRYPT` specifies the encryption algorithm and the key length for the encryption. The `ENCRYPT` clause encrypts the tablespace. Enclose this setting in single quotation marks (' '). The key lengths are included in the names of the algorithms. If you do not specify an encryption algorithm, then the default encryption algorithm, `AES128`, is used.

> **See Also:**
>
> - Supported Encryption and Integrity Algorithms
> - *Oracle Database SQL Language Reference*

Example: Creating an Encrypted Tablespace That Uses AES192

You can use the `CREATE TABLESPACE` SQL statement to create an encrypted tablespace.

Example 3-8 creates a tablespace called `securespace_1` that is encrypted using the `AES192` algorithm. The key length is 192 bits.

Example 3-8 Creating an Encrypted Tablespace That Uses AES192

```
CREATE TABLESPACE securespace_1
DATAFILE '/home/user/oradata/secure01.dbf'
SIZE 150M
ENCRYPTION USING 'AES192' ENCRYPT;
```

Example: Creating an Encrypted Tablespace That Uses the Default Algorithm

You can use the `CREATE TABLESPACE` SQL statement to create an encrypted tablespace that uses the default algorithm.

Example 3-9 creates a tablespace called `securespace_2`. Because no encryption algorithm is specified, the default encryption algorithm (`AES128`) is used. The key length is 128 bits.

You cannot encrypt an existing tablespace.

Example 3-9 Creating an Encrypted Tablespace That Uses the Default Algorithm

```
CREATE TABLESPACE securespace_2
DATAFILE '/home/user/oradata/secure01.dbf'
SIZE 150M
ENCRYPTION ENCRYPT;
```

Encrypting Future Tablespaces

You can configure Oracle Database to automatically encrypt future tablespaces that you will create.

- About Encrypting Future Tablespaces
  The ability to encrypt future tablespaces can help prevent data breaches in Oracle Cloud environments.
- Setting Future Tablespaces to be Encrypted
  You can set the `ENCRYPT_NEW_TABLESPACES` database initialization parameter to automatically encrypt future tablespaces that you create.

About Encrypting Future Tablespaces

The ability to encrypt future tablespaces can help prevent data breaches in Oracle Cloud environments.

The `ENCRYPT_NEW_TABLESPACES` database initialization parameter controls how future databases are encrypted.

You can create and run an Oracle database completely in Oracle Cloud. Because this configuration hosts the customer's data in the Cloud, Oracle recommends that

you enable encryption as much as possible. A long-term goal is to encrypt all data in Oracle Cloud. Alternatively, you can have the database both in the Cloud and on premises.

In an Oracle Cloud environment, the following scenarios may occur when you create encrypted tablespaces in Oracle Cloud and on-premises environments:

- You create a test database in Oracle Cloud and the tablespaces were encrypted by using when the `ENCRYPT_NEW_TABLESPACE` parameter has been set to automatically create the Cloud database as encrypted. However, you may not have the intention or even an Advanced Security Option license to bring the encrypted database back on premises.
- You create a hybrid definer's rights environment where the primary database is on premises and the standby database is on Oracle Cloud. If a switchover operation takes place, then the new primary is on Oracle Cloud. If a new tablespace is transparently encrypted, then a similar scenario to the first item in this list may occur. For example, suppose you do not have an Advanced Security Option (ASO) license, and you have an automatically encrypted tablespace in the Oracle Cloud. The standby database on premises is also automatically encrypted. In this case, because you do not have an ASO license, you cannot use the standby database. To remedy this problem, set the `ENCRYPT_NEW_TABLESPACES` to DDL, which prevents the encryption of the tablespace in Oracle Cloud.

Setting Future Tablespaces to be Encrypted

You can set the `ENCRYPT_NEW_TABLESPACES` database initialization parameter to automatically encrypt future tablespaces that you create.

- In SQL*Plus, enter the following `ALTER SYSTEM` statement:

  ```
  ALTER SYSTEM SET ENCRYPT_NEW_TABLESPACES = value;
  ```

  In this specification, *value* can be:

  - `CLOUD_ONLY` transparently encrypts the tablespace in the Cloud using the `AES128` algorithm if you do not specify the `ENCRYPTION` clause of the `CREATE TABLESPACE` SQL statement. It applies only to an Oracle Cloud environment. If you create the tablespace on premise, then it will follow the `CREATE TABLESPACE` statement specification that you enter. For example, if you omit the `ENCRYPTION` clause, then the tablespace is created unencrypted. If you include this clause and use a different algorithm, then the tablespace will use that algorithm. `CLOUD_ONLY` is the default.
  - `ALWAYS` automatically encrypts the tablespace using the `AES128` algorithm if you omit the `ENCRYPTION` clause of `CREATE TABLESPACE`, for both the Cloud and premises scenarios.

    If you do provide the `ENCRYPTION` clause, however, the algorithm that you specify takes precedence over `AES128`.
  - `DDL` encrypts the tablespace using the specified setting of the `ENCRYPTION` clause of `CREATE TABLESPACE`, for both Oracle Cloud and on-premises environments.

Related Topics

- ENCRYPT_NEW_TABLESPACES

Encrypted Sensitive Credential Data in the Data Dictionary

You can encrypt sensitive credential data in the `SYS.LINK$` and `SYS.SCHEDULER$_CREDENTIAL` system tables.

By default, the credential data in the `SYS.LINK$` and `SYS.SCHEDULER$_CREDENTIAL` system tables is obfuscated. However, because of the availability of many types of de-obfuscation algorithms, Oracle recommends that you encrypt this sensitive credential data. To check the status the data dictionary credentials, you can query the `DICTIONARY_CREDENTIALS_ENCRYPT` data dictionary view.

The encryption of sensitive credential data in these two system tables uses Transparent Data Encryption. Encryption of credential data uses the AES256 algorithm. To encrypt credential data, you do not need an Oracle Advanced Security Option license, but you must be granted the `SYSKM` administrative privilege and the database must have an open keystore.

Related Topics

- *Oracle Database Security Guide*

Encryption Conversions for Existing Offline Tablespaces

You can perform offline encryption conversions by using the `ALTER TABLESPACE` SQL statement `OFFLINE`, `ENCRYPT`, and `DECRYPT` clauses.

- About Encryption Conversion for Existing Offline Tablespaces
  You can encrypt or decrypt an existing data file of a user tablespace when the tablespace is offline or when the database is not open.
- Encrypting an Existing User-Defined Tablespace with Offline Conversion
  To encrypt an existing tablespace with offline conversion, you can use the `ALTER TABLESPACE` SQL statement with the `OFFLINE` and `ENCRYPT` clauses.
- Decrypting an Existing Tablespace with Offline Conversion
  To decrypt an existing tablespace with offline conversion, you can use the `ALTER TABLESPACE` SQL statement with the `OFFLINE` and `DECRYPT` clauses.

About Encryption Conversion for Existing Offline Tablespaces

You can encrypt or decrypt an existing data file of a user tablespace when the tablespace is offline or when the database is not open.

Use the offline encryption method if you do not plan to change the compatibility of your databases from Oracle Database 11c release 2 (11.2) or Oracle Database 12c release 1 (12.1) to release 18c, which is irreversible. The offline encryption method is also useful if you want to quickly make use of Transparent Data Encryption before you upgrade this database to release 18c. You can both encrypt and decrypt offline tablespaces.

Note the following:

- If you want to encrypt the Oracle Database-supplied tablespaces (`SYSTEM`, `SYSAUX`, and `UNDO`) using the offline conversion method, then you must use the method that is described in Encrypting an Existing Database with Offline Conversion.

- You can use the online method to rekey a tablespace that was previously encrypted with the offline method.
- If you have configured Oracle Data Guard, you can minimize downtime by encrypting the tablespaces on the standby first, switching over to the primary, and then encrypting the tablespaces on the primary.
- You can use the `USING ... ENCRYPT` clause to specify an encryption algorithm. Supported algorithms include `AES128`, `AES192`, `AES256`, and others, such as `ARIA` and `GOST`. To check the encryption key, query the `ENCRYPTIONALG` column in the `V$DATABASE_KEY_INFO` view.
- You can use the `ALTER TABLESPACE` statement to convert offline tablespaces in parallel by using multiple foreground sessions to encrypt different data files.
- If you are using Oracle Data Guard, you can minimize the downtime by encrypting the tablespaces on the standby first, switching over, and then encrypting the tablespaces on the original primary next.
- For Oracle Database 11g release 2 (11.2.0.4) and Oracle Database 12c release 1 (12.1.0.2), you cannot perform an offline encryption of the `SYSTEM` and `SYSAUX` tablespaces. Also, Oracle does not recommend encrypting offline the `UNDO` tablespace in these releases. Doing so prevents the keystore from being closed, and this prevents the database from functioning. In addition, encrypting the `UNDO` tablespace while the database is offline is not necessary because all undo records that are associated with any encrypted tablespaces are already automatically encrypted in the `UNDO` tablespace. If you want to encrypt the `TEMP` tablespace, you must drop and then recreate it as encrypted.

Related Topics

- Supported Encryption and Integrity Algorithms
  The supported Advanced Encryption Standard cipher keys, including tablespace and database encryption keys, can be either 128, 192, or 256 bits long. Tablespace and database encryption use the 128–bit length cipher key.

Encrypting an Existing User-Defined Tablespace with Offline Conversion

To encrypt an existing tablespace with offline conversion, you can use the `ALTER TABLESPACE` SQL statement with the `OFFLINE` and `ENCRYPT` clauses.

The procedure that is described in this section applies to the case where you want to encrypt individual user-created tablespaces within a database. These tablespaces can be encrypted offline. However, the Oracle Database-supplied `SYSTEM` and `UNDO` tablespaces cannot be brought offline. If you want to encrypt the tablespaces offline, then you must use the method that is described in Encrypting an Existing Database with Offline Conversion.

1. Connect as a user who has the `SYSDBA` administrative privilege.

   For example:

   ```
   sqlplus sec_admin as sysdba
   Enter password: password
   ```

   You must have the `SYSDBA` administrative privilege if you plan to encrypt the `SYSTEM` and `SYSAUX` tablespaces. Otherwise, connect with the `SYSKM` administrative privilege.

2. Bring the tablespace offline.

```
ALTER TABLESPACE users OFFLINE NORMAL;
```

3. Back up the tablespace.

   The offline conversion method does not use auxiliary disk space or files, and it operates directly in-place to the data files. Therefore, you should perform a full backup of the user tablespace before converting it offline.

4. As a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege, open the software keystore.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY
   software_keystore_password;
   ```

5. Encrypt the tablespace.

   For example, to encrypt an entire tablespace, include its data files:

   ```
   ALTER TABLESPACE users1 ENCRYPTION OFFLINE ENCRYPT;
   ```

   This example encrypts the tablespace with the default encryption algorithm, `AES128`. To use a different encryption algorithm, enter a statement similar to the following:

   ```
   ALTER TABLESPACE users2 ENCRYPTION OFFLINE USING 'AES256' ENCRYPT;
   ```

   To encrypt individual data files within a tablespace, use the `ALTER DATABASE DATAFILE` SQL statement. For example, to encrypt the data files `user_01.dbf` and `user_02.dbf`:

   ```
   ALTER DATABASE DATAFILE 'user_01.dbf' ENCRYPT;
   ALTER DATABASE DATAFILE 'user_02.dbf' ENCRYPT;
   ```

   In the same database session, these statements encrypt each of the data files in sequence, one after another. If you execute each statement in its own database session, then they will be executed in parallel.

   If the encryption process is interrupted, then rerun the `ALTER TABLESPACE` statement. The kinds of errors that you can expect in an interruption are general errors, such as file system or storage file system errors. The data files within the tablespace should be consistently encrypted. For example, suppose you offline a tablespace that has 10 files but for some reason, the encryption only completes for nine of the files, leaving one decrypted. Although it is possible to bring the tablespace back online with such inconsistent encryption if the `COMPATIBLE` parameter is set to `12.2.0.0` or higher, then it is not recommended to leave the tablespace in this state. If `COMPATIBLE` is less than `12.2.0.0`, then it is not possible to bring the tablespace online if the encryption property is inconsistent across the data files.

6. Bring the tablespace back online or open the database.

   - To bring the tablespace back online:

     ```
     ALTER TABLESPACE users ONLINE;
     ```

   - To open a database in a non-multitenant environment:

     ```
     ALTER DATABASE OPEN
     ```

   - In a multitenant environment, you can encrypt a data file or tablespace with the offline method if the root is open and the PDB is not open. For example, for a PDB named `hr_pdb`:

```
ALTER PLUGGABLE DATABASE hr_pdb OPEN
```

7. Perform a full backup of the converted tablespace.

Related Topics

- Opening a Software Keystore
  To open a software keystore, you must use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEYSTORE OPEN` clause.
- Supported Encryption and Integrity Algorithms
  The supported Advanced Encryption Standard cipher keys, including tablespace and database encryption keys, can be either 128, 192, or 256 bits long. Tablespace and database encryption use the 128–bit length cipher key.

Decrypting an Existing Tablespace with Offline Conversion

To decrypt an existing tablespace with offline conversion, you can use the `ALTER TABLESPACE` SQL statement with the `OFFLINE` and `DECRYPT` clauses.

1. Connect as a user who has the `SYSDBA` administrative privilege.

   For example:

   ```
   sqlplus sec_admin as sysdba
   Enter password: password
   ```

   You must have the `SYSDBA` administrative privilege if you plan to decrypt the `SYSTEM` and `SYSAUX` tablespaces. Otherwise, connect with the `SYSKM` administrative privilege.

2. Bring the tablespace offline.

   ```
   ALTER TABLESPACE users OFFLINE NORMAL;
   ```

3. As a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege, open the keystore.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY
   software_keystore_password;
   ```

4. Run the `ALTER TABLESPACE` SQL statement to perform the decryption.

   For example, for a tablespace called `users`:

   ```
   ALTER TABLESPACE users ENCRYPTION OFFLINE DECRYPT;
   ```

   If the decryption process is interrupted, then rerun the `ALTER TABLESPACE` statement. The kinds of errors that you can expect in an interruption are general errors, such as file system or storage file system errors. The data files within the tablespace should be consistently decrypted. For example, suppose you offline a tablespace that has 10 files but for some reason, the decryption only completes for nine of the files, leaving one encrypted. Although it is possible to bring the tablespace back online with such inconsistent decryption if the `COMPATIBLE` parameter is set to `12.2.0.0` or higher, then it is not recommended to leave the tablespace in this state. If `COMPATIBLE` is less than `12.2.0.0`, then it is not possible to bring the tablespace online if the encryption property is inconsistent across the data files.

5. Bring the tablespace online.

```
ALTER TABLESPACE users ONLINE;
```

Related Topics

- Opening a Software Keystore
  To open a software keystore, you must use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEYSTORE OPEN` clause.

Encryption Conversions for Existing Online Tablespaces

You can encrypt and decrypt an online existing tablespace by using the `ALTER TABLESPACE` SQL statement with the `ONLINE` and `ENCRYPT` or `DECRYPT` clauses.

- About Encryption Conversion for Existing Online Tablespaces
  You can encrypt, decrypt, or rekey existing user tablespaces, and the `SYSTEM`, `SYSAUX`, and `UNDO` tablespace when they are online.
- Encrypting an Existing Tablespace with Online Conversion
  To encrypt an existing tablespace with online conversion, use `ALTER TABLESPACE` with the `ONLINE` and `ENCRYPT` clauses.
- Rekeying an Existing Tablespace with Online Conversion
  To rekey an existing tablespace that is online, you can use the `REKEY` clause of the `ALTER TABLESPACE` SQL statement.
- Decrypting an Existing Tablespace with Online Conversion
  To decrypt an existing tablespace with online conversion, you can use the `ALTER TABLESPACE` SQL statement with `DECRYPT` clause.
- Finishing an Interrupted Online Encryption Conversion
  If an online encryption process is interrupted, then you can complete the conversion by rerunning the `ALTER TABLESPACE` statement using the `FINISH` clause.

About Encryption Conversion for Existing Online Tablespaces

You can encrypt, decrypt, or rekey existing user tablespaces, and the `SYSTEM`, `SYSAUX`, and `UNDO` tablespace when they are online.

However, you cannot encrypt, decrypt, or rekey a temporary tablespace online.

An online tablespace can be created by using the `ONLINE` clause of the `CREATE TABLESPACE` SQL statement. When you encrypt or rekey a tablespace online, the tablespace will have its own independent encryption keys and algorithms.

Note the following:

- If an offline tablespace has been encrypted, then you can rekey it online to use a different algorithm.
- You can encrypt multiple tablespaces online in parallel by using multiple foreground sessions to encrypt different tablespaces. Within each tablespace, the data files are encrypted sequentially.
- If the conversion is interrupted, then you can resume the process by issuing the `FINISH` clause of the `ALTER TABLESPACE` SQL statement.
- A redo log is generated for each online tablespace conversion.

- Do not encrypt the `SYSTEM` and `UNDO` tablespaces concurrently with other tablespaces.
- You cannot use the transportable tablespace feature with Oracle Data Pump while you are encrypting a tablespace.
- You cannot run the `ALTER TABLESPACE` statement concurrently with the following features:
  - `ADMINSTER KEY MANAGEMENT SET KEY` SQL statement
  - `FLASHBACK DATABASE` SQL statement
- If you are using Oracle-managed files for the data files, then the encryption process rekeys the data files that are associated with the tablespace and then copies or moves them to the default Oracle-managed files location.
- You can add new files to the tablespace after you have encrypted it. Oracle Database reformats the new file with the new encryption key. Blocks will be encrypted using the new key.
- Previous operations that took place in the root or the PDB may require the control files to be cross-checked against the data dictionary before you can begin the online conversion process. An `ORA-241 operation disallowed: control file is not yet checked against data dictionary` error may occur. To resolve this problem, restart the root or PDB, and then try issuing the online conversion commands again.
- For security reasons, once online conversion processes a data file, Oracle will zero out the original data file before deletion. This prevents the database from leaving ghost data on disk sectors. However, there is a known limitation that can occur if you are performing an online tablespace conversion at the same time that Oracle Recovery Manager (Oracle RMAN) is validating files. The online tablespace conversion processes each file one at a time. If Oracle RMAN is validating a file at the same time that it is being processed by the online tablespace conversion, then Oracle RMAN could report a corruption problem (`ORA-01578: ORACLE data block corrupted (file # , block # )`). It does this because it sees the blocks that comprise the file as zero. This is a false alarm and you can ignore the error. If this occurs, then try running the Oracle RMAN validation process again.

Related Topics

- Supported Encryption and Integrity Algorithms
  The supported Advanced Encryption Standard cipher keys, including tablespace and database encryption keys, can be either 128, 192, or 256 bits long. Tablespace and database encryption use the 128–bit length cipher key.

Encrypting an Existing Tablespace with Online Conversion

To encrypt an existing tablespace with online conversion, use `ALTER TABLESPACE` with the `ONLINE` and `ENCRYPT` clauses.

1. Connect as a user who has the `SYSDBA` administrative privilege.

   For example:

   ```
   sqlplus sec_admin as sysdba
   Enter password: password
   ```

You must have the SYSDBA administrative privilege if you plan to encrypt the SYSTEM and SYSAUX tablespaces. Otherwise, connect with the SYSKM administrative privilege.

2. Ensure that the COMPATIBLE initialization parameter is set to 12.2.0.0.

   You can use the SHOW PARAMETER command to check the current setting of a parameter.

3. Ensure that the database is open in read-write mode.

   You can query the STATUS column of the V$INSTANCE dynamic view to find if a database is open and the OPEN_MODE column of the V$DATABASE view to find if it in read-write mode.

4. If necessary, open the database in read-write mode.

   ```
   ALTER DATABASE OPEN READ WRITE;
   ```

5. Ensure that the auxiliary space is at least the same size as the largest data file of this tablespace.

   This size requirement is because Oracle Database performs the conversion one file at a time. For example, if the largest data file of the tablespace is 32 GB, then ensure that you have 32 GB of auxiliary space. To find the space used by a data file, query the BYTES or BLOCKS column of the V$DATAFILE dynamic performance view.

6. Create and open a master encryption key.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT CREATE KEYSTORE 'keystore_location' IDENTIFIED BY
   software_keystore_password;
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY
   software_keystore_password;
   ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY software_keystore_password
   WITH BACKUP;
   ```

7. Run the ALTER TABLESPACE statement using the ENCRYPTION and ENCRYPT clauses to perform the encryption.

   For example, for a non-Oracle managed files tablespace named users:

   ```
   ALTER TABLESPACE users ENCRYPTION ONLINE USING 'AES192' ENCRYPT
   FILE_NAME_CONVERT = ('users.dbf', 'users_enc.dbf');
   ```

   In this example:

   - ENCRYPTION ONLINE USING 'AES192' ENCRYPT sets the statement to encrypt the tablespace users while it is online and assigns it the AES192 encryption algorithm. If you omit the USING *algorithm* clause, then the default algorithm, AES128, is used. For the SYSTEM and UNDO tablespaces, you can use the ENCRYPT clause to encrypt the tablespace, but you cannot specify an encryption algorithm because they must be encrypted with the existing database key the first time. After encrypting the tablespace, use the REKEY clause to specify the algorithm.
   - FILE_NAME_CONVERT specifies one or more pairs of data files that are associated with the tablespace. The first name in the pair is an existing data file, and the second name is for the encrypted version of this data file, which will be created after the ALTER TABLESPACE statement successfully executes. If

the tablespace has more than one data file, then you must process them all in this statement. Note the following:

- Separate each file name with a comma, including multiple pairs of files. For example:

  ```
  FILE_NAME_CONVERT = ('users1.dbf', 'users1_enc.dbf', 'users2.dbf',
  'users2_enc.dbf')
  ```

- You can specify directory paths in the `FILE_NAME_CONVERT` clause. For example, the following clause converts and moves the matching files of the tablespace from the `dbs` directory to the `dbs/enc` directory:

  ```
  FILE_NAME_CONVERT = ('dbs', 'dbs/enc')
  ```

- The `FILE_NAME_CONVERT` clause recognizes patterns. The following example converts the data files `users_1.dbf` and `users_2.dbf` to `users_enc1.dbf` and `users_enc2.dbf`:

  ```
  FILE_NAME_CONVERT = ('users', 'users_enc')
  ```

- In an Oracle Data Guard environment, include the name of the standby database data file in the `FILE_NAME_CONVERT` settings.

- If you are using Oracle-managed file mode, then the new file name is internally assigned, so this file name should not affect your site's file-naming standards. If you are using non-Oracle-managed file mode and if you omit the `FILE_NAME_CONVERT` clause, then Oracle Database internally assigns an auxiliary file name, and then later renames it back to the original name. This enables the encryption process to use the name that you had originally given the file to be encrypted. The renaming operation is effectively creating another copy of the file, hence it is slower than explicitly including the `FILE_NAME_CONVERT` clause. For better performance, include the `FILE_NAME_CONVERT` clause.

- You can find the data files for a tablespace by querying the `V$DATAFILE` or `V$DATAFILE_HEADER` dynamic views.

By default, data files are in the `$ORACLE_HOME/dbs` directory. If the data files are located there, then you do not have to specify a path.

After you complete the conversion, you can check the encryption status by querying the `STATUS` column of the `V$ENCRYPTED_TABLESPACES` dynamic view. The `ENCRYPTIONALG` column of this view shows the encryption algorithm that is used. If the conversion process was interrupted, then you can resume it by running `ALTER TABLESPACE` with the `FINISH` clause. For example, if the primary data file converts but the standby data file does not, then you can run `ALTER TABLESPACE ... FINISH` on the standby database for the standby data files.

Related Topics

- Setting the COMPATIBLE Initialization Parameter for Tablespace Encryption
  To set the `COMPATIBLE` initialization parameter, you must edit the initialization parameter file for the database instance.
- Finishing an Interrupted Online Encryption Conversion
  If an online encryption process is interrupted, then you can complete the conversion by rerunning the `ALTER TABLESPACE` statement using the `FINISH` clause.

Rekeying an Existing Tablespace with Online Conversion

To rekey an existing tablespace that is online, you can use the `REKEY` clause of the `ALTER TABLESPACE` SQL statement.

Before you perform a rekey operation, be aware of the following:

- You cannot rekey the `TEMP` tablespace. If you want to assign a different encryption algorithm to a `TEMP` tablespace, then drop `TEMP` and recreate it with the correct encryption algorithm.
- Do not perform an online tablespace rekey operation with a master key operation concurrently. To find if any tablespaces are currently being rekeyed, issue the following query to find the rekey status of encrypted tablespaces:

  ```
  SELECT TS#,ENCRYPTIONALG,STATUS FROM V$ENCRYPTED_TABLESPACES;
  ```

  A status of `REKEYING` means that the corresponding tablespace is still being rekeyed. Do not rekey the master key while this status is in effect.

To rekey an existing tablespace with online conversion:

1. Connect as a user who has the `SYSDBA` administrative privilege.

   For example:

   ```
   sqlplus sec_admin as sysdba
   Enter password: password
   ```

   You must have the `SYSDBA` administrative privilege if you plan to rekey the `SYSTEM` and `SYSAUX` tablespaces. Otherwise, connect with the `SYSKM` administrative privilege.
2. Ensure that the following requirements are met:
   - The `COMPATIBLE` initialization parameter is set to 12.2.0.0.
   - The database is open and in read-write mode.
   - A master encryption key has been created and is open.
3. Query the `KEY_VERSION` and `STATUS` columns of the `V$ENCRYPTED_TABLESPACES` dynamic view to find the current status of the encryption algorithm used by the master encryption key.
4. Perform the rekey operation, based on the status returned by the `V$ENCRYPTED_TABLESPACES` dynamic view:
   - If the key version status of the tablespace is `NORMAL`, then specify the new algorithm of the online tablespace rekey.

     For example:

     ```
     ALTER TABLESPACE users ENCRYPTION USING 'AES192' REKEY FILE_NAME_CONVERT
     = ('users.dbf', 'users_enc.dbf');
     ```
   - If the key version status is `ENCRYPTING`, `DECRYPTING`, or `REKEYING`, then use the `FINISH` clause.

     For example:

     ```
     ALTER TABLESPACE users ENCRYPTION ONLINE FINISH REKEY FILE_NAME_CONVERT
     = ('users.dbf', 'users_enc.dbf');
     ```

5. If the `ORA-00241 operation disallowed: control file inconsistent with data dictionary` error appears, then restart the database.

   In a multitenant environment, restart the CDB root database and then retry Step 4.

If the conversion process was interrupted, then you can resume it by running `ALTER TABLESPACE` with the `FINISH` clause.

Related Topics

- Encrypting an Existing Tablespace with Online Conversion
  To encrypt an existing tablespace with online conversion, use `ALTER TABLESPACE` with the `ONLINE` and `ENCRYPT` clauses.
- About Encryption Conversion for Existing Online Tablespaces
  You can encrypt, decrypt, or rekey existing user tablespaces, and the `SYSTEM`, `SYSAUX`, and `UNDO` tablespace when they are online.
- Finishing an Interrupted Online Encryption Conversion
  If an online encryption process is interrupted, then you can complete the conversion by rerunning the `ALTER TABLESPACE` statement using the `FINISH` clause.

Decrypting an Existing Tablespace with Online Conversion

To decrypt an existing tablespace with online conversion, you can use the `ALTER TABLESPACE` SQL statement with `DECRYPT` clause.

1. Connect as a user who has the `SYSDBA` administrative privilege.

   For example:

   ```
   sqlplus sec_admin as sysdba
   Enter password: password
   ```

   You must have the `SYSDBA` administrative privilege if you plan to decrypt the `SYSTEM` and `SYSAUX` tablespaces. Otherwise, connect with the `SYSKM` administrative privilege.
2. Ensure that the following requirements are met:
   - The `COMPATIBLE` initialization parameter is set to 12.2.0.0.
   - The database is open and in read-write mode.
   - A master encryption key has been created and is open.
   - There is enough auxiliary space to complete the decryption.
3. Run the `ALTER TABLESPACE` SQL statement with the `DECRYPT` clause.

   For example:

   ```
   ALTER TABLESPACE users ENCRYPTION ONLINE DECRYPT FILE_NAME_CONVERT =
   ('users_enc.dbf', 'users.dbf');
   ```

   In this specification:

   - When you specify the files to decrypt, enter them in the reverse order in which they were originally encrypted. That is, first enter the name of the encrypted file (`users_enc.dbf`), followed by the data file (`users.dbf`).
   - Do not provide an algorithm key for the decryption.

If the conversion process was interrupted, then you can resume it by running ALTER TABLESPACE with the FINISH clause.

Related Topics

- Encrypting an Existing Tablespace with Online Conversion
  To encrypt an existing tablespace with online conversion, use ALTER TABLESPACE with the ONLINE and ENCRYPT clauses.
- Finishing an Interrupted Online Encryption Conversion
  If an online encryption process is interrupted, then you can complete the conversion by rerunning the ALTER TABLESPACE statement using the FINISH clause.

Finishing an Interrupted Online Encryption Conversion

If an online encryption process is interrupted, then you can complete the conversion by rerunning the ALTER TABLESPACE statement using the FINISH clause.

An interrupted encryption process (encryption, rekey, or decryption) can be, for example, an ORA-28425: missing a valid FILE_NAME_CONVERT clause error in the FILE_NAME_CONVERT clause of the ALTER TABLESPACE SQL statement. Other examples of interrupted processes are if the conversion skips a data file, which can happen if there is an error when an Oracle DataBase WRiter (DBWR) process offlines a data file, or if there is not enough space for the auxiliary file. The tablespace should be operational even if you do not rerun the ALTER TABLESPACE statement with the FINISH clause.

1. Query the V$ENCRYPTED_TABLESPACES to check the STATUS column for the tablespace.

   If the STATUS column reports ENCRYPTING, DECRYPTING, or REKEYING, then re-run the ALTER TABLESPACE statement with the FINISH clause, as described in this procedure. If the STATUS reports NORMAL, then you can rerun ALTER TABLESPACE without the FINISH clause.

   You can find the tablespace name that matches the TS# and TABLESPACE_NAME columns by querying the V$DATAFILE_HEADER view.

2. If necessary query the following additional views to find information about the tablespace whose online conversion was interrupted:
   - DBA_TABLESPACES to find if the STATUS of the tablespace indicates if it is online or offline.
   - V$ENCRYPTED_TABLESPACES to find if the STATUS of the tablespace indicates if it is encrypted, and what the KEY_VERSION of the encryption key is.
   - V$DATAFILE and V$DATAFILE_HEADER to find the data files that are associated with a tablespace.
3. Run the ALTER TABLESPACE statement using the FINISH clause.

   Examples are as follows:

   - For an encryption operation:

     ```
     ALTER TABLESPACE users ENCRYPTION ONLINE FINISH ENCRYPT
     FILE_NAME_CONVERT = ('users.dbf', 'users_enc.dbf');
     ```

   - For a decryption operation:

```
ALTER TABLESPACE users ENCRYPTION ONLINE FINISH DECRYPT
FILE_NAME_CONVERT = ('users_enc.dbf', 'users.dbf');
```

Note the order in which the files are specified: first, the name of the encrypted file, and then the name of the data file. (In the encryption operation, the name of the data file is specified first, followed by the name of the encrypted file.)

- For a rekey operation:

```
ALTER TABLESPACE users ENCRYPTION ONLINE FINISH REKEY FILE_NAME_CONVERT
= ('users.dbf', 'users_enc.dbf');
```

You cannot specify an algorithm when you use the `FINISH` clause in an `ALTER TABLESPACE` statement.

4. To check the conversion, query the `STATUS` column of the `V$ENCRYPTED_TABLESPACES` view.

   The status should be `NORMAL`. In an Oracle Data Guard environment, if the database does not have `NORMAL` as the `STATUS`, then run the `ALTER TABLESPACE ... FINISH` statement on the primary or the standby data file that did not successfully convert.

Encryption Conversions for Existing Databases

You can encrypt both offline and online databases.

- About Encryption Conversions for Existing Databases
  The encryption conversion of an entire database encrypts all tablespaces, including the Oracle-supplied `SYSTEM`, `SYSAUX`, `UNDO`, and `TEMP` tablespaces.
- Encrypting an Existing Database with Offline Conversion
  When you encrypt an existing database with offline conversion, for the Oracle-managed tablespaces, you do not specify an encryption algorithm.
- Encrypting an Existing Database with Online Conversion
  When you encrypt an existing database with online conversion, you do not specify an encryption algorithm.

About Encryption Conversions for Existing Databases

The encryption conversion of an entire database encrypts all tablespaces, including the Oracle-supplied `SYSTEM`, `SYSAUX`, `UNDO`, and `TEMP` tablespaces.

Note the following:

- To perform the encryption, you can use the offline and online functionality of the tablespace encryption conversions.
- You can encrypt any or all of the Oracle-supplied tablespaces, and in any order. The encryption of the Oracle-supplied tablespaces has no impact on the encryption of user-created tablespaces.
- When you encrypt the Oracle-supplied tablespaces, Oracle Database prevents the keystore from being closed.
- You cannot encrypt an existing temporary tablespace, but you can drop the existing temporary tablespace and then recreate it as encrypted.

- The UNDO and TEMP metadata that is generated from sensitive data in an encrypted tablespace is already automatically encrypted. Therefore, encrypting UNDO and TEMP is optional.
- Oracle recommends that you encrypt the Oracle-supplied tablespaces by using the default tablespace encryption algorithm, AES128. However, you can rekey any of these tablespaces afterwards to use a different encryption algorithm if you want. (To find the current encryption key for the current database, you can query the V$DATABASE_KEY_INFO dynamic view.)
- The performance effect of encrypting all the tablespaces in a database depends on the workload and platform. Many modern CPUs provide built-in hardware acceleration, which results in a minimal performance impact.
- In a multitenant environment, you can encrypt any tablespaces in any pluggable databases (PDBs), including the Oracle-supplied tablespaces. However, the keystore in the CDB root must be open at all times so that a PDB can open its keystore. You can check the status of whether a keystore is open by querying the STATUS column of the V$ENCRYPTION_WALLET view

Encrypting an Existing Database with Offline Conversion

When you encrypt an existing database with offline conversion, for the Oracle-managed tablespaces, you do not specify an encryption algorithm.

1. Connect as a user who has the SYSDBA administrative privilege.

   For example:

   ```
   sqlplus sec_admin as sysdba
   Enter password: password
   ```

   You must have the SYSDBA administrative privilege to encrypt the SYSTEM and SYSAUX tablespaces.

2. Mount the database.

   ```
   STARTUP MOUNT
   ```

3. Open the keystore.

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY keystore_password;
   ```

4. Run the ALTER TABLESPACE SQL statement to encrypt the SYSTEM, SYSAUX, and UNDO tablespaces. Do not encrypt the SYSTEM tablespace concurrently with the encryption of other tablespaces.

   For example, to encrypt the SYSTEM tablespace:

   ```
   ALTER TABLESPACE SYSTEM ENCRYPTION OFFLINE ENCRYPT;
   ```

5. Open the database.

   For example, to open the database in read/write mode:

   ```
   ALTER DATABASE OPEN READ WRITE;
   ```

6. Run the ALTER TABLESPACE SQL statement to encrypt other user tablespaces.

   Alternatively, you can proceed to the next step and open the database first, and then perform the steps described in Encrypting an Existing User-Defined Tablespace with Offline Conversion.

7. Open the database.

```
ALTER DATABASE OPEN;
```

After you have encrypted the tablespace, if you want to use a different encryption algorithm (change the TDE master encryption key) for the SYSTEM, SYSAUX, and UNDO tablespaces, then you must use online conversion. In addition to AES128, supported encryption algorithms are AES192 and AES256, in addition to other algorithms such as ARIA and GOST.

Related Topics

- Changing the TDE Master Encryption Key for a Tablespace
  You can use the ENCRYPT and REKEY clauses of the ALTER TABLESPACE statement to encrypt a tablespace.
- Supported Encryption and Integrity Algorithms
  The supported Advanced Encryption Standard cipher keys, including tablespace and database encryption keys, can be either 128, 192, or 256 bits long. Tablespace and database encryption use the 128–bit length cipher key.

Encrypting an Existing Database with Online Conversion

When you encrypt an existing database with online conversion, you do not specify an encryption algorithm.

The reason that you do not need to specify an encryption algorithm the first time you perform the encryption is that the tablespaces that you must use to encrypt the database are automatically encrypted with the database key. If you want to change the algorithm, then you can issue the ALTER TABLESPACE ENCRYPTION REKEY SQL statement after the initial encryption.

1. Perform the following tasks, which are described in Encrypting an Existing Tablespace with Online Conversion:
   a. Connect as a user who has been granted the SYSDBA administrative privilege.
   b. Ensure that the COMPATIBLE parameter is set to 12.2.0.0.
   c. Ensure that the database is open in read-write mode.
   d. Ensure that you have enough auxiliary space to complete the encryption.
   e. Back up the tablespaces that you must encrypt.
   f. Open the keystore.
2. Run the ALTER TABLESPACE SQL statement to encrypt the SYSTEM, SYSAUX, and UNDO tablespaces. Do not specify an algorithm, and do not encrypt the SYSTEM tablespace concurrently with the encryption of other tablespaces.

   For example, to encrypt the SYSTEM tablespace:

   ```
   ALTER TABLESPACE SYSTEM ENCRYPTION ONLINE ENCRYPT
   FILE_NAME_CONVERT=('system01.dbf','system01_enc.dbf');
   ```

3. For a temporary tablespace, drop it and then recreate it as encrypted. Do not specify an algorithm.

   For example, for a user-created tablespace:

   ```
   DROP TABLESPACE temp_01;
   CREATE TEMPORARY TABLESPACE temp_01
   TEMPFILE 'temp01.dbf' SIZE 5M AUTOEXTEND ON
   ENCRYPTION ENCRYPT;
   ```

You cannot drop the default TEMP tablespace. You must first create a new tablespace and make it the default before you can drop TEMP.

For example:

```
CREATE TEMPORARY TABLESPACE temp_01
TEMPFILE 'temp01.dbf' SIZE 5M AUTOEXTEND ON
ENCRYPTION ENCRYPT;

ALTER DATABASE DEFAULT TEMPORARY TABLESPACE temp_01;

DROP TABLESPACE TEMP;
```

Related Topics

- Changing the TDE Master Encryption Key for a Tablespace
  You can use the ENCRYPT and REKEY clauses of the ALTER TABLESPACE statement to encrypt a tablespace.

Transparent Data Encryption Data Dynamic and Data Dictionary Views

You can query a set of dynamic and data dictionary views to find more information about Transparent Data Encryption (TDE) data.

Table 3-4 describes these dynamic and data dictionary views.

Table 3-4 Transparent Data Encryption Related Views

View	Description
ALL_ENCRYPTED_COLUMNS	Displays encryption information about encrypted columns in the tables accessible to the current user
DBA_ENCRYPTED_COLUMNS	Displays encryption information for all of the encrypted columns in the database
USER_ENCRYPTED_COLUMNS	Displays encryption information for encrypted table columns in the current user's schema
DBA_TABLESPACE_USAGE_METRICS	Describes tablespace usage metrics for all types of tablespaces, including permanent, temporary, and undo tablespaces
V$CLIENT_SECRETS	Lists the properties of the strings (secrets) that were stored in the keystore for various features (clients). In a multitenant environment, when you query this view in a PDB, then it displays information about keys that were created or activated for the current PDB. If you query this view in the root, then it displays this information about keys for all of the PDBs.
V$DATABASE_KEY_INFO	Displays information about the default encryption key that is used for the current database. The default is AES128.
V$ENCRYPTED_TABLESPACES	Displays information about the tablespaces that are encrypted

Table 3-4 (Cont.) Transparent Data Encryption Related Views

View	Description
V$ENCRYPTION_KEYS	When used with keys that have been rekeyed with the ADMINISTER KEY MANAGEMENT statement, displays information about the TDE master encryption keys. In a multitenant environment, when you query this view in a PDB, it displays information about keys that were created or activated for the current PDB. If you query this view in the root, it displays this information about keys for all of the PDBs.
V$ENCRYPTION_WALLET	Displays information on the status of the keystore and the keystore location for TDE
V$WALLET	Displays metadata information for a PKI certificate, which can be used as a master encryption key for TDE

See Also:

Oracle Database Reference for detailed information about these views

4

Managing the Keystore and the Master Encryption Key

You can modify settings for the keystore and TDE master encryption key, and store Oracle Database and store Oracle GoldenGate secrets in a keystore.

- Managing the Keystore
  You can perform maintenance activities on keystores such as changing passwords, and backing up, merging, and moving keystores.
- Managing the TDE Master Encryption Key
  You can manage the TDE master encryption key in several ways.
- Storing Oracle Database Secrets
  Secrets are data that support internal Oracle Database features that integrate external clients such as Oracle GoldenGate into the database.
- Storing Oracle GoldenGate Secrets in a Keystore
  You can store Oracle GoldenGate secrets in Transparent Data Encryption keystores.

Managing the Keystore

You can perform maintenance activities on keystores such as changing passwords, and backing up, merging, and moving keystores.

- Performing Operations That Require a Keystore Password
  Many `ADMINISTER KEY MANAGEMENT` operations require access to a keystore password, for both software and hardware keystores.
- Changing the Password of a Software Keystore
  Oracle Database enables you to easily change password-protected software keystore passwords.
- Changing the Password of a Hardware Keystore
  To change the password of a hardware keystore, you must use the `ADMINISTER KEY MANAGEMENT` statement.
- Configuring an External Store for a Keystore Password
  An external store for a keystore password stores the keystore password in a centrally accessed and managed location.
- Backing Up Password-Protected Software Keystores
  When you back up a password-protected software keystore, you can create a backup identifier string to describe the backup type.
- How the V$ENCRYPTION_WALLET View Interprets Backup Operations
  The `BACKUP` column of the `V$ENCRYPTION_WALLET` view indicates a how a copy of the keystore was created.
- Backups of the Hardware Keystore
  You cannot use Oracle Database to back up hardware keystores.

- Merging Software Keystores
  You can merge software keystores in a variety of ways.
- Moving a Software Keystore to a New Location
  You move a software keystore to a new location after you have updated the `WALLET_ROOT` parameter.
- Moving a Software Keystore Out of Automatic Storage Management
  You can use the `ADMINISTER KEY MANAGEMENT` statement to move a software keystore out Automatic Storage Management.
- Migrating Between a Software Password Keystore and a Hardware Keystore
  You can migrate between password-protected software keystores and hardware keystores.
- Migration of Keystores to and from Oracle Key Vault
  You can use Oracle Key Vault to migrate both software and hardware keystores to and from Oracle Key Vault.
- Configuring Keystores for Automatic Storage Management
  You can store a software keystore on an Automatic Storage Management (ASM) disk group.
- Closing a Keystore
  You can manually close software and hardware keystores.
- Backup and Recovery of Encrypted Data
  For software keystores, you cannot access encrypted data without the TDE master encryption key.
- Dangers of Deleting Keystores
  Oracle strongly recommends that you do not delete keystores.
- Features That Are Affected by Deleted Keystores
  Some features can be adversely affected if a keystore is deleted and a TDE master encryption key residing in that keystore is later needed.

Performing Operations That Require a Keystore Password

Many `ADMINISTER KEY MANAGEMENT` operations require access to a keystore password, for both software and hardware keystores.

In some cases, a software keystore depends on an auto-login keystore before the operation can succeed. Auto-login keystores open automatically when they are configured and a key is requested. They are generally used for operations where the keystore could be closed but a database operation needs a key (for example, after the database is restarted). Because the auto-login keystore opens automatically, it can be retrieved to perform a database operation without manual intervention. However, some keystore operations that require the keystore password cannot be performed when the auto-login keystore is open. The auto-login keystore must be closed and the password-protected keystore must be opened for the keystore operations that require a password.

In a multitenant environment, the re-opening of keystores affects other PDBs. For example, an auto-login keystore in the root must be accessible by the PDBs in the CDB for this root.

You can temporarily open the keystore by including the `FORCE KEYSTORE` clause in the `ADMINISTER KEY MANAGEMENT` statement when you perform the following operations: rotating a keystore password; creating, using, rekeying, tagging, importing, exporting,

migrating, or reverse migrating encryption keys; opening or backing up keystores; adding, updating, or deleting secret keystores. In a multitenant environment, if no keystore is open in the root, then `FORCE KEYSTORE` opens the password-protected keystore in the root.

Changing the Password of a Software Keystore

Oracle Database enables you to easily change password-protected software keystore passwords.

- About Changing the Password of a Password-Protected Software Keystore
  You can only change the password for protected-protected software keystores.
- Changing the Password-Protected Software Keystore Password
  To change the password of a password-protected software keystore, you must use the `ADMINISTER KEY MANAGEMENT` statement.

About Changing the Password of a Password-Protected Software Keystore

You can only change the password for protected-protected software keystores.

You can change this password at any time, as per the security policies, compliance guidelines, and other security requirements of your site. As part of the command to change the password, you will be forced to specify the `WITH BACKUP` clause, and thus forced to make a backup of the current keystore. During the password change operation, Transparent Data Encryption operations such as encryption and decryption will continue to work normally.

You can change this password at any time. You may want to change this password if you think it was compromised.

Changing the Password-Protected Software Keystore Password

To change the password of a password-protected software keystore, you must use the `ADMINISTER KEY MANAGEMENT` statement.

1. Log in to the database instance as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.

   For example:

   ```
   sqlplus sec_admin as syskm
   Enter password: password
   Connected.
   ```

2. Change the password of the password-protected software keystore by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT ALTER KEYSTORE PASSWORD
   [FORCE KEYSTORE]
   IDENTIFIED BY
   old_password SET new_password
   [WITH BACKUP [USING 'backup_identifier']];
   ```

   In this specification:

   - `FORCE KEYSTORE` temporarily opens the password-protected keystore for this operation. You must open the keystore for this operation.

- *old_password* is the current keystore password that you want to change.
- *new_password* is the new password that you will set for the keystore.
- WITH BACKUP creates a backup of the current keystore before the password is changed. You must include this clause.
- *backup_identifier* specifies an optional identifier string for the backup that is created. The *backup_identifier* is added to the name of the backup file. Enclose *backup_identifier* in single quotation marks (' '). This identifier is appended to the named keystore file (for example, ewallet_*time_stamp*_emp_key_pwd_change.p12).

The following example backs up the current keystore and then changes the password for the keystore:

```
ADMINISTER KEY MANAGEMENT ALTER KEYSTORE PASSWORD
FORCE KEYSTORE
IDENTIFIED BY
old_password SET new_password
WITH BACKUP USING 'pwd_change';

keystore altered.
```

Changing the Password of a Hardware Keystore

To change the password of a hardware keystore, you must use the ADMINISTER KEY MANAGEMENT statement.

1. Log in to the database instance as a user who has been granted the ADMINISTER KEY MANAGEMENT or SYSKM privilege.

   For example:

   ```
   sqlplus sec_admin as syskm
   Enter password: password
   Connected.
   ```

2. Close the hardware keystore.

   For example, for an HSM:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE
   IDENTIFIED BY "psmith:password";
   ```

   For a keystore whose password is stored externally:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE
   IDENTIFIED BY EXTERNAL STORE;
   ```

3. From the hardware security module management interface, create a new hardware security module password.

4. In SQL*Plus, open the hardware keystore.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
   IDENTIFIED BY "psmith:new_password";

   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
   IDENTIFIED BY EXTERNAL STORE;
   ```

Related Topics

- Closing a Hardware Keystore
  To close a hardware keystore, you must use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEYSTORE CLOSE` clause.
- Opening a Hardware Keystore
  To open a hardware keystore, use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEYSTORE OPEN` clause.

Configuring an External Store for a Keystore Password

An external store for a keystore password stores the keystore password in a centrally accessed and managed location.

- About Configuring an External Store for a Keystore Password
- Configuring the Keystore Password External Store by Setting WALLET_ROOT
  The preferred approach to configuring the external store for the keystore password is the set the `WALLET_ROOT` parameter.
- Configuring the Keystore Password External Store by Editing sqlnet.ora
  As an alternative to setting the `WALLET_ROOT` parameter, you can edit the `sqlnet.ora` file to configure the external password store for the keystore password.
- When to Use the EXTERNAL STORE Clause After Configuration
  After you configure the external store for a keystore password, you can use the `EXTERNAL_STORE` clause in the `ADMINISTER KEY MANAGEMENT` statement.

About Configuring an External Store for a Keystore Password

An external store for a keystore password allows you to easily remove that keystore password from the `ADMINISTER KEY MANAGEMENT` command line, implementing separation of duties between database administrators and key administrators. It is also useful for situations in which you use automated tools to perform Transparent Data Encryption operations that require a password, when the scripts that run the automated tools include hard-coded password. To avoid hard-coding the password in a script, you can store this password in an external store on the database server. In a multitenant environment, different PDBs can make use of the external store.

Related Topics

- Storing Oracle Database Secrets
  Secrets are data that support internal Oracle Database features that integrate external clients such as Oracle GoldenGate into the database.

Configuring the Keystore Password External Store by Setting WALLET_ROOT

The preferred approach to configuring the external store for the keystore password is the set the `WALLET_ROOT` parameter.

1. Connect to the united mode CDB root or isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege and who has the `ALTER SYSTEM` system privilege.

   For example:

```
CONNECT c##sec_admin AS SYSKM (or CONNECT sec_admin@pdb_name AS SYSKM)
Enter password: password
```

2. Set the external keystore credential location by running the `ALTER SYSTEM` statement for the `EXTERNAL_KEYSTORE_CREDENTIAL_LOCATION` parameter.

   If you had set the `WALLET_ROOT` and `TDE_CONFIGURATION` parameters, then the external keystore credential location becomes *`WALLET_ROOT`*`/tde_seps`. In that case, You do not need to set the `EXTERNAL_KEYSTORE_CREDENTIAL_LOCATION` parameter.

   For example:

   ```
   ALTER SYSTEM SET EXTERNAL_KEYSTORE_CREDENTIAL_LOCATION = "/etc/ORACLE/
   WALLETS/orcl/external_store" SCOPE = SPFILE;
   ```

3. Create an auto-login keystore that contains the keystore password, by including the `ADD SECRET` clause to the `ADMINISTER KEY MANAGEMENT` statement.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT ADD SECRET 'password'
   FOR CLIENT 'TDE_WALLET'
   TO LOCAL AUTO_LOGIN KEYSTORE '/etc/ORACLE/WALLETS/orcl/external_store';
   ```

   In this example:

   - Enter `'TDE_WALLET'`, in capital letters and enclosed in single quotation marks, for the *`client_identifier`* value set by the `FOR CLIENT` clause. `TDE_WALLET` is for a keystore that is configured as `FILE`. Alternatively, you can enter `OKV_PASSWORD` for an Oracle Key Vault HSM, or `HSM_PASSWORD` for a third-party password. This is a fixed value and must be entered as shown here for this application of the `ADD SECRET` clause. If you do not enter this value, then TDE will be unable to find this secret, and attempts to use the `IDENTIFIED BY EXTERNAL STORE` setting will generate an `ORA-00988: missing or invalid password(s)` error message.
   - `LOCAL` in the `TO LOCAL AUTO_LOGIN KEYSTORE` clause is optional, but cannot be used if the wallet is shared between Oracle RAC instances.

   In this example,

4. Restart the database (for the CDB root) or close and re-open the PDB.

   To restart from the CDB root:

   ```
   SHUTDOWN IMMEDIATE
   STARTUP
   ```

   To close and re-open a PDB:

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

Related Topics

- When to Use the EXTERNAL STORE Clause After Configuration
  After you configure the external store for a keystore password, you can use the `EXTERNAL_STORE` clause in the `ADMINISTER KEY MANAGEMENT` statement.

Configuring the Keystore Password External Store by Editing sqlnet.ora

As an alternative to setting the WALLET_ROOT parameter, you can edit the sqlnet.ora file to configure the external password store for the keystore password.

- To set the external keystore credential location, edit the init.ora file for the database instance.

  For example:

  ```
  EXTERNAL_KEYSTORE_CREDENTIAL_LOCATION = "/etc/ORACLE/WALLETS/orcl/
  external_store"
  ```

  By default, the init.ora file is located in the ORACLE_HOME/dbs directory or in the location set by the TNS_ADMIN environment variable.

Related Topics

- When to Use the EXTERNAL STORE Clause After Configuration
  After you configure the external store for a keystore password, you can use the EXTERNAL_STORE clause in the ADMINISTER KEY MANAGEMENT statement.

When to Use the EXTERNAL STORE Clause After Configuration

After you configure the external store for a keystore password, you can use the EXTERNAL_STORE clause in the ADMINISTER KEY MANAGEMENT statement.

You must use the EXTERNAL STORE clause in the ADMINISTER KEY MANAGEMENT statement for the following operations: opening, closing, backing up the keystore; adding, updating, or deleting a secret keystore; creating, using, rekeying, tagging, importing, exporting encryption keys.

For example:

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
IDENTIFIED BY EXTERNAL STORE;
```

You can change or delete external keystore passwords by using the ADMINISTER KEY MANAGEMENT UPDATE CLIENT SECRET statement or the ADMINISTER KEY MANAGEMENT DELETE CLIENT SECRET statement.

Backing Up Password-Protected Software Keystores

When you back up a password-protected software keystore, you can create a backup identifier string to describe the backup type.

- About Backing Up Password-Protected Software Keystores
  You must back up password-protected software keystores, as per the security policy and requirements of your site.
- Creating a Backup Identifier String for the Backup Keystore
  The backup file name of a software password keystore is derived from the name of the password-protected software keystore.

- Backing Up a Password-Protected Software Keystore
  The `BACKUP KEYSTORE` clause of the `ADMINISTER KEY MANAGEMENT` statement backs up a password-protected software keystore.

About Backing Up Password-Protected Software Keystores

You must back up password-protected software keystores, as per the security policy and requirements of your site.

A backup of the keystore contains all of the keys contained in the original keystore. Oracle Database prefixes the backup keystore with the creation time stamp (UTC). If you provide an identifier string, then this string is inserted between the time stamp and keystore name.

After you complete the backup operation, the keys in the original keystore are marked as "backed up". You can check the status of keys querying the `V$ENCRYPTION_WALLET` data dictionary view.

You cannot back up auto-login or local auto-login software keystores. No new keys can be added to them directly through the `ADMINISTER KEY MANAGEMENT` statement operations. The information in these keystores is only read and hence there is no need for a backup.

You must include the `WITH BACKUP` clause in any `ADMINISTER KEY MANAGEMENT` statement that changes the wallet (for example, changing the wallet password, or setting the master encryption key).

Creating a Backup Identifier String for the Backup Keystore

The backup file name of a software password keystore is derived from the name of the password-protected software keystore.

- To create a backup identifier string for a backup keystore, use the `ADMINISTER KEY MANAGEMENT` SQL statement with the `BACKUP KEYSTORE` clause, with the following syntax:

  ```
  ewallet_creation-time-stamp-in-UTC_user-defined-string.p12
  ```

  When you create the backup identifier (*`user_defined_string`*), use the operating system file naming convention. For example, in UNIX systems, you may want to ensure that this setting does not have spaces.

  The following example shows the creation of a backup keystore that uses a user-identified string, and how the resultant keystore appears in the file system. This example includes the `FORCE KEYSTORE` clause in the event the auto-login keystore is in use or the keystore is closed.

  ```
  ADMINISTER KEY MANAGEMENT BACKUP KEYSTORE USING 'Monthly-backup-2013-04'
  FORCE KEYSTORE
  IDENTIFIED BY keystore_password;
  ```

  This version is for a scenario in which the password is stored in an external store:

  ```
  ADMINISTER KEY MANAGEMENT BACKUP KEYSTORE USING 'Monthly-backup-2013-04'
  FORCE KEYSTORE
  IDENTIFIED BY EXTERNAL STORE;
  ```

  Resultant keystore file:

```
ewallet_2013041513244657_Monthly-backup-2013-04.p12
```

Backing Up a Password-Protected Software Keystore

The BACKUP KEYSTORE clause of the ADMINISTER KEY MANAGEMENT statement backs up a password-protected software keystore.

- Back up the keystore by using the following syntax:

```
ADMINISTER KEY MANAGEMENT BACKUP KEYSTORE
[USING 'backup_identifier']
FORCE KEYSTORE]
IDENTIFIED BY [EXTERNAL STORE | software_keystore_password]
[TO 'keystore_location'];
```

  In this specification:

  - USING *backup_identifier* is an optional string that you can provide to identify the backup. Enclose this identifier in single quotation marks (' '). This identifier is appended to the named keystore file (for example, ewallet_*time-stamp*_emp_key_backup.p12).
  - FORCE KEYSTORE temporarily opens the password-protected keystore for this operation. You must open the keystore for this operation.
  - IDENTIFIED BY can be one of the following settings:
    - EXTERNAL STORE uses the keystore password stored in the external store to perform the keystore operation.
    - *software_keystore_password* is the password for the keystore.
  - *keystore_location* is the path at which the backup keystore is stored. If you do not specify the *keystore_location*, then the backup is created in the same directory as the original keystore. Enclose this location in single quotation marks (' ').

  The following example backs up a software keystore into another location.

```
ADMINISTER KEY MANAGEMENT BACKUP KEYSTORE
USING 'hr.emp_keystore'
FORCE KEYSTORE
IDENTIFIED BY software_keystore_password
TO '/etc/ORACLE/KEYSTORE/DB1/';

keystore altered.
```

  In the following version, the password for the keystore is external, so the EXTERNAL STORE clause is used. The keystore is backed up into the same directory as the current keystore.

```
ADMINISTER KEY MANAGEMENT BACKUP KEYSTORE
USING 'hr.emp_keystore'
FORCE KEYSTORE
IDENTIFIED BY EXTERNAL STORE;
```

  After you run this statement, an ewallet_*identifier*.p12 file (for example, ewallet_*time-stamp*_hr.emp_keystore.p12) appears in the keystore location.

How the V$ENCRYPTION_WALLET View Interprets Backup Operations

The BACKUP column of the V$ENCRYPTION_WALLET view indicates a how a copy of the keystore was created.

The column indicates if a copy of the keystore had been created with the WITH BACKUP clause of the ADMINISTER KEY MANAGEMENT statement or the ADMINISTER KEY MANAGEMENT BACKUP KEYSTORE statement.

When you modify a key or a secret, the modifications that you make do not exist in the previously backed-up copy, because you make a copy and then modify the key itself. Because there is no copy of the modification in the previous keystores, the BACKUP column is set to NO, even if the BACKUP had been set to YES previously. Hence, if the BACKUP column is YES, then after you perform an operation that requires a backup, such as adding a custom attribute tag, the BACKUP column value changes to NO.

Backups of the Hardware Keystore

You cannot use Oracle Database to back up hardware keystores.

See your HSM vendor instructions for information about backing up keys for hardware keystores.

Merging Software Keystores

You can merge software keystores in a variety of ways.

- About Merging Software Keystores
  You can merge any combination of software keystores, but the merged keystore must be password-protected. It can have a password that is different from the constituent keystores.
- Merging One Software Keystore into an Existing Software Keystore
  You can use the ADMINISTER KEY MANAGEMENT statement with the MERGE KEYSTORE clause to merge one software keystore into another existing software keystore.
- Merging Two Software Keystores into a Third New Keystore
  You can merge two software keystores into a third new keystore, so that the two existing keystores are not changed.
- Merging an Auto-Login Software Keystore into an Existing Password-Protected Software Keystore
  You can merge an auto-login software keystore into an existing password-protected software keystore.
- Reversing a Software Keystore Merge Operation
  You cannot directly reverse a keystore merge operation.

About Merging Software Keystores

You can merge any combination of software keystores, but the merged keystore must be password-protected. It can have a password that is different from the constituent keystores.

To use the merged keystore, you must explicitly open the merged keystore after you create it, even if one of the constituent keystores was already open before the merge.

Whether a common key from two source keystores is added or overwritten to a merged keystore depends on how you write the `ADMINISTER KEY MANAGEMENT` merge statement. For example, if you merge Keystore 1 and Keystore 2 to create Keystore 3, then the key in Keystore 1 is added to Keystore 3. If you merge Keystore 1 into Keystore 2, then the common key in Keystore 2 is not overwritten.

The `ADMINISTER KEY MANAGEMENT` merge statement has no bearing on the configured keystore that is in use. However, the merged keystore can be used as the new configured database keystore if you want. Remember that you must reopen the keystore if you are using the newly created keystore as the keystore for the database at the location configured by the `WALLET_ROOT` parameter.

Related Topics

- Migrating Between a Software Password Keystore and a Hardware Keystore
  You can migrate between password-protected software keystores and hardware keystores.
- Step 3: Open the Software Keystore
  Depending on the type of keystore you create, you must manually open the keystore before you can use it.

Merging One Software Keystore into an Existing Software Keystore

You can use the `ADMINISTER KEY MANAGEMENT` statement with the `MERGE KEYSTORE` clause to merge one software keystore into another existing software keystore.

- To perform this type of merge, follow the steps in Merging Two Software Keystores into a Third New Keystore but use the following SQL statement:

  ```
  ADMINISTER KEY MANAGEMENT MERGE KEYSTORE 'keystore1_location'
  [IDENTIFIED BY software_keystore1_password]
  INTO EXISTING KEYSTORE 'keystore2_location'
  IDENTIFIED BY software_keystore2_password
  [WITH BACKUP [USING 'backup_identifier]];
  ```

  In this specification:

  - *keystore1_location* is the directory location of the first keystore, which will be left unchanged after the merge. Enclose this path in single quotation marks (' ').
  - The `IDENTIFIED BY` clause is required for the first keystore if it is a password-protected keystore. *software_keystore1_password* is the password for the first keystore.
  - *keystore2_location* is the directory location of the second keystore into which the first keystore is to be merged. Enclose this path in single quotation marks (' ').
  - *software_keystore2_password* is the password for the second keystore.
  - `WITH BACKUP` creates a backup of the software keystore. Optionally, you can use the `USING` clause to add a brief description of the backup. Enclose this description in single quotation marks (' '). This identifier is appended to the named keystore file (for example, `ewallet_`*time-stamp_emp_key_backup*`.p12`,

with *emp_key_backup* being the backup identifier). Follow the file naming conventions that your operating system uses.

The resultant keystore after the merge operation is always a password-protected keystore.

Merging Two Software Keystores into a Third New Keystore

You can merge two software keystores into a third new keystore, so that the two existing keystores are not changed.

1. Log in to the database instance as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.

   For example:

   ```
   sqlplus sec_admin as syskm
   Enter password: password
   Connected.
   ```

2. Merge the software keystores by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT MERGE KEYSTORE 'keystore1_location'
   [IDENTIFIED BY software_keystore1_password]
   AND KEYSTORE 'keystore2_location'
   [IDENTIFIED BY software_keystore2_password]
   INTO NEW KEYSTORE 'keystore3_location'
   IDENTIFIED BY software_keystore3_password;
   ```

   In this specification:

   - *keystore1_location* is the directory location of the first keystore, which will be left unchanged after the merge. Enclose this path in single quotation marks (' ').
   - The `IDENTIFIED BY` clause is required for the first keystore if it is a password-protected keystore. *software_keystore1_password* is the current password for the first keystore.
   - *keystore2_location* is the directory location of the second keystore. Enclose this path in single quotation marks (' ').
   - The `IDENTIFIED BY` clause is required for the second keystore if it is a password-protected keystore. *software_keystore2_password* is the current password for the second keystore.
   - *keystore3_location* specifies the directory location of the new, merged keystore. Enclose this path in single quotation marks (' '). If there is already an existing keystore at this location, the command exits with an error.
   - *software_keystore3_password* is the new password for the merged keystore.

   The following example merges an auto-login software keystore with a password-protected keystore to create a merged password-protected keystore at a new location:

   ```
   ADMINISTER KEY MANAGEMENT MERGE KEYSTORE '/etc/ORACLE/KEYSTORE/DB1'
   AND KEYSTORE '/etc/ORACLE/KEYSTORE/DB2'
   IDENTIFIED BY existing_password_for_keystore_2
   INTO NEW KEYSTORE '/etc/ORACLE/KEYSTORE/DB3'
   IDENTIFIED BY new_password_for_keystore_3;

   keystore altered.
   ```

Merging an Auto-Login Software Keystore into an Existing Password-Protected Software Keystore

You can merge an auto-login software keystore into an existing password-protected software keystore.

- Use the `ADMINISTER KEY MANAGEMENT MERGE KEYSTORE` SQL statement to merge an auto-login software keystore into an existing password-protected software keystore.

Example 4-1 shows how to merge an auto-login software keystore into a password-protected software keystore. It also creates a backup of the second keystore before creating the merged keystore.

Example 4-1 Merging a Software Auto-Login Keystore into a Password Keystore

```
ADMINISTER KEY MANAGEMENT MERGE KEYSTORE '/etc/ORACLE/KEYSTORE/DB1'
INTO EXISTING KEYSTORE '/etc/ORACLE/KEYSTORE/DB2'
IDENTIFIED BY keystore_password WITH BACKUP;
```

In this specification:

- `MERGE KEYSTORE` must specify the auto-login keystore.
- `EXISTING KEYSTORE` refers to the password keystore.

Reversing a Software Keystore Merge Operation

You cannot directly reverse a keystore merge operation.

When you merge a keystore into an existing keystore (rather than creating a new one), you must include the `WITH BACKUP` clause in the `ADMINISTER KEY MANAGEMENT` statement to create a backup of this existing keystore. Later on, if you decide that you must reverse the merge, you can replace the merged software keystore with the one that you backed up.

In other words, suppose you want merge Keystore A into Keystore B. By using the `WITH BACKUP` clause, you create a backup for Keystore B before the merge operation begins. (The original Keystore A is still intact.) To reverse the merge operation, revert to the backup that you made of Keystore B.

- Use the `ADMINISTER KEY MANAGEMENT MERGE KEYSTORE` SQL statement to perform merge operations.
  - For example, to perform a merge operation into an existing keystore:

    ```
    ADMINISTER KEY MANAGEMENT MERGE KEYSTORE '/etc/ORACLE/KEYSTORE/DB1'
    INTO EXISTING KEYSTORE '/etc/ORACLE/KEYSTORE/DB2'
    IDENTIFIED BY password WITH BACKUP USING "merge1";
    ```

    Replace the new keystore with the backup keystore, which in this case would be named `ewallet_time-stamp_merge1.p12`.
  - To merge an auto-login keystore into a password-based keystore, use the `ADMINISTER KEY MANAGEMENT MERGE KEYSTORE` SQL statement.

Moving a Software Keystore to a New Location

You move a software keystore to a new location after you have updated the `WALLET_ROOT` parameter.

If you are using Oracle Key Vault, then you can configure a TDE direct connection where Key Vault directly manages the master encryption keys. In this case, you will never need to manually move the keystore to a new location.

1. Log in to the database instance as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Back up the software keystore.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT BACKUP KEYSTORE
   USING 'hr.emp_keystore'
   FORCE KEYSTORE
   IDENTIFIED BY
   software_keystore_password TO '/etc/ORACLE/KEYSTORE/DB1/';
   ```

3. Close the software keystore.

   Examples of ways that you can close the keystore are as follows.

   For an auto-login software keystore:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE;
   ```

   For a password-protected software keystore:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE
   IDENTIFIED BY software_keystore_password;
   ```

   For a keystore for which the password is stored externally:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE
   IDENTIFIED BY EXTERNAL STORE;
   ```

4. Exit the database session.

   For example, if you are logged in to SQL*Plus:

   ```
   EXIT
   ```

5. In the `init.ora` file for the database instance, update the `WALLET_ROOT` parameter to point to the new location where you want to move the keystore.
6. Use the operating system move command (such as `mv`) to move the keystore with all of its keys to the new directory location.

Related Topics

- *Oracle Key Vault Administrator's Guide*

Moving a Software Keystore Out of Automatic Storage Management

You can use the `ADMINISTER KEY MANAGEMENT` statement to move a software keystore out Automatic Storage Management.

1. Log in to the database instance as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.

   For example:

   ```
   sqlplus sec_admin as syskm
   Enter password: password
   Connected.
   ```

2. Initialize a target keystore on the file system by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT CREATE KEYSTORE targetKeystorePath
   IDENTIFIED BY targetKeystorePassword;
   ```

   In this specification:

   - *targetKeystorePath* is the directory path to the target keystore on the file system.
   - *targetKeystorePassword* is a password that you create for the keystore.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT CREATE KEYSTORE '/etc/ORACLE/KEYSTORE/DB1/'
   IDENTIFIED BY "targetKeystorePassword";
   ```

3. Copy the keystore from ASM to the target keystore that you just created.

   This step requires that you merge the keystore from ASM to the file system, as follows:

   ```
   ADMINISTER KEY MANAGEMENT MERGE KEYSTORE srcKeystorePath
   IDENTIFIED BY srcKeystorePassword
   INTO EXISTING KEYSTORE targetKeystorePath
   IDENTIFIED BY targetKeystorePassword
   WITH BACKUP USING backupIdentifier;
   ```

   In this specification:

   - *srcKeystorePath* is the directory path to the source keystore.
   - *srcKeystorePassword* is th source keystore password.
   - *targetKeystorePath* is the path to the target keystore.
   - *targetKeystorePassword* is the target keystore password.
   - *backupIdentifier* is the backup identifier to be added to the backup file name.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT MERGE KEYSTORE '+DATAFILE'
   IDENTIFIED BY "srcPassword"
   INTO EXISTING KEYSTORE '/etc/ORACLE/KEYSTORE/DB1/'
   IDENTIFIED BY "targetKeystorePassword"
   WITH BACKUP USING "bkup";
   ```

Migrating Between a Software Password Keystore and a Hardware Keystore

You can migrate between password-protected software keystores and hardware keystores.

Migrating from a Password-Protected Software Keystore to a Hardware Keystore

You can migrate from a password-protected software keystore to a hardware keystore.

Step 1: Convert the Software Keystore to Open with the Hardware Keystore

Some Oracle tools require access to the old software keystore to encrypt or decrypt data that was exported or backed up using the software keystore.

Examples of these tools are Oracle Data Pump and Oracle Recovery Manager.

- Use the `ADMINISTER KEY MANAGEMENT` SQL statement to convert a software keystore to a open with a hardware keystore.
  - To set the software keystore password as that of the hardware keystore, use the following syntax:

    ```
    ADMINISTER KEY MANAGEMENT ALTER KEYSTORE PASSWORD
    FORCE KEYSTORE
    IDENTIFIED BY software_keystore_password
    SET "hardware_keystore_credentials" WITH BACKUP
    [USING 'backup_identifier'];
    ```

    In this specification:

    * *`software_keystore_password`* is the same password that you used when creating the software keystore.
    * *`hardware_keystore_credentials`* is the new software keystore password which is the same as the password of the hardware keystore.
    * `WITH BACKUP` creates a backup of the software keystore. Optionally, you can use the `USING` clause to add a brief description of the backup. Enclose this description in single quotation marks (' '). This identifier is appended to the named keystore file (for example, `ewallet_`*`time-stamp_emp_key_backup`*`.p12`, with *`emp_key_backup`* being the backup

identifier). Follow the file naming conventions that your operating system uses.

– To create an auto-login keystore for a software keystore, use the following syntax:

```
ADMINISTER KEY MANAGEMENT CREATE [LOCAL] AUTO_LOGIN KEYSTORE
FROM KEYSTORE 'keystore_location'
IDENTIFIED BY software_keystore_password;
```

In this specification:

* `LOCAL` enables you to create a local auto-login software keystore. Otherwise, omit this clause if you want the keystore to be accessible by other computers.
* *`keystore_location`* is the path to the keystore directory location of the keystore that is configured in the `sqlnet.ora` file.
* *`software_keystore_password`* is the existing password of the configured software keystore.

Step 2: Configure the Hardware Security Module Keystore Type

You can use the `ALTER SYSTEM` statement to configure the HSM keystore type.

For the software keystore to open with the hardware keystore, either the software keystore must have the same password as the hardware keystore, or alternatively, you can create an auto-login keystore for the software keystore.

1. Log in to the database instance as a user who has been granted the `SYSDBA` administrative privilege.

   For example:

   ```
   sqlplus sec_admin as sysdba
   Enter password: password
   ```

2. Set the `TDE_CONFIGURATION` dynamic initialization parameter.

   The following example migrates from a TDE keystore to a hardware keystore:

   ```
   ALTER SYSTEM SET TDE_CONFIGURATION="KEYSTORE_CONFIGURATION=HSM|FILE";
   ```

   The next example migrates from a TDE keystore to an Oracle Key Vault keystore:

   ```
   ALTER SYSTEM SET TDE_CONFIGURATION="KEYSTORE_CONFIGURATION=OKV|FILE";
   ```

3. Restart the database.

   ```
   SHUTDOWN IMMEDIATE
   STARTUP
   ```

Step 3: Perform the Hardware Keystore Migration

You can use the `ADMINISTER KEY MANAGEMENT` SQL statement to perform a hardware keystore migration.

To migrate from the software keystore to hardware keystore, you must use the `MIGRATE USING` *`keystore_password`* clause in the `ADMINISTER KEY MANAGEMENT SET KEY` SQL statement to decrypt the existing TDE table keys and the tablespace encryption keys with the TDE master encryption key in the software keystore and then

reencrypt them with the newly created TDE master encryption key in the hardware keystore.

After you complete the migration, you do not need to restart the database, nor do you need to manually re-open the hardware keystore. The migration process automatically reloads the keystore keys in memory.

- Migrate the hardware keystores by using the following syntax:

```
ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY
IDENTIFIED BY "user_name:password"
MIGRATE USING software_keystore_password
[WITH BACKUP [USING 'backup_identifier']];
```

  In this specification:

  – *user_name*:*password* is the user ID and password that was created in Step 2 under Step 2: Configure the Hardware Security Module (in Configuring Transparent Data Encryption). Enclose this setting in double quotation marks (" ") and separate *user_name* and *password* with a colon (:).

  – *software_keystore_password* is the same password that you used when creating the software keystore or that you have changed to in Step 1: Convert the Software Keystore to Open with the Hardware Keystore.

  – USING enables you to add a brief description of the backup. Enclose this description in single quotation marks (' '). This identifier is appended to the named keystore file (for example, ewallet_*time-stamp_emp_key_backup*.p12, with *emp_key_backup* being the backup identifier). Follow the file naming conventions that your operating system uses.

> **Note:**
>
> If the database contains columns encrypted with a public key, then the columns are decrypted and reencrypted with an AES symmetric key generated by HSM-based Transparent Data Encryption.

Migrating from a Hardware Keystore to a Password-Based Software Keystore

You can migrate a hardware keystore to a software keystore.

- About Migrating Back from a Hardware Keystore
  To switch from using a hardware keystore solution to a software keystore, you can use reverse migration of the keystore.
- Step 1: Configure Hardware Security Module Keystore Type
  You can use the ALTER SYSTEM statement to configure the hardware security module keystore type.
- Step 2: Configure the Keystore for the Reverse Migration
  The ADMINISTER KEY MANAGEMENT statement with the SET ENCRYPTION KEY and REVERSE MIGRATE clauses can be used to reverse the migration of a keystore.
- Step 3: Configure the Hardware Keystore to Open with the Software Keystore
  After you complete the migration, the migration process automatically reloads the keystore keys in memory.

About Migrating Back from a Hardware Keystore

To switch from using a hardware keystore solution to a software keystore, you can use reverse migration of the keystore.

After you complete the switch, keep the hardware security module, in case earlier backup files rely on the TDE master encryption keys in the hardware security module.

If you had originally migrated from the software keystore to the hardware security module and reconfigured the software keystore as described in Migration of a Previously Configured TDE Master Encryption Key, then you already have an existing keystore with the same password as the HSM password. Reverse migration configures this keystore to act as the new software keystore with a new password. If your existing keystore is an auto-login software keystore and you have the password-based software keystore for this auto-login keystore, then use the password-based keystore. If the password-based keystore is not available, then merge the auto-login keystore into a newly created empty password-based keystore, and use the newly created password-based keystore.

If you do not have an existing keystore, then you must specify a keystore location using the `WALLET_ROOT` parameter in the `init.ora` file. When you perform the reverse migration, migrate to the previous keystore so that you do not lose the keys.

Related Topics

- Merging Software Keystores
  You can merge software keystores in a variety of ways.

Step 1: Configure Hardware Security Module Keystore Type

You can use the `ALTER SYSTEM` statement to configure the hardware security module keystore type.

1. Log in to the database instance as a user who has been granted the `SYSDBA` administrative privilege.

   For example:

   ```
   sqlplus sec_admin as sysdba
   Enter password: password
   ```

2. Set the `TDE_CONFIGURATION` dynamic initialization parameter to specify the TDE keystore type.

   For example:

   ```
   ALTER SYSTEM SET TDE_CONFIGURATION="KEYSTORE_CONFIGURATION=HSM";
   ```

   Setting `KEYSTORE_CONFIGURATION` to `HSM` eliminates the possibility of using an auto-login HSM configuration. To use an auto-login HSM configuration, you would set `KEYSTORE_CONFIGURATION` to `HSM:FILE`.

3. Restart the database.

   ```
   SHUTDOWN IMMEDIATE
   STARTUP
   ```

Step 2: Configure the Keystore for the Reverse Migration

The `ADMINISTER KEY MANAGEMENT` statement with the `SET ENCRYPTION KEY` and `REVERSE MIGRATE` clauses can be used to reverse the migration of a keystore.

1. Log in to the database instance as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.

   For example:

   ```
   sqlplus sec_admin as syskm
   Enter password: password
   Connected.
   ```

2. Reverse migrate the keystore by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY
   IDENTIFIED BY software_keystore_password
   REVERSE MIGRATE USING "user_name:password"
   [WITH BACKUP [USING 'backup_identifier']];
   ```

   In this specification:

   - *software_keystore_password* is the password for the existing keystore or the new keystore.
   - *user_name*:*password* is the user ID and password that was created in Step 2 in Step 2: Configure the Hardware Security Module (in Configuring Transparent Data Encryption). If the pre-hardware security module software keystore is the new keystore, then you must ensure that it has the same password as the *user_name*:*password* before issuing the reverse migration command. Enclose this setting in double quotation marks (" ").
   - `WITH BACKUP` creates a backup of the software keystore. Optionally, you can include the `USING` clause to add a brief description of the backup. Enclose this description in single quotation marks (' '). This identifier is appended to the named keystore file (for example, `ewallet_`*time-stamp_emp_key_backup*`.p12`, with *emp_key_backup* being the backup identifier). Follow the file naming conventions that your operating system uses.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY
   IDENTIFIED BY password
   REVERSE MIGRATE USING "psmith:password" WITH BACKUP;

   keystore altered.
   ```

3. Optionally, change the keystore password.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT ALTER KEYSTORE PASSWORD
   IDENTIFIED BY old_password
   SET new_password
   WITH BACKUP USING 'pwd_change';
   ```

Related Topics

- Changing the Password of a Software Keystore
  Oracle Database enables you to easily change password-protected software keystore passwords.

Step 3: Configure the Hardware Keystore to Open with the Software Keystore

After you complete the migration, the migration process automatically reloads the keystore keys in memory.

You do not need to restart the database, nor do you need to manually re-open the software keystore.

The hardware keystore may still be required after reverse migration because the old keys are likely to have been used for encrypted backups or by tools such as Oracle Data Pump and Oracle Recovery Manager. You should cache the hardware keystore credentials in the keystore so that the HSM can be opened with the software keystore.

Related Topics

- Configuring Auto-Login Hardware Security Modules
  A hardware security module can be configured to use the auto-login capability.

Keystore Order After a Migration

After you perform a migration, keystores can be either primary or secondary in their order.

The `WALLET_ORDER` column of the `V$ENCRYPTION_WALLET` dynamic view describes whether a keystore is primary (that is, it holds the current TDE master encryption key) or if it is secondary (it holds the previous TDE master encryption key). The `WRL_TYPE` column describes the type of locator for the keystore (for example, `FILE` for the `sqlnet.ora` file). The `WALLET_ORDER` column shows `SINGLE` if two keystores are not configured together and no migration was ever performed previously.

Table 4-1 describes how the keystore order works after you perform a migration.

Table 4-1 Keystore Order After a Migration

Type of Migration Done	WRL_TYPE	WALLET_ORDER	Description
Migration of software keystore to HSM	`HSM` `FILE`	`PRIMARY` `SECONDARY`	Both the HSM and software keystore are configured. The TDE master encryption key can be either in the HSM or the software keystore. The TDE master encryption key is first searched in the HSM. If the TDE master encryption key is not in the primary keystore (HSM), then it will be searched for in the software keystore. All of the new TDE master encryption keys will be created in the primary keystore (in this case, the HSM).

Table 4-1 (Cont.) Keystore Order After a Migration

Type of Migration Done	WRL_TYPE	WALLET_ORDER	Description
Reverse migration of HSM to software keystore	FILE HSM	PRIMARY SECONDARY	Both the HSM and software keystore are configured. The TDE master encryption key can be either in the HSM or the software keystore. The TDE master encryption key is first searched for in the software keystore. If the TDE master encryption key is not present in the primary (that is, software) keystore, then it will be searched for in the HSM. All of the new TDE master encryption keys will be created in the primary keystore (in this case, the software keystore).

Migration of Keystores to and from Oracle Key Vault

You can use Oracle Key Vault to migrate both software and hardware keystores to and from Oracle Key Vault.

This enables you to manage the keystores centrally, and then share the keystores as necessary with other TDE-enabled databases in your enterprise.

Oracle Key Vault enables you to upload a keystore to a container called a virtual wallet, and then create a new virtual wallet from the contents of previously uploaded Oracle keystores. For example, suppose you previously uploaded a keystore that contains 5 keys. You can create a new virtual wallet that consists of only 3 of these keys. You then can download this keystore to another TDE-enabled database. This process does not modify the original keystore.

In addition to Oracle keystores, Oracle Key Vault enables you to securely share other security objects, such as credential files and Java keystores, across the enterprise. It prevents the loss of keys and keystores due to forgotten passwords or accidentally deleted keystores. You can use Oracle Key Vault with products other than TDE: Oracle Real Application Security, Oracle Active Data Guard, and Oracle GoldenGate. Oracle Key Vault facilitates the movement of encrypted data using Oracle Data Pump and Oracle Transportable Tablespaces.

Related Topics

- *Oracle Key Vault Administrator's Guide*

Configuring Keystores for Automatic Storage Management

You can store a software keystore on an Automatic Storage Management (ASM) disk group.

- About Configuring Keystores for Automatic Storage Management
  You can configure a keystore for Automatic Storage Management (ASM) for a standalone database or a multitenant environment. The WALLET_ROOT location can be compliant or non-compliant with Oracle Managed File (OMF) systems.

- Configuring a Keystore on a Standalone Database to Point to an ASM Location
  On a standalone database system, you can set the WALLET_ROOT parameter to point to an ASM location.
- Configuring a Keystore in a Multitenant Environment to Point to an ASM Location
  You can set WALLET_ROOT to point to an ASM directory within which the TDE keystore of the CDB root (which all united mode PDBs share) and the TDE keystores of all isolated mode PDBs are located.
- Configuring a Keystore to Point to an ASM Location When the WALLET_ROOT Location Does Not Follow OMF Guidelines
  If the chosen WALLET_ROOT location does not comply with the Oracle Managed File (OMF) guidelines, then the Oracle database cannot perform automation of the directory creation.

About Configuring Keystores for Automatic Storage Management

You can configure a keystore for Automatic Storage Management (ASM) for a standalone database or a multitenant environment. The WALLET_ROOT location can be compliant or non-compliant with Oracle Managed File (OMF) systems.

You should use the WALLET_ROOT and TDE_CONFIGURATION initialization parameters to configure the keystore location in an ASM system. The TDE_CONFIGURATION parameter must be set with the attribute KEYSTORE_CONFIGURATION=FILE in order for the WALLET_ROOT parameter to work. Note that starting with Oracle Database release 19c, the ENCRYPTION_WALLET_LOCATION, set in the sqlnet.ora file, is deprecated in favor of WALLET_ROOT and TDE_CONFIGURATION.

To perform the configuration, you must specify a + sign, followed by the ASM disk group and path where the keystore will be located. For example:

```
WALLET_ROOT=+disk_group/path
```

Note the following:

- When you designate the path for the WALLET_ROOT for databases in standalone or multitenant environments, or environments where the WALLET_ROOT location either complies or does not comply with the Oracle Managed File (OMF) directory naming convention, be aware that this path must follow certain conventions so that the database can automate the creation of the directory components of the TDE keystore locations for you. Otherwise, you must manually create the directories under the WALLET_ROOT location.
- If you must move or merge software keystores between a regular file system and an ASM file system, then you can use the same keystore merge statements that are used to merge software keystores.
- To execute commands to manage keystores in an ASM environment, you can use the ASMCMD utility.

Related Topics

- Merging Software Keystores
  You can merge software keystores in a variety of ways.
- Step 1: Configure the Software Keystore Location and Type
  You must configure the keystore location and type by setting WALLET_ROOT in init.ora and TDE_CONFIGURATION in the database instance.
- *Oracle Automatic Storage Management Administrator's Guide*

Configuring a Keystore on a Standalone Database to Point to an ASM Location

On a standalone database system, you can set the `WALLET_ROOT` parameter to point to an ASM location.

1. Ensure that the `KEYSTORE_CONFIGURATION` attribute of the `TDE_CONFIGURATION` dynamic initialization parameter is set to `FILE`.

   For example, in SQL*Plus:

   ```
   ALTER SYSTEM SET TDE_CONFIGURATION="KEYSTORE_CONFIGURATION=FILE";
   ```

2. In the `init.ora` file, set the `WALLET_ROOT` static initialization parameter to the ASM disk group location.

   Optionally, include `OMF` (in upper case) to provide Oracle Managed File (OMF) compliance and to facilitate automation of the directory creation.

   For example:

   ```
   WALLET_ROOT=+disk_group_name/OMF
   ```

   This setting places the TDE keystore of the standalone database system in the *`WALLET_ROOT`*`/tde` directory (that is, `+`*`disk_group_name`*`/OMF/tde` with the `tde` component of the path being automatically created by the database server).

Configuring a Keystore in a Multitenant Environment to Point to an ASM Location

You can set `WALLET_ROOT` to point to an ASM directory within which the TDE keystore of the CDB root (which all united mode PDBs share) and the TDE keystores of all isolated mode PDBs are located.

1. Ensure that the `KEYSTORE_CONFIGURATION` attribute of the `TDE_CONFIGURATION` dynamic initialization parameter is set to `FILE`.

   For a CDB, set `TDE_CONFIGURATION` in the CDB root; for a PDB, set it in the PDB.

   For example, in SQL*Plus:

   ```
   ALTER SYSTEM SET TDE_CONFIGURATION="KEYSTORE_CONFIGURATION=FILE";
   ```

2. Set the `WALLET_ROOT` static initialization parameter to the ASM disk group location followed by the `DB_UNIQUE_NAME` initialization parameter value.

   The inclusion of the value of `DB_UNIQUE_NAME` is necessary to allow the database server to automate the creation of the necessary directories under this location.

   You must not use `OMF` as a directory component of the `WALLET_ROOT` location (unlike in the standalone database configuration section).

   For example:

   ```
   WALLET_ROOT=+disk_group_name/db_unique_name
   ```

   This setting locates the TDE wallet that is used by the root and by all of the united mode PDBs in the *`WALLET_ROOT`*`/`*`db_unique_name`*`/tde` directory (that is, in `+`*`disk_group_name`*`/`*`db_unique_name`*`/tde`).

This setting locates the TDE wallet which is used by each isolated mode PDB in the *WALLET_ROOT*/*db_unique_name*/*PDB_GUID*/tde directory (that is, in +*disk_group_name*/*db_unique_name*/*PDB_GUID*/tde).

Configuring a Keystore to Point to an ASM Location When the WALLET_ROOT Location Does Not Follow OMF Guidelines

If the chosen WALLET_ROOT location does not comply with the Oracle Managed File (OMF) guidelines, then the Oracle database cannot perform automation of the directory creation.

In this case, you must use the ALTER DISKGROUP command to manually create the necessary directories under the WALLET_ROOT location. You must use the ALTER DISKGROUP ... ADD DIRECTORY statement to manually create the necessary directories, because no automation of the directory creation is possible when the WALLET_ROOT parameter is not using an OMF-compliant value.

1. Ensure that the KEYSTORE_CONFIGURATION attribute of the TDE_CONFIGURATION dynamic initialization parameter is set to FILE.

   For example, in SQL*Plus:

   ```
   ALTER SYSTEM SET TDE_CONFIGURATION="KEYSTORE_CONFIGURATION=FILE";
   ```

2. In the init.ora file, set the WALLET_ROOT static initialization parameter to the ASM disk group location.

   For example, the following path after *disk_group_name* contains no uppercase OMF directory elements:

   ```
   WALLET_ROOT="+disk_group_name/mydir/wallets"
   ```

3. Connect to the database instance using the SYSASM administrative privilege.

   ```
   connect / as sysasm
   ```

4. Execute the ALTER DISKGROUP statements to create the necessary directories.

   You must perform this step because the database server cannot automate the creation of these directories, since the location chosen for WALLET_ROOT is not compliant with the Oracle Managed Files guideline (that is, it does not have the OMF component included in it in uppercase letters).

   For example:

   ```
   ALTER DISKGROUP "disk_group_name" ADD DIRECTORY
       '+disk_group_name/mydir/wallets/tde';
   ```

   For a multitenant environment, find the PDB GUID of the PDB that will store the keystore, as follows:

   ```
   SELECT GUID FROM DBA_PDBS WHERE PDB_NAME = 'pdb name';
   ```

   Next, include the PDB GUID in the following ALTER DISKGROUP statements to create the necessary directories for the isolated mode PDB within the WALLET_ROOT location. For example, assuming the GUID is 4756C705E52A8768E053F82DC40A5329:

   ```
   ALTER DISKGROUP "disk_group_name" ADD DIRECTORY
       '+disk_group_name/mydir/wallets/4756C705E52A8768E053F82DC40A5329'
   ```

```
ALTER DISKGROUP "disk_group_name" ADD DIRECTORY
    '+disk_group_name/mydir/wallets/4756C705E52A8768E053F82DC40A5329/tde';
```

Closing a Keystore

You can manually close software and hardware keystores.

- About Closing Keystores
  After you open a keystore, it remains open until you shut down the database instance.
- Closing a Software Keystore
  You can manually close password-based software keystores, auto-login software keystores, and local auto-login software keystores.
- Closing a Hardware Keystore
  To close a hardware keystore, you must use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEYSTORE CLOSE` clause.

About Closing Keystores

After you open a keystore, it remains open until you shut down the database instance.

When you restart the database instance, then auto-login and local auto-login software keystores automatically open when required (that is, when the TDE master encryption key must be accessed). However, software password-based and hardware keystores do not automatically open. You must manually open them again before you can use them.

When you close a software or hardware keystore, you disable all of the encryption and decryption operations on the database. Hence, a database user or application cannot perform any operation involving encrypted data until the keystore is reopened.

When you re-open a keystore after closing it, the keystore contents are reloaded back into the database. Thus, if the contents had been modified (such as during a migration), the database will have the latest keystore contents.

You can check if a keystore is closed by querying the `STATUS` column of the `V$ENCRYPTION_WALLET` view.

The following data operations will fail if the keystore is not accessible:

- `SELECT` data from an encrypted column
- `INSERT` data into on an encrypted column
- `CREATE` a table with encrypted columns
- `CREATE` an encrypted tablespace

Closing a Software Keystore

You can manually close password-based software keystores, auto-login software keystores, and local auto-login software keystores.

In the case of an auto-login keystore, which opens automatically when it is accessed, manually close it if you moved it to a new location. You do this if you are changing your configuration from an auto-login keystore to a password-based keystore: you move out the auto-login keystore, and then close the auto-login keystore.

1. Log in to the database instance as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.

   For example:

   ```
   sqlplus sec_admin as syskm
   Enter password: password
   Connected.
   ```

2. Run the `ADMINISTER KEY MANAGEMENT` SQL statement.

   - For a password-based software keystore, use the following syntax:

     ```
     ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE
     [IDENTIFIED BY [EXTERNAL STORE | software_keystore_password]];
     ```

     In this specification:

     - `IDENTIFIED BY` can be one of the following:
       - `EXTERNAL STORE` uses the keystore password stored in the external store to perform the keystore operation.
       - *`software_keystore_password`* is the password of the user who created the keystore.
   - For an auto-login or local auto-login software keystore, use the following SQL statement:

     ```
     ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE;
     ```

     You do not need to specify a password for this statement.

Closing a keystore disables all of the encryption and decryption operations. Any attempt to encrypt or decrypt data or access encrypted data results in an error.

Closing a Hardware Keystore

To close a hardware keystore, you must use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEYSTORE CLOSE` clause.

1. Log into the database instance as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.

   For example:

   ```
   sqlplus sec_admin as syskm
   Enter password: password
   Connected.
   ```

2. Close the hardware keystore by using this syntax:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE
   IDENTIFIED BY [EXTERNAL STORE | "hardware_keystore_credentials"];
   ```

   In this specification:

   - `IDENTIFIED BY` can be one of the following settings:
     - `EXTERNAL STORE` uses the keystore password stored in the external store to perform the keystore operation.
     - *`hardware_keystore_credentials`* refers to the credentials for either an HSM or an Oracle Key Vault hardware keystore. For an HSM, specify the

credentials using this format, enclosed in quotation marks and separating the components with a colon: *"user_name:password"*, with *user_name* being the user who created the HSM and password being this user's password. For Oracle Key Vault, enter only the password of the user who created the keystore. Enclose this password with quotation marks.

For example:

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE
IDENTIFIED BY "psmith:password";
```

Backup and Recovery of Encrypted Data

For software keystores, you cannot access encrypted data without the TDE master encryption key.

Because the TDE master encryption key is stored in the keystore, you should periodically back up the software keystore in a secure location. You must back up a copy of the keystore whenever you set a new TDE master encryption key or perform any operation that writes to the keystore.

Do not back up the software keystore in the same location as the encrypted data. Back up the software keystore separately. This is especially true when you use the auto-login keystore, which does not require a password to open. In case the backup tape is lost, a malicious user should not be able to get both the encrypted data and the keystore.

Oracle Recovery Manager (Oracle RMAN) does not back up the software keystore as part of the database backup. When using a media manager such as Oracle Secure Backup with Oracle RMAN, Oracle Secure Backup automatically excludes auto-open keystores (the `cwallet.sso` files). However, it does not automatically exclude encryption keystores (the `ewallet.p12` files). It is a good practice to add the following `exclude` data set statement to your Oracle Secure Backup configuration:

```
exclude name *.p12
```

This setting instructs Oracle Secure Backup to exclude the encryption keystore from the backup set.

If you lose the software keystore that stores the TDE master encryption key, then you can restore access to encrypted data by copying the backed-up version of the keystore to the appropriate location. If you archived the restored keystore after the last time that you reset the TDE master encryption key, then you do not need to take any additional action.

If the restored software keystore does not contain the most recent TDE master encryption key, then you can recover old data up to the point when the TDE master encryption key was reset by rolling back the state of the database to that point in time. All of the modifications to encrypted columns after the TDE master encryption key was reset are lost.

Related Topics

- *Oracle Database Backup and Recovery User's Guide*

Dangers of Deleting Keystores

Oracle strongly recommends that you do not delete keystores.

If a keystore becomes overly full, any TDE master encryption key other than the currently-active TDE master encryption key can be moved to a new keystore to reduce the overall size of the keystore, but it is important to keep a backup of this new keystore because even though the keys have been moved out of the currently-active keystore, they may still be needed by other Oracle features, such as Oracle Recovery Manager backup operations. (See Related Topics at the end of this topic for a listing of features that are affected by deleted keystores.)

Deleting a keystore that still contains keys is particularly dangerous if you have configured Transparent Data Encryption and the keystore is in use. You can find if a keystore is in use by querying the `STATUS` column of the `V$ENCRYPTION_WALLET` view after you open the keystore. How you should proceed depends on whether you are using united mode or isolated mode.

- In isolated mode, if the `STATUS` column of the `V$ENCRYPTION_WALLET` is `OPEN_NO_MASTER_KEY`, then it is safe to archive and later delete the this keystore, because there are no keys in it.
- In united mode, you must execute the query of `V$ENCRYPTION_WALLET` from the `CDB$ROOT`, not the PDB. If you execute the query in a PDB that does not yet have a key set, then the `STATUS` is `OPEN_NO_MASTER_KEY`. However, this can be misleading, because a key could have been set in the `CDB$ROOT`. After you execute the query in the root and if the `STATUS` is `OPEN_NO_MASTER_KEY`, then you can safely archive and later delete the keystore.

The reason that you should be cautious when moving keys out of the currently-active TDE keystore is that this keystore may contain keys that are still needed by the database (even though the TDE master encryption key has been rekeyed). Deleting the keystore deletes these keys, and could result in the loss of encrypted data. Even if you decrypted all of the data in your database, you still should not delete the keystore, because doing so could still hamper the normal functioning of the Oracle database. This is because a TDE master encryption key in the keystore can also be required for other Oracle Database features. (See Related Topics at the end of this topic for a listing of features that are affected by deleted keystores.)

Even after you performed TDE keystore migration (which rekeys in such a way that the location of your currently-active TDE master encryption key changes place between your software keystore and your hardware keystore), you still should not delete your original keystore. The keys in the original keystore may be needed at a later time (for example, when you must recover an offline encrypted tablespace). Even if all online tablespaces are not encrypted, the key may still be in use.

The exception is in the case of software auto-login (or local auto-login) keystores. If you do not want to use this type of keystore, then ideally you should move it to a secure directory. Only delete an auto-login keystore if you are sure that it was created from a specific password-based keystore because an auto-login keystore is always based on an ordinary software keystore. The keystore should be available and known.

If you must delete a keystore, then do so with great caution. You must first move the keys within the keystore to a new keystore by using the `ADMINISTER KEY MANAGEMENT MOVE KEYS TO NEW KEYSTORE` statement.

Related Topics

- Features That Are Affected by Deleted Keystores
  Some features can be adversely affected if a keystore is deleted and a TDE master encryption key residing in that keystore is later needed.

- Moving a TDE Master Encryption Key into a New Keystore
  The `ADMINISTER KEY MANAGEMENT MOVE KEYS` moves an existing TDE master encryption key into a new keystore from an existing keystore.
- Moving a TDE Master Encryption Key into a New Keystore in United Mode
  In united mode, you can move an existing TDE master encryption key into a new keystore from an existing software password keystore.
- Moving a TDE Master Encryption Key into a New Keystore in Isolated Mode
  In isolated mode, you can move an existing TDE master encryption key into a new keystore from an existing software password keystore.

Features That Are Affected by Deleted Keystores

Some features can be adversely affected if a keystore is deleted and a TDE master encryption key residing in that keystore is later needed.

Before you delete a keystore, consider the impact that the deletion will have in the event that you need the any TDE master encryption key in the TDE keystore at a later time. The following features and activities are affected:

- Offlined tablespace operations
- Oracle Secure Backup operations
- Media recovery and block media recovery operations
- Point-in-time recovery operations
- Physical and logical Oracle Data Guard standby operations
- Golden Gate operations
- Oracle Streams operations
- Oracle Recovery Manager operations, including restoring Oracle Recovery Manager backups
- Applying archived redo logs to a database during database crash recovery operations
- Database online block recovery. (Online block recovery implies that the database is still open. Deleting a wallet in an open database with encrypted tablespaces will cause additional problems other than those associated with online block recovery.) These problems can include the following:
  - Encrypted online data in encrypted tablespaces would no longer be decrypted. Encrypted metadata in the `SYSTEM`, `UNDO`, and `TEMP` tablespaces would no longer be decrypted. You will no longer have control over what metadata is encrypted or where that metadata can reside.
  - Buffered data or metadata needs to be encrypted before it can be written back to the disk, but if the wallet is deleted, then the buffered data or metadata would no longer be encrypted. This could cause redo generation to fail, and the DBWR background process would not be able to write the data, which would possibly lead to a database instance hang or crash.
  - After a database instance crash, the database instance recovery and database crash recovery would fail, leading to the database not being able to be restarted.

Related Topics

- Dangers of Deleting Keystores
  Oracle strongly recommends that you do not delete keystores.

Managing the TDE Master Encryption Key

You can manage the TDE master encryption key in several ways.

- Creating User-Defined TDE Master Encryption Keys
  You can create a user-defined TDE master encryption key outside the database by generating a TDE master encryption key ID.
- Creating TDE Master Encryption Keys for Later Use
  You can create a TDE master encryption key that can be activated at a later date.
- Activating TDE Master Encryption Keys
  After you activate a TDE master encryption key, it can be used.
- TDE Master Encryption Key Attribute Management
  TDE master encryption key attributes store information about the TDE master encryption key.
- Creating Custom TDE Master Encryption Key Attributes for Reports
  Custom TDE master encryption key attributes enable you to defined attributes that are specific to your needs.
- Setting or Rekeying the TDE Master Encryption Key in the Keystore
  You can set or rekey the TDE master encryption key for both software keystores and hardware keystores.
- Exporting and Importing the TDE Master Encryption Key
  You can export and import the TDE master encryption key in different ways.
- Moving TDE Master Encryption Keys into a New Keystore
  You can move an existing TDE master encryption key into a new keystore from an existing password-protected keystore.
- Management of TDE Master Encryption Keys Using Oracle Key Vault
  You can use Oracle Key Vault to manage and share TDE master encryption keys across an enterprise.

Creating User-Defined TDE Master Encryption Keys

You can create a user-defined TDE master encryption key outside the database by generating a TDE master encryption key ID.

- About User-Defined TDE Master Encryption Keys
  A TDE master encryption key that is outside the database has its own user-generated ID, which tracks the use of the TDE master encryption key.
- Creating a User-Defined TDE Master Encryption Key
  To create a user-defined TDE master encryption key, use the `ADMINISTER KEY MANAGEMENT` statement with the `SET | CREATE [ENCRYPTION] KEY` clause.

About User-Defined TDE Master Encryption Keys

A TDE master encryption key that is outside the database has its own user-generated ID, which tracks the use of the TDE master encryption key.

You can use the ADMINISTER KEY MANAGEMENT to create and set user-defined TDE master encryption key IDs. After you generate the TDE master encryption key, you can bring this key into the database. Optionally, you can specify the TDE master encryption key ID in various ADMINISTER KEY MANAGEMENT statements.

This type of configuration benefits Oracle Fusion SaaS Cloud environments in that it enables you to generate a TDE master encryption key this complies with your site's requirements. This key that you generate supports the current encryption algorithms and can be used for software keystores.

After you generate the TDE master encryption key ID, you can encrypt your data as you normally would.

The TDE master encryption key and its corresponding ID will not be captured by any auditing logs.

Creating a User-Defined TDE Master Encryption Key

To create a user-defined TDE master encryption key, use the ADMINISTER KEY MANAGEMENT statement with the SET | CREATE [ENCRYPTION] KEY clause.

1. Log in to the database instance as a user who has been granted the ADMINISTER KEY MANAGEMENT or SYSKM privilege.

   For example:

   ```
   sqlplus sec_admin as syskm
   Enter password: password
   Connected.
   ```

2. Create the user-defined TDE master encryption key by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT SET | CREATE [ENCRYPTION] KEY
   '[mkid:mk | mk]'
   [USING ALGORITHM 'algorithm']
   [FORCE KEYSTORE]
   IDENTIFIED BY [EXTERNAL STORE | keystore_password]
   [WITH BACKUP [USING 'backup_identifier']];
   ```

   In this specification:

   - SET | CREATE : Enter SET if you want to create the master and activate the TDE master encryption key now, or select CREATE if you want to create the key for later use, without activating it yet.
   - *mkid* and *mk*:
     - *mkid*, the TDE master encryption key ID, is a 16–byte hex-encoded value that you can specify or have Oracle Database generate.
     - *mk*, the TDE master encryption key, is a hex-encoded value that you can specify or have Oracle Database generate, either 32 bytes (for the for AES256, ARIA256, and GOST256 algorithms) or 16 bytes (for the SEED128 algorithm).

     If you omit the *mkid*:*mk*|*mkid* clause but include the *mk* value, then Oracle Database generates the *mkid* for the *mk*.

     If you omit the entire *mkid*:*mk*|*mkid* clause, then Oracle Database generates these values for you.
   - USING ALGORITHM: Specify one of the following supported algorithms:

  - AES256
  - ARIA256
  - SEED128
  - GOST256

  If you omit the algorithm, then the default, AES256, is used.

- FORCE KEYSTORE temporarily opens the password-protected keystore for this operation. You must open the keystore for this operation.

The following example creates a TDE master encryption key without the key-ID is when a database administrator performs that operation:

First, generate the TDE master encryption key with OpenSSL:

```
$ openssl rand 32 -hex
f24ce8cdae39742b5da7293da206fbd05a28b1e44c4781f801768cfdc3dbb6e2
```

This use this key in the following ADMINISTER KEY MANAGEMENT statement:

```
ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY
'f24ce8cdae39742b5da7293da206fbd05a28b1e44c4781f801768cfdc3dbb6e2'
USING ALGORITHM 'GOST256'
FORCE KEYSTORE
IDENTIFIED BY keystore_password;
```

The next example creates user-defined keys for both the master encryption ID and the TDE master encryption key. It omits the algorithm specification, so the default algorithm AES256 is used.

```
ADMINISTER KEY MANAGEMENT CREATE ENCRYPTION KEY
'10203040506070801112131415161718:3D432109DF88967A541967062A6F4E460E892318E30
7F017BA048707B402493C'
IDENTIFIED BY keystore_password;
```

The next scenario, where TDE master encryption key and key-ID are generated outside of the database, shows how to support separation of duties. A key administrator inserts the key with a key_ID into the wallet by using the ADMINISTER KEY MANAGEMENT CREATE [ENCRYPTION] KEY statement. Then, the key administrator sends the key_ID to the database administrator, who then activates the key by using the ADMINISTER KEY MANAGEMENT USE KEY clause.

a. Create the key-ID using OpenSSL, converting characters to upper case:

   ```
   $ openssl rand 16 | xxd -u -p
   D20765EB721AF44D054B30FE87F8E49A
   ```

b. Create the TDE master encryption key.

   ```
   $ openssl rand 32 -hex
   5a089c8ea6ee21cba774f61e58d102665df4a979a34757836b3d066a3eb3db11
   ```

c. Use both strings to insert the new TDE master encryption key, using the default of AES256, into the wallet.

   ```
   ADMINISTER KEY MANAGEMENT CREATE ENCRYPTION KEY
   'D20765EB721AF44D054B30FE87F8E49A:5a089c8ea6ee21cba774f61e58d102665df4a97
   9a34757836b3d066a3eb3db11'
   IDENTIFIED BY keystore_password;
   ```

d. Because the database internally stores the `key_ID` in BASE64 format, the key administrator must convert the HEX `key_ID` to a BASE64 key-ID, using this syntax:

```
$ echo "01hex key_ID followed by 44 zeros" | xxd -r -p | base64"
```

For example:

```
$ echo
"01D20765EB721AF44D054B30FE87F8E49A0000000000000000000000000000000000000000
000000"
```

e. After the key administrator sent the `key_ID` to the database administrator, the database administrator then executes the following statement:

```
ADMINISTER KEY MANAGEMENT USE ENCRYPTION KEY
'AdIHZetyGvRNBUsw/of45JoAAAAAAAAAAAAAAAAAAAAAAAAAAAA'
IDENTIFIED BY keystore_password;
```

Alternatively, the database administrator can retrieve the BASE64 key-ID from the database with the following `SELECT` statement:

```
SELECT ' ADMINISTER KEY MANAGEMENT USE KEY '''||KEY_ID||'''
FORCE KEYSTORE IDENTIFIED BY KEYSTORE_PASSWORD WITH BACKUP;'
AS "USE KEY COMMAND" FROM V$ENCRYPTION_KEYS
WHERE CREATOR_PDBNAME = SYS_CONTEXT('USERENV', 'CON_NAME')
ORDER BY CREATION_TIME DESC FETCH FIRST 1 ROWS ONLY;
```

Related Topics

- Supported Encryption and Integrity Algorithms
  The supported Advanced Encryption Standard cipher keys, including tablespace and database encryption keys, can be either 128, 192, or 256 bits long. Tablespace and database encryption use the 128–bit length cipher key.

Creating TDE Master Encryption Keys for Later Use

You can create a TDE master encryption key that can be activated at a later date.

- About Creating a TDE Master Encryption Key for Later Use
  The `CREATE KEY` clause of the `ADMINISTER KEY MANAGEMENT` statement can create a TDE master encryption key to be activated at a later date.
- Creating a TDE Master Encryption Key for Later Use
  A keystore must be opened before you can create a TDE master encryption key for use later on.
- Example: Creating a TDE Master Encryption Key in a Single Database
  You can use the `ADMINISTER KEY MANAGEMENT CREATE KEY USING TAG` statement to create a TDE master encryption key in a single database.

About Creating a TDE Master Encryption Key for Later Use

The `CREATE KEY` clause of the `ADMINISTER KEY MANAGEMENT` statement can create a TDE master encryption key to be activated at a later date.

You then can activate this key on the same database or export it to another database and activate it there.

This method of TDE master encryption key creation is useful in a multitenant environment when you must re-create the TDE master encryption keys. The `CREATE KEY` clause enables you to use a single SQL statement to generate a new TDE master encryption key for all of the PDBs within a multitenant environment. The creation time of the new TDE master encryption key is later than the activation of the TDE master encryption key that is currently in use. Hence, the creation time can serve as a reminder to all of the PDBs to activate the most recently created TDE master encryption key as soon as possible.

Creating a TDE Master Encryption Key for Later Use

A keystore must be opened before you can create a TDE master encryption key for use later on.

1. Log in to the database instance as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.

   For example:

   ```
   sqlplus sec_admin as syskm
   Enter password: password
   Connected.
   ```

2. Create the TDE master encryption key by using this syntax:

   ```
   ADMINISTER KEY MANAGEMENT CREATE KEY [USING TAG 'tag']
   [FORCE KEYSTORE]
   IDENTIFIED BY [EXTERNAL STORE | keystore_password]
   [WITH BACKUP [USING 'backup_identifier']];
   ```

   In this specification:

   - *tag* is the associated attribute and information that you define. Enclose this setting in single quotation marks (' ').
   - `FORCE KEYSTORE` temporarily opens the password-protected keystore for this operation. You must open the keystore for this operation.
   - `IDENTIFIED BY` can be one of the following settings:
     - `EXTERNAL STORE` uses the keystore password stored in the external store to perform the keystore operation.
     - *keystore_password* is the mandatory keystore password that you used when you created the original keystore. It is case sensitive.
   - `WITH BACKUP` backs up the TDE master encryption key in the same location as the key, as identified by the `WRL_PARAMETER` column of the `V$ENCRYPTION_WALLET` view. To find the key locations for all of the database instances, query the `GV$ENCRYPTION_WALLET` view.

     You must back up password-based software keystores. You do not need to back up auto-login or local auto-login software keystores. Optionally, include the `USING` *backup_identifier* clause to add a description of the backup. Enclose *backup_identifier* in single quotation marks (' ').

3. If necessary, activate the TDE master encryption key.

   a. Fin the key ID.

      ```
      SELECT KEY_ID FROM V$ENCRYPTION_KEYS;
      ```

```
KEY_ID
------------------------------------------------
AWsHwVYC2U+Nv3RVphn/yAIAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

b. Use this key ID to activate the key.

```
ADMINISTER KEY MANAGEMENT USE KEY
'AWsHwVYC2U+Nv3RVphn/yAIAAAAAAAAAAAAAAAAAAAAAAAAAAA'
USING TAG 'quarter:second;description:Activate Key on standby'
IDENTIFIED BY password
WITH BACKUP;
```

Related Topics

- Step 3: Open the Software Keystore
  Depending on the type of keystore you create, you must manually open the keystore before you can use it.
- Opening a Hardware Keystore
  To open a hardware keystore, use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEYSTORE OPEN` clause.
- Activating TDE Master Encryption Keys
  After you activate a TDE master encryption key, it can be used.

Example: Creating a TDE Master Encryption Key in a Single Database

You can use the `ADMINISTER KEY MANAGEMENT CREATE KEY USING TAG` statement to create a TDE master encryption key in a single database.

Example 4-2 shows how to create a TDE master encryption key in a single database. After you run this statement, a TDE master encryption key with the tag definition is created in the keystore for that database. You can query the `TAG` column of the `V$ENCRYPTION_KEYS` view for the identifier of the newly created key. You can query the `CREATION_TIME` column to find the most recently created key, which would be the key that you created from this statement. You can export this key to another database if you want or activate it locally later on, as described in Activating TDE Master Encryption Keys.

Example 4-2 Creating a TDE Master Encryption Key in a Single Database

```
ADMINISTER KEY MANAGEMENT CREATE KEY USING TAG
'source:admin@source;target:db1@target'
IDENTIFIED BY password WITH BACKUP;

keystore altered.
```

Activating TDE Master Encryption Keys

After you activate a TDE master encryption key, it can be used.

- About Activating TDE Master Encryption Keys
  You can activate a previously created or imported TDE master encryption key by using the `USE KEY` clause of `ADMINSTER KEY MANAGEMENT`.
- Activating a TDE Master Encryption Key
  To activate a TDE master encryption key, you must open the keystore and use `ADMINISTER KEY MANAGEMENT` with the `USE KEY` clause.

- Example: Activating a TDE Master Encryption Key
  You can use the `ADMINISTER KEY MANAGEMENT` SQL statement to activate a TDE master encryption key.

About Activating TDE Master Encryption Keys

You can activate a previously created or imported TDE master encryption key by using the `USE KEY` clause of `ADMINSTER KEY MANAGEMENT`.

After you activate the key, it is available for use. The key will be used to protect all of the column keys and all of the tablespace encryption keys. If you have deployed a logical standby database, then you must export the TDE master encryption keys after recreating them, and then import them into the standby database. You can have the TDE master encryption key in use on both the primary and the standby databases. To do so, you must activate the TDE master encryption key after you import it to the logical standby database.

Activating a TDE Master Encryption Key

To activate a TDE master encryption key, you must open the keystore and use `ADMINISTER KEY MANAGEMENT` with the `USE KEY` clause.

1. Log in to the database instance as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.

   For example:

   ```
   sqlplus sec_admin as syskm
   Enter password: password
   Connected.
   ```

2. Query the `KEY_ID` column of the `V$ENCRYPTION_KEYS` view to find the key identifier.

   For example:

   ```
   SELECT KEY_ID FROM V$ENCRYPTION_KEYS;

   KEY_ID
   ----------------------------------------------------
   ARaHD762tUkkvyLgPzAi6hMAAAAAAAAAAAAAAAAAAAAAAAAAAAA
   ```

3. Use this key identifier to activate the TDE master encryption key by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT USE KEY 'key_identifier_from_V$ENCRYPTION_KEYS'
   [USING TAG 'tag']
   [FORCE KEYSTORE]
   IDENTIFIED BY [EXTERNAL STORE | keystore_password]
   [WITH BACKUP [USING 'backup_identifier']];
   ```

   In this specification:

   - *key_identifier_from_V$ENCRYPTION_KEYS* is the key identifie. Enclose this setting in single quotation marks (' ').
   - *tag* is the associated attributes and information that you define. Enclose this setting in single quotation marks (' ').
   - `FORCE KEYSTORE` temporarily opens the password-protected keystore for this operation. You must open the keystore for this operation.

- `IDENTIFIED BY` can be one of the following settings:
  - `EXTERNAL STORE` uses the keystore password stored in the external store to perform the keystore operation.
  - *`keystore_password`* is the mandatory keystore password that you used when you created the original keystore.
- `WITH BACKUP` backs up the TDE master encryption key in the same location as the key, as identified by the `WRL_PARAMETER` column of the `V$ENCRYPTION_WALLET` view. To find the key locations for all of the database instances, query the `GV$ENCRYPTION_WALLET` view.

  You must back up password-based software keystores. You do not need to back up auto-login or local auto-login software keystores. Optionally, include the `USING` *`backup_identifier`* clause to add a description of the backup. Enclose `backup_identifier` in single quotation marks (`' '`).

Related Topics

- Opening a Software Keystore
  To open a software keystore, you must use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEYSTORE OPEN` clause.
- Opening a Hardware Keystore
  To open a hardware keystore, use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEYSTORE OPEN` clause.

Example: Activating a TDE Master Encryption Key

You can use the `ADMINISTER KEY MANAGEMENT` SQL statement to activate a TDE master encryption key.

Example 4-3 shows how to activate a previously imported TDE master encryption key and then update its tag. This key is activated with the current database time stamp and time zone.

Example 4-3 Activating a TDE Master Encryption Key

```
ADMINISTER KEY MANAGEMENT USE KEY
'ARaHD762tUkkvyLgPzAi6hMAAAAAAAAAAAAAAAAAAAAAAAAAAAAA'
USING TAG 'quarter:second;description:Activate Key on standby'
IDENTIFIED BY password WITH BACKUP;

keystore altered.
```

In this version of the same operation, the `FORCE KEYSTORE` clause is added in the event that the auto-login keystore is in use, or if the keystore is closed. The password of the keystore is stored externally, so the `EXTERNAL STORE` setting is used for the `IDENTIFIED BY` clause.

```
ADMINISTER KEY MANAGEMENT USE KEY
'ARaHD762tUkkvyLgPzAi6hMAAAAAAAAAAAAAAAAAAAAAAAAAAAAA'
USING TAG 'quarter:second;description:Activate Key on standby'
FORCE KEYSTORE IDENTIFIED BY EXTERNAL STORE WITH BACKUP;

keystore altered.
```

TDE Master Encryption Key Attribute Management

TDE master encryption key attributes store information about the TDE master encryption key.

- TDE Master Encryption Key Attributes
  TDE master encryption key attributes include detailed information about the TDE master encryption key.
- Finding the TDE Master Encryption Key That Is in Use
  A TDE master encryption key that is in use is the encryption key that was activated most recently for the database.

TDE Master Encryption Key Attributes

TDE master encryption key attributes include detailed information about the TDE master encryption key.

The information contains the following types:

- **Key time stamp information:** Internal security policies and compliance policies usually determine the key rekeying frequency. You should expire keys when they reach the end of their lifetimes and then generate new keys. Time stamp attributes such as key creation time and activation time help you to determine the key age accurately, and automate key generation.

  The `V$ENCRYPTION_KEYS` view includes columns such as `CREATION_TIME` and `ACTIVATION_TIME`. See *Oracle Database Reference* for a complete description of the `V$ENCRYPTION_KEYS` view.
- **Key owner information:** Key owner attributes help you to determine the user who created or activated the key. These attributes can be important for security, auditing, and tracking purposes. Key owner attributes also include key use information, such as whether the key is used for standalone TDE operations or used in a multitenant environment.

  The `V$ENCRYPTION_KEYS` view includes columns such as `CREATOR, CREATOR_ID, USER, USER_ID,` and `KEY_USE`.
- **Key source information:** Keys often must be moved between databases for operations such as import-export operations and Data Guard-related operations. Key source attributes enable you to track the origin of each key. You can track whether a key was created locally or imported, and the database name and instance number of the database that created the key. In a multitenant environment, you can track the PDB where the key was created.

  The `V$ENCRYPTION_KEYS` view includes columns such as `CREATOR_DBNAME, CREATOR_DBID, CREATOR_INSTANCE_NAME, CREATOR_INSTANCE_NUMBER, CREATOR_PDBNAME,` and so on.
- **Key usage information:** Key usage information determines the database or PDB where the key is being used. It also helps determine whether a key is in active use or not.

  The `V$ENCRYPTION_KEYS` view includes columns such as `ACTIVATING_DBNAME, ACTIVATING_DBID, ACTIVATING_INSTANCE_NAME, ACTIVATING_PDBNAME,` and so on.

- **User-defined information and other information:** When creating a key, you can tag it with information using the `TAG` option. Each key contains important information such as whether or not it has been backed up.

  The `V$ENCRYPTION_KEYS` view includes columns such as `KEY_ID, TAG,` and other miscellaneous columns, for example `BACKED_UP`.

> **Note:**
>
> TDE Master Key Attributes and Tag are only supported with a hardware security module that has PKCS#11 data object support.

Finding the TDE Master Encryption Key That Is in Use

A TDE master encryption key that is in use is the encryption key that was activated most recently for the database.

- To find the TDE master encryption key, query the `V$ENCRYPTION_KEYS` dynamic view.

  For example:

```
SELECT KEY_ID
FROM V$ENCRYPTION_KEYS
WHERE ACTIVATION_TIME = (SELECT MAX(ACTIVATION_TIME)
                         FROM V$ENCRYPTION_KEYS
                         WHERE ACTIVATING_DBID = (SELECT DBID FROM
V$DATABASE));
```

Creating Custom TDE Master Encryption Key Attributes for Reports

Custom TDE master encryption key attributes enable you to defined attributes that are specific to your needs.

- About Creating Custom Attribute Tags
  Attribute tags enable you to monitor specific activities users perform, such as accessing a particular terminal ID.
- Creating a Custom Attribute Tag
  To create a custom attribute tag, you must use the `SET TAG` clause of the `ADMINISTER KEY MANAGEMENT` statement.

About Creating Custom Attribute Tags

Attribute tags enable you to monitor specific activities users perform, such as accessing a particular terminal ID.

By default, Oracle Database defines a set of attributes that describe various characteristics of the TDE master encryption keys that you create, such as the creation time, database in which the TDE master encryption key is used, and so on. These attributes are captured by the `V$ENCRYPTION_KEY` dynamic view.

You can create custom attributes that can be captured by the `TAG` column of the `V$ENCRYPTION_KEYS` dynamic view. This enables you to define behaviors that you may want to monitor, such as users who perform activities on encryption keys. The tag can encompass multiple attributes, such as session IDs from a specific terminal.

After you create the tag for a TDE master encryption key, its name should appear in the TAG column of the V$ENCRYPTION_KEYS view for that TDE master encryption key. If you create a tag for the secret, then the tag appears in the SECRET_TAG column of the V$CLIENT_SECRETS view. If you create a secret with a tag, then the tag appears in the SECRET_TAG column of the V$CLIENT_SECRETS view.

Creating a Custom Attribute Tag

To create a custom attribute tag, you must use the SET TAG clause of the ADMINISTER KEY MANAGEMENT statement.

1. Log in to the database instance as a user who has been granted the ADMINISTER KEY MANAGEMENT or SYSKM privilege.

   For example:

   ```
   sqlplus sec_admin as syskm
   Enter password: password
   Connected.
   ```

2. If necessary, query the TAG column of the V$ENCRYPTION_KEY dynamic view to find a listing of existing tags for the TDE master encryption keys.

   When you create a new tag for a TDE master encryption key, it overwrites the existing tag for that TDE master encryption key.

3. Create the custom attribute tag by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT SET TAG 'tag'
   FOR 'master_key_identifier'
   [FORCE KEYSTORE]
   IDENTIFIED BY [EXTERNAL STORE | keystore_password]
   [WITH BACKUP [USING 'backup_identifier']];
   ```

   In this specification

   - *tag* is the associated attributes or information that you define. Enclose this information in single quotation marks (' ').
   - *master_key_identifier* identifies the TDE master encryption key for which the *tag* is set. To find a list of TDE master encryption key identifiers, query the KEY_ID column of the V$ENCRYPTION_KEYS dynamic view.
   - FORCE KEYSTORE temporarily opens the password-protected keystore for this operation. You must open the keystore for this operation.
   - IDENTIFIED BY can be one of the following settings:
     - EXTERNAL STORE uses the keystore password stored in the external store to perform the keystore operation.
     - *keystore_password* is the password that was used to create the keystore.
   - *backup_identifier* defines the tag values. Enclose this setting in single quotation marks (' ') and separate each value with a colon.

   For example, to create a tag that uses two values, one to capture a specific session ID and the second to capture a specific terminal ID:

   ```
   ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY
   USING TAG 'sessionid=3205062574:terminal=xcvt'
   IDENTIFIED BY keystore_password
   WITH BACKUP;
   ```

```
keystore altered.
```

Both the session ID (`3205062574`) and terminal ID (`xcvt`) can derive their values by using either the `SYS_CONTEXT` function with the `USERENV` namespace, or by using the `USERENV` function.

Setting or Rekeying the TDE Master Encryption Key in the Keystore

You can set or rekey the TDE master encryption key for both software keystores and hardware keystores.

- About Setting or Rekeying the TDE Master Encryption Key in the Keystore
  You can set or rekey the TDE master encryption key for both software password-based and hardware keystores.
- Creating, Tagging, and Backing Up a TDE Master Encryption Key
  The `ADMINISTER KEY MANAGEMENT` statement enables you to create, tag, and back up a TDE master encryption key.
- About Rekeying the TDE Master Encryption Key
  Oracle Database uses a unified TDE Master Encryption Key for both TDE column encryption and TDE tablespace encryption.
- Rekeying the TDE Master Encryption Key
  You can use the `ADMINISTER KEY MANAGEMENT` statement to rekey a TDE master encryption key.
- Changing the TDE Master Encryption Key for a Tablespace
  You can use the `ENCRYPT` and `REKEY` clauses of the `ALTER TABLESPACE` statement to encrypt a tablespace.

About Setting or Rekeying the TDE Master Encryption Key in the Keystore

You can set or rekey the TDE master encryption key for both software password-based and hardware keystores.

The TDE master encryption key is stored in an external security module (keystore), and it is used to protect the TDE table keys and tablespace encryption keys. By default, the TDE master encryption key is a system-generated random value created by Transparent Data Encryption (TDE).

Use the `ADMINISTER KEY MANAGEMENT` statement to set or reset (`REKEY`) the TDE master encryption key. When the master encryption key is set, then TDE is considered enabled and cannot be disabled.

Before you can encrypt or decrypt database columns or tablespaces, you must generate a TDE master encryption key. Oracle Database uses the same TDE master encryption key for both TDE column encryption and TDE tablespace encryption. The instructions for setting a software or hardware TDE master encryption key explain how to generate a tDE master encryption key.

Related Topics

- Step 4: Set the TDE Master Encryption Key in the Software Keystore
  Once the keystore is open, you can set a TDE master encryption key for it.

- Step 4: Set the Hardware Keystore TDE Master Encryption Key
  After you have opened the hardware keystore, you are ready to set the hardware keystore TDE master encryption key.

Creating, Tagging, and Backing Up a TDE Master Encryption Key

The `ADMINISTER KEY MANAGEMENT` statement enables you to create, tag, and back up a TDE master encryption key.

1. Log in to the database instance as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.

   For example:

   ```
   sqlplus sec_admin as syskm
   Enter password: password
   Connected.
   ```

2. Create and back up the TDE master encryption key, and apply a tag, by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY
   [USING TAG 'tag']
   [FORCE KEYSTORE]
   IDENTIFIED BY [EXTERNAL STORE | keystore_password]
   WITH BACKUP [USING 'backup_identifier'];
   ```

   In this specification:

   - *tag* is the tag that you want to create. Enclose this tag in single quotation marks (`' '`).
   - `FORCE KEYSTORE` temporarily opens the password-protected keystore for this operation. You must open the keystore for this operation.
   - `IDENTIFIED BY` can be one of the following settings:
     - `EXTERNAL STORE` uses the keystore password stored in the external store to perform the keystore operation.
     - *keystore_password* is either *software_keystore_password* or *hardware_keystore_credentials*. As with software passwords, it is case sensitive. You must enclose the password string in double quotation marks (`" "`). Separate *user_name* and *password* with a colon (`:`).
   - `WITH BACKUP` backs the TDE master encryption key up in the same location as the key, as identified by the `WRL_PARAMETER` column of the `V$ENCRYPTION_WALLET` view. To find the `WRL_PARAMETER` values for all of the database instances, query the `GV$ENCRYPTION_WALLET` view.

     You must back up password-based software keystores. You do not need to use it for auto-login or local auto-login software keystores. Optionally, include the `USING` *backup_identifier* clause to add a description of the backup. Enclose this identifier in single quotation marks (`' '`).

   For example:

   ```
   ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY
   USING TAG 'backups"
   IDENTIFIED BY password
   WITH BACKUP USING 'hr.emp_key_backup';
   ```

```
keystore altered.
```

Oracle Database uses the keystore in the keystore location specified by the WALLET_ROOT parameter in the initialization parameter file to store the TDE master encryption key.

Related Topics

- Creating Custom TDE Master Encryption Key Attributes for Reports
  Custom TDE master encryption key attributes enable you to defined attributes that are specific to your needs.
- About the Keystore Location in the sqlnet.ora File
  Oracle recommends that you use the WALLET_ROOT and TDE_CONFIGURATION parameters to configure the keystore location, but an alternative is to configure the sqlnet.ora file.

About Rekeying the TDE Master Encryption Key

Oracle Database uses a unified TDE Master Encryption Key for both TDE column encryption and TDE tablespace encryption.

When you rekey the TDE master encryption key for TDE column encryption, the TDE Master Encryption Key for TDE tablespace encryption also is rekeyed. Rekey the TDE Master Encryption Key only if it was compromised or as per the security policies of the organization. This process deactivates the previous TDE master encryption key.

For better security and to meet compliance regulations, periodically rekey the TDE master encryption key. This process deactivates the previous TDE master encryption key, creates a new TDE master encryption key, and then activates it. You can check the keys that were created recently by querying the CREATION_TIME column in the V$ENCRYPTION_KEYS view. To find the keys that were activated recently, query the ACTIVATION_TIME column in the V$ENCRYPTION_KEYS view.

You cannot change the TDE master encryption key or rekey a TDE master encryption key for an auto-login keystore. Because auto-login keystores do not have a password, an administrator or a privileged user can change the keys without the knowledge of the security officer. However, if both the auto-login and the password-based keystores are present in the configured location (as set in the sqlnet.ora file), then when you rekey the TDE master encryption key, a TDE master encryption key is added to both the auto-login and password-based keystores. If the auto-login keystore is in use in a location that is different from that of the password-based keystore, then you must re-create the auto-login keystore.

Do not perform a rekey operation of the master key concurrently with an online tablespace rekey operation. You can find if an online tablespace is in the process of being TDE Master Encryption Keyed by issuing the following query:

```
SELECT TS#,ENCRYPTIONALG,STATUS FROM V$ENCRYPTED_TABLESPACES;
```

A status of REKEYING means that the corresponding tablespace is still being rekeyed.

> **Note:**
>
> You cannot add new information to auto-login keystores separately.

Rekeying the TDE Master Encryption Key

You can use the `ADMINISTER KEY MANAGEMENT` statement to rekey a TDE master encryption key.

1. Log in to the database instance as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.

   For example:

   ```
   sqlplus sec_admin as syskm
   Enter password: password
   Connected.
   ```

2. If you are rekeying the TDE master encryption key for a keystore that has auto login enabled, then ensure that both the auto login keystore, identified by the `.sso` file, and the encryption keystore, identified by the `.p12` file, are present.

   You can find the location of these files by querying the `WRL_PARAMETER` column of the `V$ENCRYPTION_WALLET` view. To find the `WRL_PARAMETER` values for all of the database instances, query the `GV$ENCRYPTION_WALLET` view.

3. Rekey the TDE master encryption key by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT SET [ENCRYPTION] KEY
   [FORCE KEYSTORE]
   [USING TAG 'tag_name']
   IDENTIFIED BY [EXTERNAL STORE | keystore_password]
   [WITH BACKUP [USING 'backup_identifier']];
   ```

   In this specification:

   - *tag* is the associated attributes and information that you define. Enclose this setting in single quotation marks (`' '`).
   - `FORCE KEYSTORE` temporarily opens the password-protected keystore for this operation. You must open the keystore for this operation.
   - `IDENTIFIED BY` can be one of the following settings:
     - `EXTERNAL STORE` uses the keystore password stored in the external store to perform the keystore operation.
     - *keystore_password* is the mandatory keystore password that you created when you created the keystore in Step 2: Create the Software Keystore.
   - `WITH BACKUP` creates a backup of the keystore. You must use this option for password-based and hardware keystores. Optionally, you can use the `USING` clause to add a brief description of the backup. Enclose this description in single quotation marks (`' '`). This identifier is appended to the named keystore file (for example, `ewallet_`*time-stamp*`_emp_key_backup.p12`). Follow the file naming conventions that your operating system uses.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT SET KEY
   FORCE KEYSTORE
   IDENTIFIED BY keystore_password
   WITH BACKUP USING 'emp_key_backup';

   keystore altered.
   ```

Related Topics

- Step 3: Open the Software Keystore
  Depending on the type of keystore you create, you must manually open the keystore before you can use it.
- Step 3: Open the Hardware Keystore
  After you have configured the hardware security module, you must open the hardware keystore before it can be used.

Changing the TDE Master Encryption Key for a Tablespace

You can use the `ENCRYPT` and `REKEY` clauses of the `ALTER TABLESPACE` statement to encrypt a tablespace.

1. Ensure that the tablespace open in read-write mode.

   You can query the `STATUS` column of the `V$INSTANCE` dynamic view to find if a database is open and the `OPEN_MODE` column of the `V$DATABASE` view to find if it in read-write mode.

2. If necessary, open the database in read-write mode.

   ```
   ALTER DATABASE OPEN READ WRITE;
   ```

3. Run the `ALTER TABLESPACE` SQL statement to encrypt the tablespace.

   If the tablespace has not yet been encrypted, then use the `ENCRYPT` clause:

   ```
   ALTER TABLESPACE encrypt_ts ENCRYPTION USING 'AES256' ENCRYPT;
   ```

   To change the encryption of the `SYSTEM`, `SYSAUX`, or `UNDO` tablespace, you must rekey the tablespace online. Use the `ONLINE` and `REKEY` clauses. For example, to change the encryption algorithm of the `SYSTEM` tablespace:

   ```
   ALTER TABLESPACE SYSTEM ENCRYPTION ONLINE USING 'AES256' REKEY;
   ```

Related Topics

- About Encryption Conversion for Existing Online Tablespaces
  You can encrypt, decrypt, or rekey existing user tablespaces, and the `SYSTEM`, `SYSAUX`, and `UNDO` tablespace when they are online.

Exporting and Importing the TDE Master Encryption Key

You can export and import the TDE master encryption key in different ways.

- About Exporting and Importing the TDE Master Encryption Key
  Oracle Database features such as transportable tablespaces and Oracle Data Pump move data that is possibly encrypted between databases.
- About Exporting TDE Master Encryption Keys
  You can use `ADMINISTER KEY MANAGEMENT EXPORT` to export TDE master encryption keys from a keystore, and then import them into another keystore.
- Exporting a TDE Master Encryption Key
  The `ADMINISTER KEY MANAGEMENT` statement with the `EXPORT [ENCRYPTION] KEYS WITH SECRET` clause exports a TDE master encryption key.

- Example: Exporting a TDE Master Encryption Key by Using a Subquery
  The `ADMINISTER KEY MANAGEMENT EXPORT ENCRYPTION KEYS` statement can export a TDE master encryption key by using a subquery.
- Example: Exporting a List of TDE Master Encryption Key Identifiers to a File
  The `ADMINISTER KEY MANAGEMENT EXPORT ENCRYPTION KEYS WITH SECRET` statement can export a list of TDE master encryption key identifiers to a file.
- Example: Exporting All TDE Master Encryption Keys of the Database
  The `ADMINISTER KEY MANAGEMENT EXPORT ENCRYPTION KEYS` SQL statement can export all TDE master encryption keys of a database.
- About Importing TDE Master Encryption Keys
  The `ADMINISTER KEY MANAGEMENT IMPORT` statement can import exported TDE master encryption keys from a key export file into a target keystore.
- Importing a TDE Master Encryption Key
  The `ADMINISTER KEY MANAGEMENT` statement with the `IMPORT [ENCRYPTION] KEYS WITH SECRET` clause can import a TDE master encryption key.
- Example: Importing a TDE Master Encryption Key
  You can use the `ADMINISTER KEY MANAGEMENT IMPORT KEYS` SQL statement to import a TDE master encryption key.
- How Keystore Merge Differs from TDE Master Encryption Key Export or Import
  The keystore merge operation differs from the TDE master encryption key export and import operations.

About Exporting and Importing the TDE Master Encryption Key

Oracle Database features such as transportable tablespaces and Oracle Data Pump move data that is possibly encrypted between databases.

These are some common scenarios in which you can choose to export and import TDE master encryption keys to move them between source and target keystores. For Data Guard (Logical Standby), you must copy the keystore that is in the primary database to the standby database. Instead of merging the primary database keystore with the standby database, you can export the TDE master encryption key that is in use and then import it to the standby database. Moving transportable tablespaces that are encrypted between databases requires that you export the TDE master encryption key at the source database and then import it into the target database.

About Exporting TDE Master Encryption Keys

You can use `ADMINISTER KEY MANAGEMENT EXPORT` to export TDE master encryption keys from a keystore, and then import them into another keystore.

A TDE master encryption key is exported together with its key identifier and key attributes. The exported keys are protected with a password (secret) in the export file.

You can specify the TDE master encryption keys to be exported by using the `WITH IDENTIFIER` clause of the `ADMINSITER KEY MANAGENT EXPORT` statement. To export the TDE master encryption keys, you can either specify their key identifiers as a comma-separated list, or you can specify a query that enumerates their key identifiers. Be aware that Oracle Database executes the query determining the key identifiers within the current user's rights and not with definer's rights.

If you omit the `WITH IDENTIFER` clause, then all of the TDE master encryption keys of the database are exported.

Related Topics

- Exporting and Importing Master Encryption Keys for a PDB in Isolated Mode
  In isolated mode, the `EXPORT` and `IMPORT` clauses of `ADMINISTER KEY MANAGEMENT EXPORT` can export or import master encryption keys for a PDB.
- Exporting and Importing Master Encryption Keys for a PDB in Isolated Mode
  In isolated mode, the `EXPORT` and `IMPORT` clauses of `ADMINISTER KEY MANAGEMENT EXPORT` can export or import master encryption keys for a PDB.

Exporting a TDE Master Encryption Key

The `ADMINISTER KEY MANAGEMENT` statement with the `EXPORT [ENCRYPTION] KEYS WITH SECRET` clause exports a TDE master encryption key.

1. Log in to the database instance as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.

   For example:

   ```
   sqlplus sec_admin as syskm
   Enter password: password
   Connected.
   ```

2. Export the TDE master encryption keys by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT EXPORT [ENCRYPTION] KEYS
   WITH SECRET "export_secret"
   TO 'file_path'
   [FORCE KEYSTORE]
   IDENTIFIED BY [EXTERNAL STORE | keystore_password]
   [WITH IDENTIFIER IN 'key_id1', 'key_id2', 'key_idn' | (SQL_query)];
   ```

   In this specification:

   - *export_secret* is a password that you can specify to encrypt the export the file that contains the exported keys. Enclose this secret in double quotation marks (" "), or you can omit the quotation marks if the secret has no spaces.
   - *file_path* is the complete path and name of the file to which the keys must be exported. Enclose this path in single quotation marks (' '). You can export to regular file systems only.
   - `FORCE KEYSTORE` temporarily opens the password-protected keystore for this operation. You must open the keystore for this operation.
   - `IDENTIFIED BY` can be one of the following settings:
     - `EXTERNAL STORE` uses the keystore password stored in the external store to perform the keystore operation.
     - *software_keystore_password* is the password of the keystore containing the keys.
   - *key_id1*, *key_id2*, *key_idn* is a string of one or more TDE master encryption key identifiers for the TDE master encryption key being exported. Separate each key identifier with a comma and enclose each of these key identifiers in single quotation marks (' '). To find a list of TDE master encryption key identifiers, query the `KEY_ID` column of the `V$ENCRYPTION_KEYS` dynamic view.

- *SQL_query* is a query that fetches a list of the TDE master encryption key identifiers. It should return only one column which contains the TDE master encryption key identifiers. This query is executed with current user rights.

Related Topics

- Step 3: Open the Software Keystore
  Depending on the type of keystore you create, you must manually open the keystore before you can use it.

Example: Exporting a TDE Master Encryption Key by Using a Subquery

The `ADMINISTER KEY MANAGEMENT EXPORT ENCRYPTION KEYS` statement can export a TDE master encryption key by using a subquery.

Example 4-5 shows how to export TDE master encryption keys whose identifiers are fetched by a query to a file called `export.exp`. The TDE master encryption keys in the file are encrypted using the secret `my_secret`. The `SELECT` statement finds the identifiers for the TDE master encryption keys to be exported.

Example 4-4 Exporting a List of TDE Master Encryption Key Identifiers to a File

```
ADMINISTER KEY MANAGEMENT EXPORT ENCRYPTION KEYS
WITH SECRET "my_secret"
TO '/TDE/export.exp'
FORCE KEYSTORE
IDENTIFIED BY password
WITH IDENTIFIER IN 'AdoxnJ0uH08cv7xkz83ovwsAAAAAAAAAAAAAAAAAAAAAAAAAAAA',
'AW5z3CoyKE/yv3cNT5CWCXUAAAAAAAAAAAAAAAAAAAAAAAAAAAAA';

keystore altered.
```

Example: Exporting a List of TDE Master Encryption Key Identifiers to a File

The `ADMINISTER KEY MANAGEMENT EXPORT ENCRYPTION KEYS WITH SECRET` statement can export a list of TDE master encryption key identifiers to a file.

Example 4-4 shows how to export TDE master encryption keys by specifying their identifiers as a list, to a file called `export.exp`. TDE master encryption keys in the file are encrypted using the secret `my_secret`. The identifiers of the TDE master encryption key to be exported are provided as a comma-separated list.

Example 4-5 Exporting TDE Master Encryption Key Identifiers by Using a Subquery

```
ADMINISTER KEY MANAGEMENT EXPORT ENCRYPTION KEYS
WITH SECRET "my_secret" TO '/etc/TDE/export.exp'
FORCE KEYSTORE
IDENTIFIED BY password
WITH IDENTIFIER IN (SELECT KEY_ID FROM V$ENCRYPTION_KEYS WHERE ROWNUM <3);

keystore altered.
```

Example: Exporting All TDE Master Encryption Keys of the Database

The `ADMINISTER KEY MANAGEMENT EXPORT ENCRYPTION KEYS` SQL statement can export all TDE master encryption keys of a database.

Example 4-6 shows how to export all of the TDE master encryption keys of the database to a file called `export.exp`. The TDE master encryption keys in the file are encrypted using the secret `my_secret`.

Example 4-6 Exporting All of the TDE Master Encryption Keys of the Database

```
ADMINISTER KEY MANAGEMENT EXPORT ENCRYPTION KEYS
WITH SECRET "my_secret" TO '/etc/TDE/export.exp'
FORCE KEYSTORE
IDENTIFIED BY password;

keystore altered.
```

About Importing TDE Master Encryption Keys

The `ADMINISTER KEY MANAGEMENT IMPORT` statement can import exported TDE master encryption keys from a key export file into a target keystore.

You cannot re-import TDE master encryption keys that have already been imported.

Related Topics

- Exporting and Importing Master Encryption Keys for a PDB in Isolated Mode
  In isolated mode, the `EXPORT` and `IMPORT` clauses of `ADMINISTER KEY MANAGEMENT EXPORT` can export or import master encryption keys for a PDB.

Importing a TDE Master Encryption Key

The `ADMINISTER KEY MANAGEMENT` statement with the `IMPORT [ENCRYPTION] KEYS WITH SECRET` clause can import a TDE master encryption key.

1. Log in to the database instance as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.

   ```
   sqlplus sec_admin as syskm
   Enter password: password
   Connected.
   ```

2. Run the following SQL statement:

   ```
   ADMINISTER KEY MANAGEMENT IMPORT [ENCRYPTION] KEYS
   WITH SECRET "import_secret"
   FROM 'file_name'
   [FORCE KEYSTORE]
   IDENTIFIED BY [EXTERNAL STORE | keystore_password]
   [WITH BACKUP [USING 'backup_identifier']];
   ```

   In this specification:

   - *import_secret* is the same password that was used to encrypt the keys during the export operation. Enclose this secret in double quotation marks (" "), or you can omit the quotation marks if the secret has no spaces.
   - *file_name* is the complete path and name of the file from which the keys need to be imported. Enclose this setting in single quotation marks (' ').
   - `FORCE KEYSTORE` temporarily opens the password-protected keystore for this operation. You must open the keystore for this operation.
   - `IDENTIFIED BY` can be one of the following settings:

  - `EXTERNAL STORE` uses the keystore password stored in the external store to perform the keystore operation.
  - *`software_keystore_password`* is the password of the software keystore where the keys are being imported.
- `WITH BACKUP` must be used in case the target keystore was not backed up before the import operation. *`backup_identifier`* is an optional string that you can provide to identify the keystore backup. Enclose this setting in single quotation marks (`' '`).

Related Topics

- Step 3: Open the Software Keystore
  Depending on the type of keystore you create, you must manually open the keystore before you can use it.

Example: Importing a TDE Master Encryption Key

You can use the `ADMINISTER KEY MANAGEMENT IMPORT KEYS` SQL statement to import a TDE master encryption key.

Example 4-7 shows how to import the TDE master encryption key identifiers that are stored in the file `export.exp` and encrypted with the secret `my_secret`.

Example 4-7 Importing TDE Master Encryption Key Identifiers from an Export File

```
ADMINISTER KEY MANAGEMENT IMPORT KEYS
WITH SECRET "my_secret"
FROM '/etc/TDE/export.exp'
FORCE KEYSTORE
IDENTIFIED BY password WITH BACKUP;

keystore altered.
```

How Keystore Merge Differs from TDE Master Encryption Key Export or Import

The keystore merge operation differs from the TDE master encryption key export and import operations.

Even though both the `ADMINISTER KEY MANAGEMENT MERGE` statement and the `ADMINISTER KEY MANAGEMENT EXPORT` and `IMPORT` statements eventually move the TDE master encryption keys from one keystore to the next, there are differences in how these two statements function.

- The `MERGE` statement merges two keystores whereas the `EXPORT` and `IMPORT` statements export the keys to a file or import the keys from a file. The keystore is different from the export file, and the two cannot be used interchangeably. The export file is not a keystore and cannot be configured to be used with a database as a keystore. Similarly, the `IMPORT` statement cannot extract the TDE master encryption keys from the keystore.
- The `MERGE` statement merges all of the TDE master encryption keys of the specified keystores where as the `EXPORT` and `IMPORT` statements can be selective.
- The `EXPORT` and `IMPORT` statements require the user to provide both a location (`filepath`) and the file name of the export file, whereas the `MERGE` statement only takes in the location of the keystores.

- The file name of the keystores is fixed and is determined by the `MERGE` operation and can be either `ewallet.p12` or `cwallet.sso`. The file names for the export files used in the `EXPORT` the `IMPORT` statements are specified by the user.
- The keystores on Automatic Storage Management (ASM) disk groups or regular file systems can be merged with `MERGE` statements. The export files used in the `EXPORT` and the `IMPORT` statements can only be a regular operating system file and cannot be located on an ASM disk group.
- The keystores merged using the `MERGE` statement do not need to be configured or in use with the database. The `EXPORT` statement can only export the keys from a keystore that is configured and in use with the database and is also open when the export is done. The `IMPORT` statement can only import the keys into a keystore that is open, configured, and in use with the database.
- The `MERGE` statement never modifies the metadata associated with the TDE master encryption keys. The `EXPORT` and `IMPORT` operations can modify the metadata of the TDE master encryption keys when required, such as during a PDB plug operation.

Moving TDE Master Encryption Keys into a New Keystore

You can move an existing TDE master encryption key into a new keystore from an existing password-protected keystore.

- About Moving TDE Master Encryption Keys into a New Keystore
  You can move an unused (and safely archived) TDE master encryption key into a new keystore.
- Moving a TDE Master Encryption Key into a New Keystore
  The `ADMINISTER KEY MANAGEMENT MOVE KEYS` moves an existing TDE master encryption key into a new keystore from an existing keystore.

About Moving TDE Master Encryption Keys into a New Keystore

You can move an unused (and safely archived) TDE master encryption key into a new keystore.

Use great caution when you decide to execute `ADMINISTER KEY MANAGEMENT MOVE KEYS`. Even though this statement will not move an active master encryption key, it can still affect keys that are necessary for a range of database features. If you have deleted a key, then the data that was encrypted by that key is rendered permanently inaccessible (equivalent to deleting the data) for these features to be used. See Related Topics at the end of this topic for more information about features that are affected by deleted keystores.

Therefore, before you move the keystore, it is very important that you safely archive it. Delete the keystore only after a period of time has passed, to ensure that the keystore is no longer really useful.

To move a TDE master encryption key into a new keystore, you use the `ADMINISTER KEY MANAGEMENT MOVE KEYS` statement. This statement does not move the active TDE master encryption key (that is, the key that is currently in use at the time that `ADMINISTER KEY MANAGEMENT MOVE KEYS` is issued) because the database is currently using it.

If you mistakenly use the `ADMINISTER KEY MANAGEMENT MOVE KEYS` statement instead of `ADMINISTER KEY MANAGEMENT MERGE KEYSTORE` when you are configuring a new TDE keystore (for example, when you are changing the TDE keystore configuration from one where the software keystore is located in the operating system's file system to one where the software keystore is located in Oracle Automatic Storage Management (Oracle ASM)), then the following symptoms may help you to identify the TDE misconfiguration that was introduced by the use of the wrong key management command:

- When you open the TDE keystore that was the target of the earlier `ADMINISTER KEY MANAGEMENT MOVE KEYS` operation, an `ORA-28374: typed master key not found in wallet` error is seen, because the active TDE master encryption key was not moved to that keystore.
- The value shown in the `STATUS` column of the `V$ENCRYPTION_WALLET` view is `OPEN_NO_MASTER_KEY` after you open the new keystore. The `OPEN_NO_MASTER_KEY` status is expected, because the new TDE keystore that was mistakenly populated by means of the `ADMINISTER KEY MANAGEMENT MOVE KEYS` statement does not contain the active TDE master encryption key.

Related Topics

- Dangers of Deleting Keystores
  Oracle strongly recommends that you do not delete keystores.
- Features That Are Affected by Deleted Keystores
  Some features can be adversely affected if a keystore is deleted and a TDE master encryption key residing in that keystore is later needed.

Moving a TDE Master Encryption Key into a New Keystore

The `ADMINISTER KEY MANAGEMENT MOVE KEYS` moves an existing TDE master encryption key into a new keystore from an existing keystore.

This feature enables you to delete unused keystores. Before you move the keystore, ensure that it is safely archived. After you move the encryption key to a new keystore, and when you are sure that the old keystore is no longer needed, you then can delete the old keystore.

1. Log in to the database instance as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.

   For example:

   ```
   sqlplus sec_admin as syskm
   Enter password: password
   Connected.
   ```

2. Query the `KEY_ID` column of the `V$ENCRYPTION_KEYS` view to find the key identifier of the keys you want to move to the "key archival" keystore.

   For example:

   ```
   SELECT KEY_ID FROM V$ENCRYPTION_KEYS;

   KEY_ID
   ------------------------------------------------
   ARaHD762tUkkvyLgPzAi6hMAAAAAAAAAAAAAAAAAAAAAAAAAAAA
   ```

3. Move the key into a new keystore by using the following syntax:

```
ADMINISTER KEY MANAGEMENT
MOVE [ENCRYPTION] KEYS
TO NEW KEYSTORE 'keystore_location1'
IDENTIFIED BY keystore1_password
FROM [FORCE] KEYSTORE
IDENTIFIED BY keystore_password
[WITH IDENTIFIER IN
{ 'key_identifier' [, 'key_identifier' ]... | ( subquery ) } ]
[WITH BACKUP [USING 'backup_identifier'] ];
```

In this specification:

- *keystore_location1* is the path to the `wallet` directory that will store the new keystore `.p12` file. By default, this directory is in `$ORACLE_BASE/admin/`*db_unique_name*`/wallet`.
- *keystore1_password* is the password for the keystore from which the new keystore is moved.
- *keystore_password* is the password for the keystore from which the key is moving.
- *key_identifier* is the key identifier that you find from querying the `KEY_ID` column of the `V$ENCRYPTION_KEYS` view. Enclose this setting in single quotation marks (`' '`).
- *subquery* can be used to find the exact key identifier that you want.
- *backup_identifier* is an optional description of the backup. Enclose *backup_identifier* in single quotation marks (`' '`).

For example:

```
ADMINISTER KEY MANAGEMENT MOVE KEYS
TO NEW KEYSTORE $ORACLE_BASE/admin/orcl/wallet
IDENTIFIED BY keystore_password
FROM FORCE KEYSTORE
IDENTIFIED BY keystore_password
WITH IDENTIFIER IN
(SELECT KEY_ID FROM V$ENCRYPTION_KEYS WHERE ROWNUM < 2);
```

4. To delete the old keystore, go to the `wallet` directory and then manually delete the `.p12` file containing the keystore.

   To find the location of the keystore, open the keystore, and then query the `WRL_PARAMETER` column of the `V$ENCRYPTION_WALLET` view.

Related Topics

- Dangers of Deleting Keystores
  Oracle strongly recommends that you do not delete keystores.
- Features That Are Affected by Deleted Keystores
  Some features can be adversely affected if a keystore is deleted and a TDE master encryption key residing in that keystore is later needed.

Management of TDE Master Encryption Keys Using Oracle Key Vault

You can use Oracle Key Vault to manage and share TDE master encryption keys across an enterprise.

Oracle Key Vault securely stores the keys in a central repository, along with other security objects such as credential files and Java keystores, and enables you to share these objects with other TDE-enabled databases.

Related Topics

- Migration of Keystores to and from Oracle Key Vault
  You can use Oracle Key Vault to migrate both software and hardware keystores to and from Oracle Key Vault.
- *Oracle Key Vault Administrator's Guide*

Storing Oracle Database Secrets

Secrets are data that support internal Oracle Database features that integrate external clients such as Oracle GoldenGate into the database.

- About Storing Oracle Database Secrets in a Keystore
  Keystores can store secrets that support internal Oracle Database features and integrate external clients such as Oracle GoldenGate.
- Storage of Oracle Database Secrets in a Software Keystore
  The `ADMINISTER KEY MANAGEMENT ADD SECRET|UPDATE SECRET|DELETE SECRET` statements can add secrets, update secrets, and delete secrets from a keystore.
- Example: Adding an HSM Password to a Software Keystore
  The `ADMINISTER KEY MANAGEMENT ADD SECRET` statement can add an HSM password to a software keystore.
- Example: Changing an HSM Password Stored as a Secret in a Software Keystore
  The `ADMINISTER KEY MANAGEMENT UPDATE SECRET` statement can change an HSM password that is stored as a secret in a software keystore.
- Example: Deleting an HSM Password Stored as a Secret in a Software Keystore
  The `ADMINISTER KEY MANAGEMENT DELETE SECRET` statement can delete HSM passwords that are stored as secrets in a software keystore.
- Storage of Oracle Database Secrets in a Hardware Keystore
  The `ADMINISTER KEY MANAGEMENT ADD SECRET|UPDATE SECRET|DELETE SECRET` statements can add, update, and delete secrets.
- Example: Adding an Oracle Database Secret to a Hardware Keystore
  The `ADMINISTER KEY MANAGEMENT ADD SECRET` statement can add an Oracle Database secret to a hardware keystore.
- Example: Changing an Oracle Database Secret in a Hardware Keystore
  The `ADMINISTER KEY MANAGEMENT MANAGEMENT UPDATE SECRET` statement can change an Oracle Database secret in a hardware keystore.
- Example: Deleting an Oracle Database Secret in a Hardware Keystore
  The `ADMINISTER KEY MANAGEMENT DELETE SECRET FOR CLIENT` statement can delete an Oracle Database secret that is in a hardware keystore.
- Configuring Auto-Login Hardware Security Modules
  A hardware security module can be configured to use the auto-login capability.

About Storing Oracle Database Secrets in a Keystore

Keystores can store secrets that support internal Oracle Database features and integrate external clients such as Oracle GoldenGate.

The secret key must be a string adhering to Oracle identifier rules. You can add, update, or delete a client secret in an existing keystore. The Oracle GoldenGate Extract process must have data encryption keys to decrypt the data that is in data files and in REDO or UNDO logs. Keys are encrypted with shared secrets when you share the keys between an Oracle database and an Oracle GoldenGate client. The software keystore stores the shared secrets.

Depending on your site's requirements, you may require automated open keystore operations even when a hardware security module is configured. For this reason, the hardware security module password can be stored in a software auto-login keystore, which enables the auto-login capability for the hardware security module. The Oracle Database side can also store the credentials for the database to log in to an external storage server in the software keystore.

You can store Oracle Database secrets in both software keystores and hardware keystores:

- **Software keystores:** You can store secrets in software password-based, auto-login, and local auto-login software keystores. If you want to store secrets in an auto-login (or auto-login local) keystore, then note the following:
  - If the software auto-login keystore is in the same location as its corresponding password-based software keystore, then the secrets are added automatically.
  - If the software auto-login keystore is in a different location from its corresponding password-based software keystore, then you must create the auto-login keystore again from the password-based keystore, and keep the two keystores in synchronization.
- **Hardware keystores:** You can store secrets in standard hardware security modules.

Related Topics

- Storage of Oracle Database Secrets in a Hardware Keystore
  The ADMINISTER KEY MANAGEMENT ADD SECRET|UPDATE SECRET|DELETE SECRET statements can add, update, and delete secrets.
- Configuring Auto-Login Hardware Security Modules
  A hardware security module can be configured to use the auto-login capability.

Storage of Oracle Database Secrets in a Software Keystore

The ADMINISTER KEY MANAGEMENT ADD SECRET|UPDATE SECRET|DELETE SECRET statements can add secrets, update secrets, and delete secrets from a keystore.

As with all of the ADMINISTER KEY MANAGEMENT statements, you must have the ADMINISTER KEY MANAGEMENT or the SYSKM administrative privilege. To find information about existing secrets, you can query the V$CLIENT_SECRETS dynamic view.

- **Adding a secret:** Use the following syntax:

```
ADMINISTER KEY MANAGEMENT
ADD SECRET 'secret' FOR CLIENT 'client_identifier'
[USING TAG 'tag']
[TO [[LOCAL] AUTO_LOGIN] KEYSTORE keystore_location
[FORCE KEYSTORE]
IDENTIFIED BY [EXTERNAL STORE | keystore_password]
[WITH BACKUP [USING backup_id];
```

- **Updating a secret:** Use the following syntax:

```
ADMINISTER KEY MANAGEMENT
UPDATE SECRET 'secret' FOR CLIENT 'client_identifier'
[USING TAG 'tag']
 [TO [[LOCAL] AUTO_LOGIN] KEYSTORE keystore_location
[FORCE KEYSTORE]
IDENTIFIED BY [EXTERNAL STORE | keystore_password]
[WITH BACKUP [USING backup_id];
```

- **Deleting a secret:** Use the following syntax:

```
ADMINISTER KEY MANAGEMENT
DELETE SECRET FOR CLIENT 'client_identifier'
[FROM [[LOCAL] AUTO_LOGIN] KEYSTORE keystore_location
[FORCE KEYSTORE]
IDENTIFIED BY [EXTERNAL STORE | keystore_password]
[WITH BACKUP [USING backup_id];
```

In all of these statements, the specification is as follows:

- *secret* is the client secret key to be stored, updated, or deleted. Enclose this setting in single quotation marks (' ') or omit the quotation marks if the secret has no spaces. To find information about existing secrets and their client identifiers, query the V$CLIENT_SECRETS dynamic view.
- *client_identifier* is an alphanumeric string used to identify the secret key. *client_identifier* does not have a default value. Enclose this setting in single quotation marks (' ').
- TO [[LOCAL] AUTO_LOGIN] KEYSTORE refers to the location of an auto-login keystore, which is specified in the sqlnet.ora file.
- *tag* is an optional, user-defined description for the secret key to be stored. You can use *tag* with the ADD and UPDATE operations. Enclose this setting in single quotation marks (' '). This tag appears in the SECRET_TAG column of the V$CLIENT_SECRETS view.

  WITH BACKUP is required in case the keystore was not backed up before the ADD, UPDATE, or DELETE operation. *backup_identifier* is an optional user-defined description for the backup. Enclose *backup_identifier* in single quotation marks (' ').
- [TO | FROM [[LOCAL] AUTOLLOGIN] KEYSTORE specifies the location of the auto-login or password keystore, which is set in the sqlnet.ora file.
- FORCE KEYSTORE temporarily opens the password-protected keystore for this operation if an auto-login keystore is open (and in use) or if the keystore is closed.
- IDENTIFIED BY can be one of the following settings:
  - EXTERNAL STORE uses the keystore password stored in the external store to perform the keystore operation.
  - *keystore_password* is the password for the keystore.

Related Topics

- Creating Custom TDE Master Encryption Key Attributes for Reports
  Custom TDE master encryption key attributes enable you to defined attributes that are specific to your needs.

Example: Adding an HSM Password to a Software Keystore

The `ADMINISTER KEY MANAGEMENT ADD SECRET` statement can add an HSM password to a software keystore.

Example 4-8 shows how to add a hardware security module (HSM) password as a secret to a software keystore.

Example 4-8 Adding an Oracle Database Secret to a Software Keystore

```
ADMINISTER KEY MANAGEMENT
ADD SECRET 'psmith:password' FOR CLIENT 'HSM_PASSWORD'
USING TAG 'HSM credentials' FORCE KEYSTORE
IDENTIFIED BY password WITH BACKUP;
```

In this version, the keystore password is in an external store, so the `EXTERNAL STORE` setting is used for `IDENTIFIED BY`:

```
ADMINISTER KEY MANAGEMENT
ADD SECRET 'psmith:password' FOR CLIENT 'HSM_PASSWORD'
USING TAG 'HSM credentials' FORCE KEYSTORE
IDENTIFIED BY EXTERNAL STORE WITH BACKUP;
```

Example: Changing an HSM Password Stored as a Secret in a Software Keystore

The `ADMINISTER KEY MANAGEMENT UPDATE SECRET` statement can change an HSM password that is stored as a secret in a software keystore.

Example 4-9 shows how to change an HSM password that is stored as a secret in a software keystore.

Example 4-9 Changing an Oracle Database Secret to a Software Keystore

```
ADMINISTER KEY MANAGEMENT
UPDATE SECRET admin_password FOR CLIENT 'admin@myhost'
USING TAG 'new_host_credentials' FORCE KEYSTORE
IDENTIFIED BY software_keytore_password;
```

In this version, the password for the keystore is in an external store:

```
DMINISTER KEY MANAGEMENT
UPDATE SECRET admin_password FOR CLIENT 'admin@myhost'
USING TAG 'new_host_credentials' FORCE KEYSTORE
IDENTIFIED BY EXTERNAL STORE;
```

Example: Deleting an HSM Password Stored as a Secret in a Software Keystore

The `ADMINISTER KEY MANAGEMENT DELETE SECRET` statement can delete HSM passwords that are stored as secrets in a software keystore.

Example 4-10 shows how to delete an HSM password that is stored as a secret in the software keystore.

Example 4-10 Deleting an Oracle Database Secret in a Software Keystore

```
ADMINISTER KEY MANAGEMENT
DELETE SECRET FOR CLIENT 'HSM_PASSWORD'
FORCE KEYSTORE
IDENTIFIED BY password WITH BACKUP;
```

In this version, the password for the keystore is in an external store:

```
ADMINISTER KEY MANAGEMENT
DELETE SECRET FOR CLIENT 'HSM_PASSWORD'
FORCE KEYSTORE
IDENTIFIED BY EXTERNAL STORE WITH BACKUP;
```

Storage of Oracle Database Secrets in a Hardware Keystore

The `ADMINISTER KEY MANAGEMENT ADD SECRET|UPDATE SECRET|DELETE SECRET` statements can add, update, and delete secrets.

As with all `ADMINISTER KEY MANAGEMENT` statements, you must have the `ADMINISTER KEY MANAGEMENT` or the `SYSKM` administrative privilege. When you add, update, or delete secrets from a keystore that is in use or presently open, then you must run `ADMINISTER KEY MANAGEMENT` in the root.

You can store Oracle Database secrets in both HSM and Oracle Key Vault hardware keystores, but be aware that automatic logins do not work if you store the `HSM_PASSWORD` in a Key Vault keystore.

> **Note:**
>
> Before you attempt to add a secret to a hardware security module, ensure that it has PKCS#11 data object support.

- **Adding a secret:** Use the following syntax:

  ```
  ADMINISTER KEY MANAGEMENT ADD SECRET 'secret'
  FOR CLIENT 'client_identifier' [USING TAG 'tag']
  [TO [[LOCAL] AUTO_LOGIN] KEYSTORE keystore_location
  [FORCE KEYSTORE]
  IDENTIFIED BY "hardware_keystore_credentials"
  [WITH BACKUP [USING backup_id]];
  ```

- **Updating a secret:** Use the following syntax:

  ```
  ADMINISTER KEY MANAGEMENT UPDATE SECRET 'secret'
  FOR CLIENT 'client_identifier' [USING TAG 'tag']
  [TO [[LOCAL] AUTO_LOGIN] KEYSTORE keystore_location
  [FORCE KEYSTORE]
  IDENTIFIED BY "hardware_keystore_credentials"
  [WITH BACKUP [USING backup_id]];
  ```

- **Deleting a secret:** Use the following syntax:

  ```
  ADMINISTER KEY MANAGEMENT DELETE SECRET
  FOR CLIENT 'client_identifier'
  [FROM [[LOCAL] AUTO_LOGIN] KEYSTORE keystore_location
  [FORCE KEYSTORE]
  ```

```
IDENTIFIED BY "hardware_keystore_credentials"
[WITH BACKUP [USING backup_id]];
```

In all of these statements, the specification as follows:

- *secret* is the client secret key to be stored, updated, or deleted. Enclose this setting in double quotation marks (' ') or omit the quotation marks if the secret has no spaces. To find information about existing secrets and their client identifiers, query the V$CLIENT_SECRETS dynamic view.
- *client_identifier* is an alphanumeric string used to identify the secret key. *client_identifier* does not have a default value. Enclose this setting in single quotation marks (' ').
- *tag* is an optional, user-defined description for the secret key to be stored. You can use *tag* with the ADD and UPDATE operations. Enclose this setting in single quotation marks (' '). This tag appears in the SECRET_TAG column of the V$CLIENT_SECRETS view.
- [TO | FROM [[LOCAL] AUTO_LOGIN] KEYSTORE specifies the location of the keystore used for the hardware keystore.
- *hardware_keystore_credentials* refers to the credentials for either an HSM or an Oracle Key Vault hardware keystore. For an HSM, specify the credentials using this format, enclosed in quotation marks and separating the components with a colon: "*user_name*:*password*", with *user_name* being the user who created the HSM and password being this user's password. For Oracle Key Vault, enter only the password of the user who created the keystore. Enclose this password with quotation marks.

Related Topics

- Creating Custom TDE Master Encryption Key Attributes for Reports
  Custom TDE master encryption key attributes enable you to defined attributes that are specific to your needs.

Example: Adding an Oracle Database Secret to a Hardware Keystore

The ADMINISTER KEY MANAGEMENT ADD SECRET statement can add an Oracle Database secret to a hardware keystore.

Example 4-11 shows how to add a password for a user to a hardware keystore.

Example 4-11 Adding an Oracle Database Secret to a Hardware Keystore

```
ADMINISTER KEY MANAGEMENT ADD SECRET 'password'
FOR CLIENT 'admin@myhost' USING TAG 'myhost admin credentials'
IDENTIFIED BY "psmith:password";
```

In this version, the keystore password is in an external store, so the EXTERNAL STORE setting is used for IDENTIFIED BY:

```
ADMINISTER KEY MANAGEMENT ADD SECRET 'password'
FOR CLIENT 'admin@myhost' USING TAG 'myhost admin credentials'
IDENTIFIED BY EXTERNAL STORE;
```

Example: Changing an Oracle Database Secret in a Hardware Keystore

The `ADMINISTER KEY MANAGEMENT MANAGEMENT UPDATE SECRET` statement can change an Oracle Database secret in a hardware keystore.

Example 4-12 shows how to change a password that is stored as a secret in a hardware keystore.

Example 4-12 Changing an Oracle Database Secret in a Hardware Keystore

```
ADMINISTER KEY MANAGEMENT MANAGEMENT UPDATE SECRET 'password2'
FOR CLIENT 'admin@myhost' USING TAG 'New host credentials'
IDENTIFIED BY "psmith:password";
```

In this version, the password for the keystore is in an external store:

```
ADMINISTER KEY MANAGEMENT MANAGEMENT UPDATE SECRET 'password2'
FOR CLIENT 'admin@myhost' USING TAG 'New host credentials'
IDENTIFIED BY EXTERNAL STORE;
```

Example: Deleting an Oracle Database Secret in a Hardware Keystore

The `ADMINISTER KEY MANAGEMENT DELETE SECRET FOR CLIENT` statement can delete an Oracle Database secret that is in a hardware keystore.

Example 4-13 shows how to delete a hardware security module password that is stored as a secret in the hardware keystore.

Example 4-13 Deleting an Oracle Database Secret in a Hardware Keystore

```
ADMINISTER KEY MANAGEMENT DELETE SECRET FOR CLIENT 'admin@myhost'
IDENTIFIED BY "psmith:password";
```

In this version, the password for the keystore is in an external store:

```
ADMINISTER KEY MANAGEMENT DELETE SECRET FOR CLIENT 'admin@myhost'
IDENTIFIED BY EXTERNAL STORE;
```

Configuring Auto-Login Hardware Security Modules

A hardware security module can be configured to use the auto-login capability.

- About Configuring Auto-Login Hardware Security Modules
  An auto-login hardware security module stores the hardware security module credentials in an auto-login keystore.
- Configuring an Auto-Login Hardware Security Module
  To configure the auto-login HSM, you must use the `ADMINISTER KEY MANAGEMENT` statement to add or update a client secret that is to be used to authenticate to the HSM.

About Configuring Auto-Login Hardware Security Modules

An auto-login hardware security module stores the hardware security module credentials in an auto-login keystore.

This configuration reduces the security of the system as a whole. However, this configuration does support unmanned or automated operations and is useful in deployments where automatic re-login to the hardware security module is necessary.

Be aware that executing the query `SELECT * FROM V$ENCRYPTION_WALLET` will automatically open an auto-login hardware security module. For example, suppose you have an auto-login hardware security module configured. If you close the keystore and query the `V$ENCRYPTION_WALLET` view, then the output will indicate that a keystore is open. This is because `V$ENCRYPTION_WALLET` opened up the auto-login hardware keystore and then displayed the status of the auto-login keystore.

To enable the auto-login capability for a hardware keystore, you must store the hardware keystore's credentials in an auto-login wallet.

When you use the `ADMINISTER KEY MANAGEMENT` statement, there are conceptually two sets of commands that act on client secrets:

- `ADMINISTER KEY MANAGEMENT` commands that act on the wallet that is currently in use (in other words, a wallet that contains an active TDE master encryption key).
- `ADMINISTER KEY MANAGEMENT` commands that act on a wallet that is not currently being used to hold the active TDE master encryption key. Oracle recommends that you use this approach when you configure an auto-login hardware keystore.

Related Topics

- Configuring an Auto-Login Hardware Security Module
  To configure the auto-login HSM, you must use the `ADMINISTER KEY MANAGEMENT` statement to add or update a client secret that is to be used to authenticate to the HSM.

Configuring an Auto-Login Hardware Security Module

To configure the auto-login HSM, you must use the `ADMINISTER KEY MANAGEMENT` statement to add or update a client secret that is to be used to authenticate to the HSM.

Before you begin this procedure, ensure that you have configured the TDE hardware keystore.

In this procedure, the wallet that is created does not contain any keys. It only holds the client secret. So, when you query the `V$ENCRYPTION_WALLET` dynamic view for this wallet, the `STATUS` column shows `OPEN_NO_MASTER_KEY` rather than `OPEN`, because the wallet only contains the client secret.

1. Reconfigure the `WALLET_ROOT` parameter in the `init.ora` file to include the keystore location of the software keystore, if it is not already present.

   The software keystore location may already be present if you had previously migrated to using the HSM.

   For example:

   ```
   WALLET_ROOT=/etc/ORACLE/WALLETS/orcl
   ```

2. Add or update the secret in the software keystore.

   The secret is the hardware security module password and the client is the `HSM_PASSWORD`. `HSM_PASSWORD` is an Oracle-defined client name that is used to represent the HSM password as a secret in the software keystore.

For example:

```
ADMINISTER KEY MANAGEMENT ADD SECRET 'user_name:password'
FOR CLIENT 'HSM_PASSWORD'
TO LOCAL AUTO_LOGIN KEYSTORE software_keystore_location
WITH BACKUP;
```

In this example:

- *software_keystore_location* is the location of the software keystore within the WALLET_ROOT location that you just defined in Step 1.
  For the CDB root and for any PDB that is configured in united mode, the value to use for the *software_keystore_location* location is *WALLET_ROOT*/tde.

  For any PDB that is configured in isolated mode, the value to use for the *software_keystore_location* location is *WALLET_ROOT/pdb_guid*/tde. When you are in the PDB, execute the following query to find this GUID: SELECT GUID FROM V$PDBS;
- LOCAL creates a local auto-login wallet file, cwallet.sso, to hold the credentials for the HSM. This wallet is tied to the host on which it was created. For an Oracle Real Application Clusters environment, omit the LOCAL keyword, because each Oracle RAC node has a different host name, yet they all use the same HSM. If you configure a local auto-login wallet for the Oracle RAC instance, then only the first Oracle RAC node, where the cwallet.sso file was created, would be able to access the HSM credentials. If you try to open the keystore from another node instead of from that first node, there would be a problem auto-opening cwallet.sso, and so it would result in a failure to auto-open the auto-login HSM. This restriction applies if you are using a shared location to hold the cwallet.sso file for the Oracle RAC cluster, because using LOCAL only works if you have a separate cwallet.sso file (containing the same credentials) on each node of the Oracle RAC environment.

At this stage, the next time a TDE operation executes, the hardware security module auto-login keystore opens automatically. An example of a TDE operation is to query the V$ENCRYPTION_WALLET view, for example:

```
SELECT * FROM V$ENCRYPTION_WALLET;
```

Related Topics

- About the Keystore Location in the sqlnet.ora File
  Oracle recommends that you use the WALLET_ROOT and TDE_CONFIGURATION parameters to configure the keystore location, but an alternative is to configure the sqlnet.ora file.
- Configuring a Hardware Keystore
  A hardware keystore resides in a hardware security module (HSM), which is designed to store encryption keys.

Storing Oracle GoldenGate Secrets in a Keystore

You can store Oracle GoldenGate secrets in Transparent Data Encryption keystores.

- About Storing Oracle GoldenGate Secrets in Keystores
  You can use a keystore to store secret keys for tools and external clients such as Oracle GoldenGate.

- Oracle GoldenGate Extract Classic Capture Mode TDE Requirements
  Ensure that you meet the requirements for Oracle GoldenGate Extract to support Transparent Data Encryption capture.
- Configuring Keystore Support for Oracle GoldenGate
  You can configure Transparent Data Encryption keystore support for Oracle GoldenGate by using a shared secret for the keystore.

About Storing Oracle GoldenGate Secrets in Keystores

You can use a keystore to store secret keys for tools and external clients such as Oracle GoldenGate.

The secret key must be a string adhering to Oracle identifier rules. You can add, update, or delete a client secret in an existing keystore. This section describes how to capture Transparent Data Encryption encrypted data in the Oracle GoldenGate Extract (Extract) process using classic capture mode.

TDE support when Extract is in classic capture mode requires the exchange of the following keys:

- TDE support for Oracle GoldenGate in the classic capture mode of the Extract process requires that an Oracle database and the Extract process share the secret to encrypt sensitive information being exchanged. The shared secret is stored securely in the Oracle database and Oracle GoldenGate domains. The shared secret is stored in the software keystore or the HSM as the database secret.
- The decryption key is a password known as the shared secret that is stored securely in the Oracle database and Oracle GoldenGate domains. Only a party that has possession of the shared secret can decrypt the table and redo log keys.

After you configure the shared secret, Oracle GoldenGate Extract uses the shared secret to decrypt the data. Oracle GoldenGate Extract does not handle the TDE master encryption key itself, nor is it aware of the keystore password. The TDE master encryption key and password remain within the Oracle database configuration.

Oracle GoldenGate Extract only writes the decrypted data to the Oracle GoldenGate trail file, which Oracle GoldenGate persists during transit. You can protect this file using your site's operating system standard security protocols, as well as the Oracle GoldenGate AES encryption options. Oracle GoldenGate does not write the encrypted data to a discard file (specified with the `DISCARDFILE` parameter). The word `ENCRYPTED` will be written to any discard file that is in use.

Oracle GoldenGate does require that the keystore be open when processing encrypted data. There is no performance effect of Oracle GoldenGate feature on the TDE operations.

Oracle GoldenGate Extract Classic Capture Mode TDE Requirements

Ensure that you meet the requirements for Oracle GoldenGate Extract to support Transparent Data Encryption capture.

The requirements are as follows:

- To maintain high security standards, ensure that the Oracle GoldenGate Extract process runs as part of the Oracle user (the user that runs the Oracle database). That way, the keys are protected in memory by the same privileges as the Oracle user.

- Run the Oracle GoldenGate Extract process on the same computer as the Oracle database installation.

Configuring Keystore Support for Oracle GoldenGate

You can configure Transparent Data Encryption keystore support for Oracle GoldenGate by using a shared secret for the keystore.

- Step 1: Decide on a Shared Secret for the Keystore
  A shared secret for a keystore is a password.
- Step 2: Configure Oracle Database for TDE Support for Oracle GoldenGate
  The `DBMS_INTERNAL_CLKM` PL/SQL package enables you to configure TDE support for Oracle GoldenGate.
- Step 3: Store the TDE GoldenGate Shared Secret in the Keystore
  The `ADMINISTER KEY MANAGEMENT` statement can store a TDE GoldenGate shared secret in a keystore.
- Step 4: Set the TDE Oracle GoldenGate Shared Secret in the Extract Process
  The GoldenGate Software Command Interface (GGSCI) utility sets the TDE Oracle GoldenGate shared secret in the extract process.

Step 1: Decide on a Shared Secret for the Keystore

A shared secret for a keystore is a password.

- Decide on a shared secret that meets or exceeds Oracle Database password standards.

Do not share this password with any user other than trusted administrators who are responsible for configuring Transparent Data Encryption to work with Oracle GoldenGate Extract.

Related Topics

- *Oracle Database Security Guide*

Step 2: Configure Oracle Database for TDE Support for Oracle GoldenGate

The `DBMS_INTERNAL_CLKM` PL/SQL package enables you to configure TDE support for Oracle GoldenGate.

1. Log in to the database instance as user `SYS` with the `SYSDBA` administrative privilege.

   For example

   ```
   sqlplus sys as sysdba
   Enter password: password
   Connected.
   ```

2. Load the Oracle Database-supplied `DBMS_INTERNAL_CLKM` PL/SQL package.

   For example:

   ```
   @?/app/oracle/product/12.2/rdbms/admin/prvtclkm.plb
   ```

   The `prvtclkm.plb` file also enables Oracle GoldenGate to extract encrypted data from an Oracle database.

3. Grant the EXECUTE privilege on the DBMS_INTERNAL_CLKM PL/SQL package to the Oracle GoldenGate Extract database user.

   For example:

   ```
   GRANT EXECUTE ON DBMS_INTERNAL_CLKM TO psmith;
   ```

   This procedure enables the Oracle database and Oracle GoldenGate Extract to exchange information.

4. Exit SQL*Plus.

Step 3: Store the TDE GoldenGate Shared Secret in the Keystore

The ADMINISTER KEY MANAGEMENT statement can store a TDE GoldenGate shared secret in a keystore.

Before you begin this procedure, ensure that you have configured the TDE software or hardware keystore, based on Configuring a Software Keystore and Configuring a Hardware Keystore.

1. Set the Oracle GoldenGate-TDE key in the keystore by using the following syntax.

   ```
   ADMINISTER KEY MANAGEMENT ADD|UPDATE|DELETE SECRET 'secret'
   FOR CLIENT 'secret_identifier' [USING TAG 'tag']
   IDENTIFIED BY keystore_password [WITH BACKUP [USING 'backup_identifier']];
   ```

   In this specification:

   - *secret* is the client secret key to be stored, updated, or deleted. Enclose this setting in single quotation marks (' ').
   - *secret_identifier* is an alphanumeric string used to identify the secret key. *secret_identifier* does not have a default value. Enclose this setting in single quotation marks (' ').
   - *tag* is an optional, user-defined description for the secret key to be stored. *tag* can be used with the ADD and UPDATE operations. Enclose this setting in single quotation marks (' '). This tag appears in the SECRET_TAG column of the V$CLIENT_SECRETS view. Creating Custom TDE Master Encryption Key Attributes for Reports
   - *keystore_password* is the password for the keystore that is configured.
   - WITH BACKUP is required in case the keystore was not backed up before the ADD, UPDATE or DELETE operation. *backup_identifier* is an optional user-defined description for the backup. Enclose *backup_identifier* in single quotation marks (' ').

   The following example adds a secret key to the keystore and creates a backup in the same directory as the keystore:

   ```
   ADMINISTER KEY MANAGEMENT ADD SECRET 'some_secret'
   FOR CLIENT 'ORACLE_GG' USING TAG 'GoldenGate Secret'
   IDENTIFIED BY password WITH BACKUP USING 'GG backup';
   ```

2. Verify the entry that you just created.

   For example:

   ```
   SELECT CLIENT, SECRET_TAG FROM V$CLIENT_SECRETS WHERE CLIENT = 'ORACLEGG';

   CLIENT   SECRET_TAG
   ```

```
-------- ----------------------------------------
ORACLEGG some_secret
```

3. Switch the log files.

```
CONNECT / AS SYSDBA

ALTER SYSTEM SWITCH LOGFILE;
```

> **See Also:**
>
> - Creating Custom TDE Master Encryption Key Attributes for Reports for more information about tags
> - *Oracle Database Administrator's Guide* for more information about switching log files
> - How Transparent Data Encryption Works with Oracle Real Application Clusters if you are having problems using this procedure in an Oracle Real Application Clusters environment

Step 4: Set the TDE Oracle GoldenGate Shared Secret in the Extract Process

The GoldenGate Software Command Interface (GGSCI) utility sets the TDE Oracle GoldenGate shared secret in the extract process.

1. Start the GGSCI utility.

   For example:

   ```
   ggsci
   ```

2. In the GGSCI utility, run the `ENCRYPT PASSWORD` command to encrypt the shared secret within the Oracle GoldenGate Extract parameter file.

   ```
   ENCRYPT PASSWORD shared_secret algorithm ENCRYPTKEY keyname
   ```

   In this specification:

   - *`shared_secret`* is the clear-text shared secret that you created when you decided on a shared secret for the keystore. This setting is case sensitive.
   - *`algorithm`* is one of the following values to specify AES encryption:
     - `AES128`
     - `AES192`
     - `AES256`
   - *`keyname`* is the logical name of the encryption key in the `ENCKEYS` lookup file. Oracle GoldenGate uses this name to look up the actual key in the `ENCKEYS` file.

   For example:

   ```
   ENCRYPT PASSWORD password AES256 ENCRYPTKEY mykey1
   ```

3. In the Oracle GoldenGate Extract parameter file, set the `DBOPTIONS` parameter with the `DECRYPTPASSWORD` option.

As input, supply the encrypted shared secret and the Oracle GoldenGate-generated or user-defined decryption key.

```
DBOPTIONS DECRYPTPASSWORD shared_secret algorithm ENCRYPTKEY keyname
```

In this specification:

- *shared_secret* is the clear-text shared secret that you created when you decided on a shared secret for the keystore. This setting is case sensitive.
- *algorithm* is one of the following values to specify AES encryption:
  - AES128
  - AES192
  - AES256
- *keyname* is the logical name of the encryption key in the ENCKEYS lookup file.

  For example:

  ```
  DBOPTIONS DECRYPTPASSWORD AACAAAAAAAAAAAIALCKDZIRHOJBHOJUH AES256
  ENCRYPTKEY mykey1
  ```

Related Topics

- Step 1: Decide on a Shared Secret for the Keystore
  A shared secret for a keystore is a password.

5

Managing Keystores and TDE Master Encryption Keys in United Mode

United mode enables you to create a common keystore for the CDB and the PDBs for which the keystore is in united mode.

The keys for the CDB and the PDBs reside in the common keystore.

- About Managing Keystores and TDE Master Encryption Keys in United Mode
  In united mode, you create the keystore and TDE master encryption key for CDB and PDBs that reside in the same keystore.
- Operations That Are Allowed in United Mode
  Many `ADMINISTER KEY MANAGEMENT` operations performed in the CDB root apply to keystores and encryption keys in the united mode PDB.
- Operations That Are Not Allowed in a United Mode PDB
  `ADMINISTER KEY MANAGEMENT` operations that are not allowed in a united mode PDB can be performed in the CDB root.
- Configuring the Keystore Location and Type for United Mode
  For united mode, you can configure the keystore location and type by using only parameters or a combination of parameters and the `ALTER SYSTEM` statement.
- Configuring a Software Keystore for Use in United Mode
  In united mode, the software keystore resides in the CDB root but the master keys from this keystore are available for the PDBs that have their keystore in united mode.
- Configuring a Hardware Keystore in United Mode
  In united mode, a hardware keystore resides in a hardware security module (HSM), which is designed to store encryption keys.
- Administering Keystores and TDE Master Encryption Keys in United Mode
  After you configure a keystore and master encryption key for use in united mode, you can perform tasks such as rekeying TDE master encryption keys.
- Administering Transparent Data Encryption in United Mode
  You can perform general administrative tasks with Transparent Data Encryption in united mode.

About Managing Keystores and TDE Master Encryption Keys in United Mode

In united mode, you create the keystore and TDE master encryption key for CDB and PDBs that reside in the same keystore.

The keys for PDBs having keystore in united mode, can be created from CDB root or from the PDB.

This design enables you to have one keystore to manage the entire CDB environment, enabling the PDBs to share this keystore, but you can customize the behavior of this keystore in the individual united mode PDBs. For example, in a united mode PDB, you can configure a TDE master encryption key for the PDB in the united keystore that you created in the CDB root, open the keystore locally, and close the keystore locally. In order to perform these actions, the keystore in the CDB root must be open.

Before you configure your environment to use united mode or isolated mode, all the PDBs in the CDB environment are considered to be in united mode.

To use united mode, you must follow these general steps:

1. In the CDB root, configure the database to use united mode by setting the `WALLET_ROOT` and `TDE_CONFIGURATION` parameters.

   The `WALLET_ROOT` parameter sets the location for the wallet directory and the `TDE_CONFIGURATION` parameter sets the type of keystore to use.
2. Restart the database so that these settings take effect.
3. In the CDB root, create the keystore, open the keystore, and then create the TDE master encryption key.
4. In each united mode PDB, perform TDE master encryption key tasks as needed, such as opening the keystore locally in the united mode PDB and creating the TDE master encryption key for the PDB. Remember that the keystore is managed by the CDB root, but must contain a TDE master encryption key that is specific to the PDB for the PDB to be able to use TDE.

When you run `ADMINISTER KEY MANAGEMENT` statements in united mode from the CDB root, if the statement accepts the `CONTAINER` clause, and if you set it to `ALL`, then the statement applies only to the CDB root and its associated united mode PDBs. Any PDB that is in isolated mode is not affected.

Operations That Are Allowed in United Mode

Many `ADMINISTER KEY MANAGEMENT` operations performed in the CDB root apply to keystores and encryption keys in the united mode PDB.

Available United Mode-Related Operations in a CDB Root

Table 5-1 describes the `ADMINISTER KEY MANAGEMENT` operations that you can perform in the CDB root.

Table 5-1 ADMINISTER KEY MANAGEMENT United Mode Operations in a CDB Root

Operation	Syntax	United Mode Notes
Creating a keystore	`ADMINISTER KEY MANAGEMENT CREATE KEYSTORE ['keystore_location'] IDENTIFIED BY keystore_password;`	After you create the keystore in the CDB root, by default it is available in the united mode PDBs. Do not include the `CONTAINER` clause.

Table 5-1 (Cont.) ADMINISTER KEY MANAGEMENT United Mode Operations in a CDB Root

Operation	Syntax	United Mode Notes
Opening a keystore	`ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY [EXTERNAL STORE | keystore_password] [CONTAINER = ALL | CURRENT];`	In this operation, the `EXTERNAL STORE` clause uses the password in the SSO wallet located in the `tde_seps` directory under the per-PDB `WALLET_ROOT` location.
Changing a keystore password	`ADMINISTER KEY MANAGEMENT ALTER KEYSTORE PASSWORD IDENTIFIED BY old_keystore_password SET new_keystore_passwordWITH BACKUP [USING 'backup_identifier'];`	Do not include the `CONTAINER` clause.
Backing up a keystore	`ADMINISTER KEY MANAGEMENT BACKUP KEYSTORE [USING 'backup_identifier'] [FORCE KEYSTORE] IDENTIFIED BY [EXTERNAL STORE | keystore_password] [TO 'keystore_location'];`	Do not include the `CONTAINER` clause.
Closing a keystore without force	`ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE [IDENTIFIED BY [EXTERNAL STORE | keystore_password]] [CONTAINER = ALL | CURRENT];`	If an isolated mode PDB keystore is open, then this statement raises an `ORA-46692 cannot close wallet` error.
Closing a keystore with force	`ADMINISTER KEY MANAGEMENT FORCE KEYSTORE CLOSE [IDENTIFIED BY [EXTERNAL STORE | keystore_password]] [CONTAINER = ALL | CURRENT];`	This operation allows the keystore to be closed in the CDB root when an isolated keystore is open.
Creating and activating a new TDE master encryption key (rekeying)	`ADMINISTER KEY MANAGEMENT SET [ENCRYPTION] KEY [FORCE KEYSTORE] [USING TAG 'tag_name'] IDENTIFIED BY [EXTERNAL STORE | keystore_password] [WITH BACKUP [USING 'backup_identifier']] [CONTAINER = ALL | CURRENT]`	-

Table 5-1 (Cont.) ADMINISTER KEY MANAGEMENT United Mode Operations in a CDB Root

Operation	Syntax	United Mode Notes
Creating a user-defined TDE master encryption key for either now (SET) or later on (CREATE)	`ADMINISTER KEY MANAGEMENT [SET | CREATE] [ENCRYPTION] KEY 'mkid:mk | mk' [USING ALGORITHM 'algorithm'] [FORCE KEYSTORE] [USING TAG 'tag_name'] IDENTIFIED BY [EXTERNAL STORE | keystore_password] [WITH BACKUP [USING 'backup_identifier']] [CONTAINER = CURRENT];`	-
Activating an existing TDE master encryption key	`ADMINISTER KEY MANAGEMENT USE [ENCRYPTION] KEY 'key_id [USING TAG 'tag'] IDENTIFIED BY [EXTERNAL STORE | keystore_password] WITH BACKUP [USING 'backup_identifier'];`	Do not include the `CONTAINER` clause.
Tagging a TDE master encryption key	`ADMINISTER KEY MANAGEMENT SET TAG 'tag' FOR 'key_id' IDENTIFIED BY [EXTERNAL STORE | keystore_password ] WITH BACKUP [USING 'backup_identifier'];`	Do not include the `CONTAINER` clause.
Moving a TDE master encryption key to a new keystore	`ADMINISTER KEY MANAGEMENT MOVE [ENCRYPTION] KEYS TO NEW KEYSTORE 'keystore_location1' IDENTIFIED BY keystore1_password FROM [FORCE] KEYSTORE IDENTIFIED BY keystore_password [WITH IDENTIFIER IN { 'key_id' [, 'key_id' ]... | ( subquery ) } ] [WITH BACKUP [USING 'backup_identifier'];`	You can only move the master encryption key to a keystore that is within the same container (for example, between keystores in the CDB root or between keystores in the same PDB). You cannot move the master encryption key from a keystore in the CDB root to a keystore in a PDB, and vice versa. Do not include the `CONTAINER` clause.

Available Operations in a United Mode PDB

Table 5-2 describes the `ADMINISTER KEY MANAGEMENT` operations that you can perform in a united mode PDB.

Table 5-2 ADMINISTER KEY MANAGEMENT United Mode PDB Operations

Operation	Syntax	United Mode Notes
Opening a keystore	`ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY [EXTERNAL STORE | keystore_password] [CONTAINER = CURRENT];`	In this operation, the `EXTERNAL_STORE` clause uses the password in the Secure Sockets Layer (SSL) wallet. This wallet is located in the `tde_seps` directory in the `WALLET_ROOT` location.
Closing a keystore without force	`ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE [IDENTIFIED BY keystore_password] [CONTAINER = CURRENT];`	-
Closing a keystore with force	`ADMINISTER KEY MANAGEMENT FORCE KEYSTORE CLOSE IDENTIFIED BY [EXTERNAL STORE | keystore_password] [CONTAINER = CURRENT];`	-
Creating and activating a new TDE master encryption key (rekeying or rotating)	`ADMINISTER KEY MANAGEMENT SET [ENCRYPTION] KEY [FORCE KEYSTORE] [USING TAG 'tag_name'] IDENTIFIED BY [EXTERNAL STORE | keystore_password] [WITH BACKUP [USING 'backup_identifier']] [CONTAINER = CURRENT];`	-
Creating a user-defined TDE master encryption key for use either now (SET) or later on (CREATE)	`ADMINISTER KEY MANAGEMENT SET | CREATE [ENCRYPTION] KEY 'mkid:mk | mk' [USING ALGORITHM 'algorithm'] [FORCE KEYSTORE] IDENTIFIED BY [EXTERNAL STORE | keystore_password] [WITH BACKUP [USING 'backup_identifier']] [CONTAINER = CURRENT];`	-

Table 5-2 (Cont.) ADMINISTER KEY MANAGEMENT United Mode PDB Operations

Operation	Syntax	United Mode Notes
Activating an existing TDE master encryption key	`ADMINISTER KEY MANAGEMENT USE [ENCRYPTION] KEY 'key_id' [USING TAG 'tag'] IDENTIFIED BY [EXTERNAL STORE	keystore_password] [WITH BACKUP [USING 'backup_identifier']];`	Do not include the `CONTAINER` clause.
Tagging a TDE master encryption key	`ADMINISTER KEY MANAGEMENT SET TAG 'tag' FOR 'key_id' [FORCE KEYSTORE] IDENTIFIED BY [EXTERNAL STORE	keystore_password] [WITH BACKUP [USING 'backup_identifier']];`	Do not include the `CONTAINER` clause.
Moving an encryption key to a new keystore	`ADMINISTER KEY MANAGEMENT MOVE [ENCRYPTION] KEYS TO NEW KEYSTORE 'keystore_location1' IDENTIFIED BY keystore1_password FROM [FORCE] KEYSTORE IDENTIFIED BY keystore_password [WITH IDENTIFIER IN { 'key_id' [, 'key_id']...	(subquery) }] [WITH BACKUP [USING 'backup_identifier']];`	Do not include the `CONTAINER` clause.
Moving a key from a united mode keystore in the CDB root to an isolated mode keystore in a PDB	`ADMINISTER KEY MANAGEMENT ISOLATE KEYSTORE IDENTIFIED BY isolated_keystore_password FROM ROOT KEYSTORE [FORCE KEYSTORE] IDENTIFIED BY [EXTERNAL STORE	united_keystore_password] [WITH BACKUP [USING backup_id]];`	Do not include the `CONTAINER` clause.

Table 5-2 (Cont.) ADMINISTER KEY MANAGEMENT United Mode PDB Operations

Operation	Syntax	United Mode Notes
Using the FORCE clause when a clone of a PDB is using the TDE master encryption key that is being isolated; then coping (rather than moving) the TDE master encryption keys from the keystore that is in the CDB root into the isolated mode keystore of the PDB	`ADMINISTER KEY MANAGEMENT FORCE ISOLATE KEYSTORE IDENTIFIED BY *isolated_keystore_password* FROM ROOT KEYSTORE [FORCE KEYSTORE] IDENTIFIED BY [EXTERNAL STORE | *united_keystore_password*] [WITH BACKUP [USING *backup_id*]];`	-

Operations That Are Not Allowed in a United Mode PDB

`ADMINISTER KEY MANAGEMENT` operations that are not allowed in a united mode PDB can be performed in the CDB root.

These operations are as follows:

- Keystore operations:
  - Performing merge operations on keystores
  - Exporting a keystore
  - Importing a keystore
  - Migrating a keystore
  - Reverse-migrating a keystore
  - Moving the keys of a keystore that is in the CDB root into the keystores of a PDB
  - Moving the keys from a PDB into a united mode keystore that is in the CDB root
- Encryption key operations:
  - Using the `CONTAINER = ALL` clause to create a new TDE master encryption key for later user in each pluggable database (PDB)
- Client secret operations:
  - Adding client secrets
  - Updating client secrets
  - Deleting client secrets

Configuring the Keystore Location and Type for United Mode

For united mode, you can configure the keystore location and type by using only parameters or a combination of parameters and the `ALTER SYSTEM` statement.

- Configuring United Mode by Editing the Initialization Parameter File
  You can configure united mode by setting both the `WALLET_ROOT` and `TDE_CONFIGURATION` parameters in the initialization parameter file.
- Configuring United Mode with the Initialization Parameter File and ALTER SYSTEM
  If your environment relies on server parameter files (spfile) or parameter files (pfile), then you can set `TDE_CONFIGURATION` using `ALTER SYSTEM` with `SCOPE`.

Configuring United Mode by Editing the Initialization Parameter File

You can configure united mode by setting both the `WALLET_ROOT` and `TDE_CONFIGURATION` parameters in the initialization parameter file.

1. Log in to the server where the CDB root of the Oracle database resides.
2. If necessary, create a wallet directory.

   Typically, the wallet directory is located in the `$ORACLE_BASE/admin/db_unique_name` directory, and it is named `wallet`. Preferably, this directory should be empty.
3. Edit the initialization parameter file, which by default is located in the `$ORACLE_HOME/dbs` directory, to include the following parameters:
   - `WALLET_ROOT`, to point to the location of the wallet directory.
     For example, for a database named `orcl`:

     ```
     wallet_root=$ORACLE_BASE/admin/orcl/wallet
     ```
   - `TDE_CONFIGURATION`, to specify one of the following keystore types:
     - `FILE` specifies a software keystore.
     - `OKV` specifies an Oracle Key Vault keystore.
     - `HSM` specifies a hardware security module (HSM) keystore.

     For example, to specify the TDE keystore type:

     ```
     tde_configuration="keystore_configuration=file"
     ```
4. Log in to the CDB root as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.

   For example:

   ```
   sqlplus c##sec_admin as syskm
   Enter password: password
   ```
5. Check the configuration settings.
   - For the `WALLET_ROOT` parameter:

     ```
     SHOW PARAMETER WALLET_ROOT
     ```

The VALUE column of the output should show the absolute path location of the wallet directory.

- For the TDE_CONFIGURATION parameter:

```
SHOW PARAMETER TDE_CONFIGURATION
```

The VALUE column should show the keystore type, prepended with KEYSTORE_CONFIGURATION=.

If the values do not appear, then try restarting your database with the STARTUP command pointing to the location of the initialization parameter file where you added these settings. For example:

```
STARTUP PFILE = /u01/oracle/dbs/init.ora
```

6. Confirm that the TDE_CONFIGURATION parameter was set correctly.

```
SELECT CON_ID, KEYSTORE_MODE FROM V$ENCRYPTION_WALLET;
```

The output should be similar to the following:

```
    CON_ID KEYSTORE
---------- --------
         1 NONE
         2 UNITED
         3 UNITED
         4 UNITED
         5 UNITED
```

The CDB root (CON_ID 1) will always be in the NONE state, and at this stage, the remaining CON_IDs should be set to UNITED. PDBs can be either UNITED or ISOLATED, depending on how you configure them. When you query the V$ENCRYPTION_WALLET view, if the ORA-46691: The value of the KEYSTORE_CONFIGURATION attribute is invalid error appears, then check the initialization parameter file where you added this setting.

After you configure united mode, you can create keystores and master encryption keys, and when these are configured, you can encrypt data.

Configuring United Mode with the Initialization Parameter File and ALTER SYSTEM

If your environment relies on server parameter files (spfile) or parameter files (pfile), then you can set TDE_CONFIGURATION using ALTER SYSTEM with SCOPE.

With this method, you still set the WALLET_ROOT static initialization parameter in the initialization parameter file.

1. Log in to the server where the CDB root of the Oracle database resides.
2. If necessary, create a wallet directory.

   Typically, the wallet directory is located in the $ORACLE_BASE/admin/*db_unique_name* directory, and it is named wallet. Preferably, this directory should be empty.
3. Edit the initialization parameter file to include the WALLET_ROOT static initialization parameter for the wallet directory.

By default, the initialization parameter file is located in the `$ORACLE_HOME/dbs` directory.

For example, for a database instance named `orcl`:

```
wallet_root=$ORACLE_BASE/admin/orcl/wallet
```

4. Log in to the CDB root as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.

   For example:

   ```
   sqlplus c##sec_admin as syskm
   Enter password: password
   ```

5. Restart the database.

   ```
   SHUTDOWN IMMEDIATE
   STARTUP
   ```

   To start the database by pointing to the location of the initialization file where you added the `WALLET_ROOT` setting, issue a `STARTUP` command similar to the following:

   ```
   STARTUP PFILE = /u01/oracle/dbs/init.ora
   ```

6. Check the `WALLET_ROOT` setting.

   ```
   SHOW PARAMETER WALLET_ROOT
   ```

   The `VALUE` column of the output should show the absolute path location of the `wallet` directory.

7. Set the `TDE_CONFIGURATION` dynamic initialization parameter to specify the keystore type, using the following syntax:

   ```
   ALTER SYSTEM SET TDE_CONFIGURATION="KEYSTORE_CONFIGURATION=keystore_type"
   SCOPE=scope_type;
   ```

   In this specification:

   - *`keystore_type`* can be one of the following settings for united mode:
     - `FILE` configures a TDE keystore.
     - `OKV` configures an Oracle Key Vault keystore.
     - `HSM` configures a hardware security module (HSM) keystore.
   - *`scope_type`* sets the type of scope (for example, `both`, `memory`, `spfile`, `pfile`.

   For example, to configure a TDE keystore if the parameter file (`pfile`) is in use, set `scope` to `memory`:

   ```
   ALTER SYSTEM SET TDE_CONFIGURATION="KEYSTORE_CONFIGURATION=FILE"
   SCOPE=memory;
   ```

   To configure a TDE keystore if the server parameter file (`spfile`) is in use, set scope to `both`:

   ```
   ALTER SYSTEM SET TDE_CONFIGURATION="KEYSTORE_CONFIGURATION=FILE"
   SCOPE=both;
   ```

8. Check the `TDE_CONFIGURATION` parameter setting.

   ```
   SHOW PARAMETER TDE_CONFIGURATION
   ```

The VALUE column should show the keystore type, prepended with KEYSTORE_CONFIGURATION=.

9. Confirm that the TDE_CONFIGURATION parameter was set correctly.

```
SELECT CON_ID, KEYSTORE_MODE FROM V$ENCRYPTION_WALLET;
```

The output should be similar to the following:

```
    CON_ID KEYSTORE
---------- --------
         1 NONE
         2 UNITED
         3 UNITED
         4 UNITED
         5 UNITED
```

The CDB root (CON_ID 1) will always be in the NONE state, and at this stage, the remaining CON_IDs should be set to UNITED. PDBs can be either UNITED or ISOLATED, depending on how you configure them. When you query the V$ENCRYPTION_WALLET view, if the ORA-46691: The value of the KEYSTORE_CONFIGURATION attribute is invalid error appears, then check the initialization parameter file where you added this setting.

After you configure united mode, you can create keystores and master encryption keys, and when these are configured, you can encrypt data.

Configuring a Software Keystore for Use in United Mode

In united mode, the software keystore resides in the CDB root but the master keys from this keystore are available for the PDBs that have their keystore in united mode.

- About Configuring a Software Keystore in United Mode
  In united mode, the keystore that you create in the CDB root will be accessible by the united mode PDBs.
- Step 1: Create the Software Keystore
  In united mode, you must create the keystore in the CDB root.
- Opening the Software Keystore in a United Mode PDB
  To open a software keystore in united mode, you must use the ADMINISTER KEY MANAGEMENT statement with the SET KEYSTORE OPEN clause.
- Step 3: Set the TDE Master Encryption Key in the Software Keystore in United Mode
  To set the TDE master encryption key in the keystore when the PDB is configured in united mode, use the ADMINISTER KEY MANAGEMENT statement with the SET KEY clause.
- Step 4: Encrypt Your Data in United Mode
  Now that you have completed the keystore configuration and the PDB is configured in united mode, you can begin to encrypt data in the PDB.

About Configuring a Software Keystore in United Mode

In united mode, the keystore that you create in the CDB root will be accessible by the united mode PDBs.

In general, to configure a united mode software keystore after you have enabled united mode, you create and open the keystore in the CDB root, and then create a master encryption key for this keystore. Afterward, you can begin to encrypt data for tables and tablespaces that will be accessible throughout the CDB environment.

The V$ENCRYPTION_WALLET dynamic view describes the status and location of the keystore. For example, the following query shows the open-closed status and the keystore location of the CDB root keystore (CON_ID 1) and its associated united mode PDBs. The WRL_PARAMETER column shows the CDB root keystore location being in the $ORACLE_BASE/wallet/tde directory.

```
SELECT CON_ID, STATUS, WRL_PARAMETER FROM V$ENCRYPTION_WALLET;

CON_ID STATUS WRL_PARAMETER
------ ------ ----------------------------
     1 OPEN   /app/oracle/wallet/tde/
     2 CLOSED
     3 OPEN
     4 OPEN
     5 OPEN
```

In this output, there is no keystore path listed for the other PDBs in this CDB because these PDBs use the keystore in the CDB root. If any of these PDBs are isolated and you create a keystore in the isolated mode PDB, then when you perform this query, the WRL_PARAMETER column will show the keystore path for the isolated mode PDB.

You can create a secure external store for the software keystore. This feature enables you to hide the password from the operating system: it removes the need for storing clear-text keystore passwords in scripts or other tools that can access the database without user intervention, such as overnight batch scripts. The location for this keystore is set by the EXTERNAL_KEYSTORE_CREDENTIAL_LOCATION initialization parameter. In a multitenant environment, different PDBs can access this external store location when you run the ADMINISTER KEY MANAGEMENT statement using the IDENTIFIED BY EXTERNAL STORE clause. This way, you can centrally locate the password and then update it only once in the external store.

Step 1: Create the Software Keystore

In united mode, you must create the keystore in the CDB root.

After you create this keystore in the CDB root, it becomes available in any united mode PDB, but not in any isolated mode PDBs.

1. Log in to the CDB root as a user who has been granted the ADMINISTER KEY MANAGEMENT or SYSKM privilege.

   For example:

   ```
   sqlplus c##sec_admin as syskm
   Enter password: password
   ```

2. Run the ADMINISTER KEY MANAGEMENT SQL statement to create the keystore using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT CREATE KEYSTORE
   ['keystore_location']
   IDENTIFIED BY software_keystore_password;
   ```

   In this specification:

- *keystore_location* is the path to the keystore directory location of the password-protected keystore for which you want to create the auto-login keystore. If the path that is set by the WALLET_ROOT parameter is the path that you want to use, then you can omit the *keystore_location* setting.

  If you specify the *keystore_location*, then enclose it in single quotation marks (' '). To find the default location, you can query the WRL_PARAMETER column of the V$ENCRYPTION_WALLET view. (If the keystore was not created in the default location, then the STATUS column of the V$ENCRYPTION_WALLET view is NOT_AVAILABLE.)

- *software_keystore_password* is the password of the keystore that you, the security administrator, creates.

For example, to create the keystore in the default location, assuming that WALLET_ROOT has been set:

```
ADMINISTER KEY MANAGEMENT CREATE KEYSTORE
IDENTIFIED BY password;

keystore altered.
```

After you complete these steps, the ewallet.p12 file, which contains the keystore, appears in the designated keystore location. For example, if you had set the WALLET_ROOT parameter to $ORACLE_BASE/wallet and the TDE_CONFIGURATION parameter to FILE (for TDE, which creates a tde directory in the wallet root location), then the keystore will be created in the $ORACLE_BASE/wallet/tde directory. The name of the keystore is ewallet.p12.

Related Topics

- Configuring an External Store for a Keystore Password
  An external store for a keystore password stores the keystore password in a centrally accessed and managed location.

Opening the Software Keystore in a United Mode PDB

To open a software keystore in united mode, you must use the ADMINISTER KEY MANAGEMENT statement with the SET KEYSTORE OPEN clause.

1. Connect to the CDB root as a common user who has been granted the ADMINISTER KEY MANAGEMENT or SYSKM privilege.

   For example:

   ```
   CONNECT c##sec_admin AS SYSKM
   Enter password: password
   ```

2. Open the keystore in the CDB root.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
   IDENTIFIED BY password;

   keystore altered.
   ```

   If the CDB is configured using the EXTERNAL_KEYSTORE_CREDENTIAL_LOCATION instance initialization parameter and has a keystore at that location containing the credentials of the password-protected keystore, and you want to switch over from using an auto-login keystore to using the password-protected keystore with these

credentials, you must include the FORCE KEYSTORE clause and the IDENTIFIED BY EXTERNAL STORE clause in the ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN statement, as follows:

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
FORCE KEYSTORE
IDENTIFIED BY EXTERNAL STORE;

keystore altered.
```

If the WALLET_ROOT parameter has been set, then Oracle Database finds the external store by searching in this path in the CDB root: *WALLET_ROOT*/tde_seps.

3. Ensure that the PDB in which you want to open the keystore is in READ WRITE mode.

   For example:

```
SHOW PDBS

CON_ID CON_NAME   OPEN MODE    RESTRICTED
------ ---------- ------------ -----------
2      PDB$SEED   READ ONLY    NO
3      CDB1_PDB1  READ WRITE   NO
```

   If any PDB has an OPEN MODE value that is different from READ WRITE, then run the following statement to open the PDB, which will set it to READ WRITE mode:

```
ALTER PLUGGABLE DATABASE CDB1_PDB1 OPEN;
```

   Now the keystore can be opened in both the CDB root and the PDB.

4. Connect to the PDB.
5. Run the ADMINISTER KEY MANAGEMENT statement to open the keystore.

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
IDENTIFIED BY password;
keystore altered.
```

   To switch over to opening the password-protected software keystore when an auto-login keystore is configured and is currently open, specify the FORCE KEYSTORE clause as follows.

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
FORCE KEYSTORE
IDENTIFIED BY EXTERNAL STORE;
keystore altered.
```

   FORCE KEYSTORE is also useful for databases that are heavily loaded. The IDENTIFIED BY EXTERNAL STORE clause is included in the statement because the keystore credentials exist in an external store. This enables the password-protected keystore to be opened without specifying the keystore password within the statement itself.

   If the WALLET_ROOT parameter has been set, then Oracle Database finds the external store by searching in this path: *WALLET_ROOT*/*PDB_GUID*/tde_seps.

6. Confirm that the keystore is open.

```
SELECT STATUS FROM V$ENCRYPTION_WALLET;
```

Note that if the keystore is open but you have not created a TDE master encryption key yet, the STATUS column of the V$ENCRYPTION_WALLET view reminds you with an OPEN_NO_MASTER_KEY status.

Step 3: Set the TDE Master Encryption Key in the Software Keystore in United Mode

To set the TDE master encryption key in the keystore when the PDB is configured in united mode, use the ADMINISTER KEY MANAGEMENT statement with the SET KEY clause.

1. Log in to the CDB root or to the PDB that is configured for united mode as a user who has been granted the ADMINISTER KEY MANAGEMENT or SYSKM privilege.
2. Ensure that the database is open in READ WRITE mode.

   To find the status, run the show pdbs command.
3. Run the ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY statement to create or rekey the TDE master encryption key in the keystore.

   For example, if the keystore is password-protected and open, and you want to create or rekey the TDE master encryption key in the current container:

   ```
   ADMINISTER KEY MANAGEMENT SET KEY
   IDENTIFIED BY keystore_password
   WITH BACKUP USING 'emp_key_backup';

   keystore altered.
   ```

   If the keystore is closed:

   ```
   ADMINISTER KEY MANAGEMENT SET KEY
   FORCE KEYSTORE
   IDENTIFIED BY keystore_password
   WITH BACKUP USING 'emp_key_backup';

   keystore altered.
   ```

   In this specification:

   - FORCE KEYSTORE should be included if the keystore is closed. This automatically opens the keystore before setting the TDE master encryption key. The FORCE KEYSTORE clause also switches over to opening the password-protected software keystore when an auto-login keystore is configured and is currently open.
   - IDENTIFIED BY specifies the keystore password. Alternatively, if the keystore password is in an external store, you can use the IDENTIFIED BY EXTERNAL STORE clause.
4. Confirm that the TDE master encryption key is set.

   ```
   SELECT MASTERKEY_ACTIVATED FROM V$DATABASE_KEY_INFO;
   ```

   The output should be YES.

Related Topics

- About Setting the Software Keystore TDE Master Encryption Key
  The TDE master encryption key is stored in the keystore.

Step 4: Encrypt Your Data in United Mode

Now that you have completed the keystore configuration and the PDB is configured in united mode, you can begin to encrypt data in the PDB.

Related Topics

- Encrypting Columns in Tables
  You can use Transparent Data Encryption to encrypt individual columns in database tables.
- Encryption Conversions for Tablespaces and Databases
  You can perform encryption operations on both offline and online tablespaces and databases.

Configuring a Hardware Keystore in United Mode

In united mode, a hardware keystore resides in a hardware security module (HSM), which is designed to store encryption keys.

About Configuring a Hardware Keystore in United Mode

You must create a hardware keystore in a PDB using isolated mode but you can perform other tasks in united mode.

To configure a hardware keystore for a PDB in isolated mode, you first must set the `WALLET_ROOT` parameter. This is necessary for two reasons: first, to have support for migrating to a software keystore in the future, and second, because the configuration file for Oracle Key Vault is retrieved from a location under `WALLET_ROOT`. Afterwards, you must set the `KEYSTORE_CONFIGURATION` attribute of the `TDE_CONFIGURATION` parameter to `HSM` or `OKV`, open the configured hardware keystore, and then set the TDE master encryption key for the PDB. After you complete these tasks, you can begin to encrypt data in the PDB.

How you specify the IDENTIFIED BY clause when you run the ADMINISTER KEY MANAGEMENT statement depends on the type of hardware keystore. For a hardware security module (HSM), you use the following syntax:

```
IDENTIFIED BY "user_name:password"
```

For an Oracle Key Vault keystore, you can omit the *user_name* and colon (but keep the quotation marks):

```
IDENTIFIED BY "password"
```

Step 1: Configure the United Mode Hardware Security Module

To configure a third-party hardware security module, you must copy the PKCS#11 library to the correct location and follow your vendor's instructions

Related Topics

- Step 2: Configure the Hardware Security Module
  To configure a third-party hardware security module, you must copy the PKCS#11 library to the correct location and follow your vendor's instructions.

Step 2: Open the Hardware Keystore in a United Mode PDB

To open a hardware keystore in united mode, you must use the ADMINISTER KEY MANAGEMENT statement with the SET KEYSTORE OPEN clause.

1. Log in to the CDB root as a common user who has been granted the ADMINISTER KEY MANAGEMENT or SYSKM privilege.

   For example:

   ```
   sqlplus c##sec_admin as syskm
   Enter password: password
   ```

2. Open the keystore in the CDB root by using the following syntax.

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
   IDENTIFIED BY "hardware_keystore_credentials";
   ```

   The type of hardware keystore determines how you specify the hardware keystore password. For hardware security modules, you must use the *user_name*:*password* syntax. For example:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
   IDENTIFIED BY "psmith:password";

   keystore altered.
   ```

   For an Oracle Key Vault keystore, you can only provide the password. No user name is allowed in the IDENTIFIED BY clause. Enclose the password in double quotation marks.

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
   IDENTIFIED BY "password";
   ```

3. Ensure that the PDB in which you want to open the keystore is in READ WRITE mode.

   For example:

```
SHOW PDBS

CON_ID CON_NAME   OPEN MODE    RESTRICTED
------ ---------- ------------ ----------
2      PDB$SEED   READ ONLY    NO
3      CDB1_PDB1  READ WRITE   NO
```

If any PDB has an `OPEN MODE` value that is different from `READ WRITE`, then run the following statement to open the PDB, which will set it to `READ WRITE` mode:

```
ALTER PLUGGABLE DATABASE CDB1_PDB1 OPEN;
```

Now the keystore can be opened in both the CDB root and the PDB.

4. Connect to the PDB and run the `ADMINISTER KEY MANAGEMENT` statement to open the keystore.

   For example, for an HSM:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
   IDENTIFIED BY "psmith:password";

   keystore altered.
   ```

   You can include the `FORCE KEYSTORE` clause if there is a chance that the CDB root keystore has been closed, or the database is heavily loaded.

5. Confirm that the keystore is open.

   ```
   SELECT STATUS FROM V$ENCRYPTION_WALLET;
   ```

6. Repeat this procedure each time you restart the database instance.

Related Topics

- About Opening Hardware Keystores
  You must open the hardware keystore so that it is accessible to the database before you can perform any encryption or decryption.

Step 3: Set the TDE Master Encryption Key in the Hardware Keystore in United Mode

To set the TDE master encryption key in the keystore when the PDB is configured in united mode, use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEY` clause.

1. Log in to the CDB root or to the PDB that is configured for united mode as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Ensure that the database is open in `READ WRITE` mode.

   To find the status, run the `SHOW PDBS` command.

3. Run the `ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY` statement to create or rekey the TDE master encryption key in the keystore.

   ```
   ADMINISTER KEY MANAGEMENT SET KEY
   [USING TAG 'tag']
   [FORCE KEYSTORE]
   IDENTIFIED BY [EXTERNAL STORE | "hardware_keystore_credentials"];
   ```

   In this specification:

- `FORCE KEYSTORE` temporarily opens the password-protected keystore for this operation. You must open the keystore for this operation.
- *`hardware_keystore_credentials`* refers to the credentials for either an HSM or an Oracle Key Vault hardware keystore. For an HSM, specify the credentials using this format, enclosed in quotation marks and separating the components with a colon: *"user_name:password"*, with *`user_name`* being the user who created the HSM and password being this user's password. For Oracle Key Vault, enter only the password of the user who created the keystore. Enclose this password with quotation marks.

For example:

```
ADMINISTER KEY MANAGEMENT SET KEY
FORCE KEYSTORE
IDENTIFIED BY "psmith:password";

keystore altered.
```

4. Confirm that the TDE master encryption key is set.

```
SELECT MASTERKEY_ACTIVATED FROM V$DATABASE_KEY_INFO;
```

The output should be `YES`.

Related Topics

- About Setting the Hardware Keystore TDE Master Encryption Key
  You must create a TDE master encryption key that is stored inside the hardware keystore.

Step 4: Encrypt Your Data in United Mode

Now that you have completed the keystore configuration and the PDB is configured in united mode, you can begin to encrypt data in the PDB.

Related Topics

- Encrypting Columns in Tables
  You can use Transparent Data Encryption to encrypt individual columns in database tables.
- Encryption Conversions for Tablespaces and Databases
  You can perform encryption operations on both offline and online tablespaces and databases.

Administering Keystores and TDE Master Encryption Keys in United Mode

After you configure a keystore and master encryption key for use in united mode, you can perform tasks such as rekeying TDE master encryption keys.

- Changing the Keystore Password in United Mode
  You can change the password of either a software keystore or a hardware keystores only in the CDB root.

- Backing Up a Password-Protected Software Keystore in United Mode
  The `BACKUP KEYSTORE` clause of the `ADMINISTER KEY MANAGEMENT` statement backs up a password-protected software keystore.
- Closing Keystores in United Mode
  You can close both software and hardware keystores in united mode, unless the system tablespace is encrypted.
- Creating a User-Defined TDE Master Encryption Key in United Mode
  To create a user-defined TDE master encryption key, use the `ADMINISTER KEY MANAGEMENT` statement with the `SET | CREATE [ENCRYPTION] KEY` clause.
- Example: Creating a Master Encryption Key in All PDBs
  You can use the `ADMINISTER KEY MANAGEMENT CREATE KEY USING TAG` statement to create a TDE master encryption key in all PDBs.
- Creating a TDE Master Encryption Key for Later Use in United Mode
  A keystore must be opened before you can create a TDE master encryption key for use later on in united mode.
- Activating a TDE Master Encryption Key in United Mode
  To activate a TDE master encryption key in united mode, you must open the keystore and use `ADMINISTER KEY MANAGEMENT` with the `USE KEY` clause.
- Rekeying the TDE Master Encryption Key in United Mode
  You can use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEY` clause to rekey a TDE master encryption key.
- Finding the TDE Master Encryption Key That Is in Use in United Mode
  A TDE master encryption key that is in use is the key that was activated most recently for the database.
- Creating a Custom Attribute Tag in United Mode
  To create a custom attribute tag in united mode, you must use the `SET TAG` clause of the `ADMINISTER KEY MANAGEMENT` statement.
- Moving a TDE Master Encryption Key into a New Keystore in United Mode
  In united mode, you can move an existing TDE master encryption key into a new keystore from an existing software password keystore.
- Automatically Removing Inactive TDE Master Encryption Keys in United Mode
  In united mode, the `REMOVE_INACTIVE_STANDBY_TDE_MASTER_KEY` initialization parameter can configure the automatic removal of inactive TDE master encryption keys.
- Isolating a Pluggable Database Keystore
  Isolating a PDB keystore moves the master encryption key from the CDB root keystore into an isolated mode keystore in the a PDB.

Changing the Keystore Password in United Mode

You can change the password of either a software keystore or a hardware keystores only in the CDB root.

- Changing the Password-Protected Software Keystore Password in United Mode
  To change the password of a password-protected software keystore in united mode, you must use the `ADMINISTER KEY MANAGEMENT` statement in the CDB root.

- Changing the Password of a Hardware Keystore in United Mode
  To change the password of a hardware keystore, you must close the hardware keystore and then change the password from the hardware keystore management interface.

Changing the Password-Protected Software Keystore Password in United Mode

To change the password of a password-protected software keystore in united mode, you must use the `ADMINISTER KEY MANAGEMENT` statement in the CDB root.

You cannot change keystore passwords from a united mode PDB.

1. Log in to the CDB root as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Use the following syntax to change the password for the keystore:

```
ADMINISTER KEY MANAGEMENT ALTER KEYSTORE PASSWORD
[FORCE KEYSTORE]
IDENTIFIED BY
old_keystore_password SET new_keystore_password
[WITH BACKUP [USING 'backup_identifier']];
```

   In this specification:

   - `FORCE KEYSTORE` temporarily opens the password-protected keystore for this operation if the keystore is closed if an auto-login keystore is configured and is currently open, or if a password-protected keystore is configured and is currently closed.
   - *`old_password`* is the current keystore password that you want to change.
   - *`new_password`* is the new password that you set for the keystore.
   - You do not need to include the `CONTAINER` clause because the password can only be changed locally, in the CDB root.

   The following example creates a backup of the keystore and then changes the password:

```
ADMINISTER KEY MANAGEMENT ALTER KEYSTORE PASSWORD
IDENTIFIED BY
old_password SET new_password
WITH BACKUP USING 'pwd_change';

keystore altered.
```

   This example performs the same operation but uses the `FORCE KEYSTORE` clause in case the auto-login software keystore is in use or the password-protected software keystore is closed.

```
ADMINISTER KEY MANAGEMENT ALTER KEYSTORE PASSWORD
FORCE KEYSTORE
IDENTIFIED BY
old_password SET new_password
WITH BACKUP USING 'pwd_change';

keystore altered.
```

Related Topics

- Performing Operations That Require a Keystore Password
  Many `ADMINISTER KEY MANAGEMENT` operations require access to a keystore password, for both software and hardware keystores.
- Changing the Password of a Software Keystore
  Oracle Database enables you to easily change password-protected software keystore passwords.

Changing the Password of a Hardware Keystore in United Mode

To change the password of a hardware keystore, you must close the hardware keystore and then change the password from the hardware keystore management interface.

You cannot change keystore passwords from a united mode PDB.

1. Log in to the CDB root as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Close the hardware keystore.

   For example, for an HSM:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE
   IDENTIFIED BY "psmith:password";
   ```

   For a keystore whose password is stored externally:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE
   IDENTIFIED BY EXTERNAL STORE;
   ```

3. From the hardware security module management interface, create a new hardware security module password.
4. Update the credentials of the HSM in the external store to use "*user_name*:*password*".

   Currently, the external store contains the old HSM credentials, which would no longer work.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT
   UPDATE SECRET 'user_name:password'
   FOR CLIENT 'TDE_WALLET'
   TO LOCAL AUTO_LOGIN KEYSTORE '/etc/ORACLE/WALLETS/orcl/external_store';
   ```

5. In SQL*Plus, open the hardware keystore.

   For example, for an HSM:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
   IDENTIFIED BY "psmith:new_password";
   ```

   For a hardware keystore whose password is stored externally:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
   IDENTIFIED BY EXTERNAL STORE;
   ```

Related Topics

- Performing Operations That Require a Keystore Password
  Many `ADMINISTER KEY MANAGEMENT` operations require access to a keystore password, for both software and hardware keystores.

Backing Up a Password-Protected Software Keystore in United Mode

The `BACKUP KEYSTORE` clause of the `ADMINISTER KEY MANAGEMENT` statement backs up a password-protected software keystore.

1. Log in to the CDB root as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.

   For example:

   ```
   sqlplus c##sec_admin@hrpdb as syskm
   Enter password: password
   Connected.
   ```

2. Back up the keystore by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT BACKUP KEYSTORE
   [USING 'backup_identifier']
   [FORCE KEYSTORE]
   IDENTIFIED BY [EXTERNAL STORE | software_keystore_password]
   [TO 'keystore_location'];
   ```

   In this specification:

   - `USING` *`backup_identifier`* is an optional string that you can provide to identify the backup. Enclose this identifier in single quotation marks (`' '`). This identifier is appended to the named keystore file (for example, `ewallet_`*`time-stamp`*`_emp_key_backup.p12`).
   - `FORCE KEYSTORE` temporarily opens the password-protected keystore for this operation. You must open the keystore for this operation.
   - `IDENTIFIED BY` is required for the `BACKUP KEYSTORE` operation on a password-protected keystore because although the backup is simply a copy of the existing keystore, the status of the TDE master encryption key in the password-protected keystore must be set to `BACKED UP` and for this change the keystore password is required.
   - *`keystore_location`* is the path at which the backup keystore is stored. This setting is restricted to the PDB when the PDB lockdown profile `EXTERNAL_FILE_ACCESS` setting is blocked in the PDB or when the `PATH_PREFIX` variable was not set when the PDB was created. If you do not specify the *`keystore_location`*, then the backup is created in the same directory as the original keystore. Enclose this location in single quotation marks (`' '`).
   - You do not need to include the `CONTAINER` clause because the keystore can only be backup up locally, in the CDB root.

   The following example backs up a software keystore in the same location as the source keystore.

   ```
   ADMINISTER KEY MANAGEMENT BACKUP KEYSTORE
   USING 'hr.emp_keystore'
   FORCE KEYSTORE
   IDENTIFIED BY
   ```

```
software_keystore_password ;

keystore altered.
```

In the following version, the password for the keystore is external, so the `EXTERNAL STORE` clause is used.

```
ADMINISTER KEY MANAGEMENT BACKUP KEYSTORE
USING 'hr.emp_keystore'
FORCE KEYSTORE
IDENTIFIED BY EXTERNAL STORE;
```

After you run this statement, an `ewallet_identifier.p12` file (for example, `ewallet_time-stamp_hr.emp_keystore.p12`) appears in the keystore backup location.

Related Topics

- Backing Up Password-Protected Software Keystores
  When you back up a password-protected software keystore, you can create a backup identifier string to describe the backup type.

Closing Keystores in United Mode

You can close both software and hardware keystores in united mode, unless the system tablespace is encrypted.

- Closing a Software Keystore in United Mode
  You can close password-protected keystores, auto-login keystores, and local auto-login software keystores in united mode.
- Closing a Hardware Keystore in United Mode
  To close a hardware keystore, you must use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEYSTORE CLOSE` clause.

Closing a Software Keystore in United Mode

You can close password-protected keystores, auto-login keystores, and local auto-login software keystores in united mode.

In the case of an auto-login keystore, which opens automatically when it is accessed, you must first move it to a new location where it cannot be automatically opened, then you must manually close it. You must do this if you are changing your configuration from an auto-login keystore to a password-protected keystore: you change the configuration to stop using the auto-login keystore (by moving the auto-login keystore to another location where it cannot be automatically opened), and then closing the auto-login keystore.

1. Log in to the CDB root or the united mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Close the software keystore.
   - For a password-protected software keystore, use the following syntax if you are in the CDB root:

     ```
     ADMINISTER KEY MANAGEMENT SET | FORCE KEYSTORE CLOSE
     [IDENTIFIED BY [EXTERNAL STORE | software_keystore_password]]
     [CONTAINER = ALL | CURRENT];
     ```

Use the SET clause to close the keystore without force. If there is a dependent keystore that is open (for example, an isolated mode PDB keystore and you are trying to close the CDB root keystore), then an ORA-46692 cannot close wallet error appears. If this happens, then use the FORCE clause instead of SET to temporarily close the dependent keystore during the close operation. The STATUS column of the V$ENCRYPTION_WALLET view shows if a keystore is open.

If you are in the united mode PDB, then either omit the CONTAINER clause or set it to CURRENT.

- For an auto-login or local auto-login software keystore, use this syntax if you are in the CDB root:

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE
[CONTAINER =  ALL | CURRENT];
```

Closing a keystore disables all of the encryption and decryption operations. Any attempt to encrypt or decrypt data or access encrypted data results in an error.

Related Topics

- About Closing Keystores
  After you open a keystore, it remains open until you shut down the database instance.

Closing a Hardware Keystore in United Mode

To close a hardware keystore, you must use the ADMINISTER KEY MANAGEMENT statement with the SET KEYSTORE CLOSE clause.

1. Log in to the CDB root or the united mode PDB as a user who has been granted the ADMINISTER KEY MANAGEMENT or SYSKM privilege.
2. Close the hardware keystore by using the following syntax:

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE
IDENTIFIED BY [EXTERNAL STORE | "hardware_keystore_credentials"]
[CONTAINER = ALL | CURRENT];
```

For example, for an HSM:

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE
IDENTIFIED BY "psmith:password"
CONTAINER = ALL;
```

If an ORA-46692 cannot close wallet error appears, then check if any isolated mode keystores are open. To find the status of a keystore, query the STATUS column of the V$ENCRYPTION_WALLET view.

Closing a keystore disables all of the encryption and decryption operations. Any attempt to encrypt or decrypt data or access encrypted data results in an error.

Related Topics

- About Closing Keystores
  After you open a keystore, it remains open until you shut down the database instance.

Creating a User-Defined TDE Master Encryption Key in United Mode

To create a user-defined TDE master encryption key, use the `ADMINISTER KEY MANAGEMENT` statement with the `SET | CREATE [ENCRYPTION] KEY` clause.

1. Log in to the CDB root a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.

   For example:

   ```
   sqlplus c##sec_admin as syskm
   Enter password: password
   ```

2. Create the user-defined TDE master encryption key by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT SET | CREATE [ENCRYPTION] KEY
   'mkid:mk | mk'
   [USING ALGORITHM 'algorithm']
   [FORCE KEYSTORE]
   [USING TAG 'tag_name']
   IDENTIFIED BY [EXTERNAL STORE | keystore_password]
   [WITH BACKUP [USING 'backup_identifier']]
   [CONTAINER = CURRENT];
   ```

   In this specification:

   - `SET | CREATE` : Enter `SET` if you want to create the master and activate the TDE master encryption key now, or enter `CREATE` if you want to create the key for later use, without activating it yet.
   - *`mkid`* and *`mk`*:
     - *`mkid`*, the TDE master encryption key ID, is a 16–byte hex-encoded value that you can specify or have Oracle Database generate.
     - *`mk`*, the TDE master encryption key, is a hex-encoded value that you can specify or have Oracle Database generate, either 32 bytes (for the for `AES256`, `ARIA256`, and `GOST256` algorithms) or 16 bytes (for the `SEED128` algorithm).

     If you omit the *`mkid`* value but include the *`mk`*, then Oracle Database generates the *`mkid`* for the *`mk`*.

     If you omit the entire *`mkid`*`:`*`mk`*`|`*`mkid`* clause, then Oracle Database generates these values for you.
   - `USING ALGORITHM`: Specify one of the following supported algorithms:
     - `AES256`
     - `ARIA256`
     - `SEED128`
     - `GOST256`

     If you omit the algorithm, then the default, `AES256`, is used.
   - `FORCE KEYSTORE` temporarily opens the password-protected keystore for this operation. You must open the keystore for this operation.
   - `CONTAINER`: If you include this clause, then set it to `CURRENT`. (`CURRENT` is the default.)

The following example includes a user-created TDE master encryption key but no TDE master encryption key ID, so that the TDE master encryption key is generated:

```
ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY
'3D432109DF88967A541967062A6F4E460E892318E307F017BA048707B402493C'
USING ALGORITHM 'GOST256'
FORCE KEYSTORE
IDENTIFIED BY keystore_password;
```

The next example creates user-defined keys for both the master encryption ID and the TDE master encryption key. It omits the algorithm specification, so the default algorithm AES256 is used.

```
ADMINISTER KEY MANAGEMENT CREATE ENCRYPTION KEY
'10203040506070801112131415161718:3D432109DF88967A541967062A6F4E460E892318E30
7F017BA048707B402493C'
IDENTIFIED BY keystore_password
CONTAINER = CURRENT;
```

Related Topics

- Supported Encryption and Integrity Algorithms
  The supported Advanced Encryption Standard cipher keys, including tablespace and database encryption keys, can be either 128, 192, or 256 bits long. Tablespace and database encryption use the 128–bit length cipher key.

Example: Creating a Master Encryption Key in All PDBs

You can use the ADMINISTER KEY MANAGEMENT CREATE KEY USING TAG statement to create a TDE master encryption key in all PDBs.

Example 5-1 shows how to create a master encryption key in all of the PDBs in a multitenant environment. It uses the FORCE KEYSTORE clause in the event that the auto-login keystore in the CDB root is open. The password is stored externally, so the EXTERNAL STORE setting is used for the IDENTIFIED BY clause. After you execute this statement, a master encryption key is created in each PDB. You can find the identifiers for these keys as follows:

- Log in to the PDB and then query the TAG column of the V$ENCRYPTION_KEYS view.
- Log in to the CDB root and then query the INST_ID and TAG columns of the GV$ENCRYPTION_KEYS view.

You also can check the CREATION_TIME column of these views to find the most recently created key, which would be the key that you created from this statement. After you create the keys, you can individually activate the keys in each of the PDBs.

Example 5-1 Creating a Master Encryption Key in All of the PDBs

```
ADMINISTER KEY MANAGEMENT CREATE KEY USING TAG
'scope:all pdbs;description:Create Key for ALL PDBS'
FORCE KEYSTORE IDENTIFIED BY EXTERNAL STORE
WITH BACKUP
CONTAINER = ALL;

keystore altered.
```

Creating a TDE Master Encryption Key for Later Use in United Mode

A keystore must be opened before you can create a TDE master encryption key for use later on in united mode.

1. Log in to the CDB root or the united mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Create the TDE master encryption key by using the following syntax:

```
ADMINISTER KEY MANAGEMENT CREATE [ENCRYPTION] KEY
[FORCE KEYSTORE]
[USING TAG 'tag_name']
IDENTIFIED BY [EXTERNAL STORE | keystore_password]
[WITH BACKUP [USING 'backup_identifier']]
[CONTAINER = ALL | CURRENT];
```

   In this specification:

   - `FORCE KEYSTORE` temporarily opens the password-protected keystore for this operation. You must open the keystore for this operation.
   - `CONTAINER`: In the CDB root, set `CONTAINER` to either `ALL` or `CURRENT`. In a PDB, set it to `CURRENT`. In both cases, omitting `CONTAINER` defaults to `CURRENT`.

   For example:

```
ADMINISTER KEY MANAGEMENT CREATE KEY
FORCE KEYSTORE
IDENTIFIED BY keystore_password
WITH BACKUP
CONTAINER = CURRENT;
```

3. If necessary, activate the TDE master encryption key.
   a. Find the key ID.

```
SELECT KEY_ID FROM V$ENCRYPTION_KEYS;

KEY_ID
------------------------------------------------------
AWsHwVYC2U+Nv3RVphn/yAIAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

   b. Use this key ID to activate the key.

```
ADMINISTER KEY MANAGEMENT USE KEY
'AWsHwVYC2U+Nv3RVphn/yAIAAAAAAAAAAAAAAAAAAAAAAAAAAAA'
USING TAG 'quarter:second;description:Activate Key on standby'
IDENTIFIED BY password
WITH BACKUP;
```

Related Topics

- Creating TDE Master Encryption Keys for Later Use
  You can create a TDE master encryption key that can be activated at a later date.

Activating a TDE Master Encryption Key in United Mode

To activate a TDE master encryption key in united mode, you must open the keystore and use `ADMINISTER KEY MANAGEMENT` with the `USE KEY` clause.

1. Log in to the CDB root or the united mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Query the `ORIGIN` and `KEY_ID` columns of the `V$ENCRYPTION_KEYS` view to find the key identifier.

   For example:

   ```
   SELECT ORIGIN, KEY_ID FROM V$ENCRYPTION_KEYS;

   ORIGIN  KEY_ID
   ------  ----------------------------------------
   LOCAL   ARaHD762tUkkvyLgPzAi6hMAAAAAAAAAAAAAAAAAAAAAAAAAAAA
   ```

3. Use this key identifier to activate the TDE master encryption key by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT USE KEY 'key_identifier_from_V$ENCRYPTION_KEYS'
   [USING TAG 'tag']
   [FORCE KEYSTORE]
   IDENTIFIED BY [EXTERNAL STORE | keystore_password]
   WITH BACKUP [USING 'backup_identifier']
   [CONTAINER = ALL | CURRENT];
   ```

   In this specification:

   - `FORCE KEYSTORE` temporarily opens the password-protected keystore for this operation. You must open the keystore for this operation.
   - `CONTAINER`: In the CDB root, set `CONTAINER` to either `ALL` or `CURRENT`. In a PDB, omit the `CONTAINER` clause. In both cases, omitting `CONTAINER` defaults to `CURRENT`.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT USE KEY
   'ARaHD762tUkkvyLgPzAi6hMAAAAAAAAAAAAAAAAAAAAAAAAAAAA'
   FORCE KEYSTORE
   IDENTIFIED BY EXTERNAL STORE
   WITH BACKUP
   CONTAINER = ALL;
   ```

Related Topics

- About Activating TDE Master Encryption Keys
  You can activate a previously created or imported TDE master encryption key by using the `USE KEY` clause of `ADMINSTER KEY MANAGEMENT`.

Rekeying the TDE Master Encryption Key in United Mode

You can use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEY` clause to rekey a TDE master encryption key.

1. Log in to the CDB root or the united mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.

2. If you are rekeying the TDE master encryption key for a keystore that has auto login enabled, then ensure that both the auto login keystore, identified by the `.sso` file, and the encryption keystore, identified by the `.p12` file, are present.

   You can find the location of these files by querying the `WRL_PARAMETER` column of the `V$ENCRYPTION_WALLET` view. To find the `WRL_PARAMETER` values for all of the database instances, query the `GV$ENCRYPTION_WALLET` view.

3. Rekey the TDE master encryption key by using the following syntax:

```
ADMINISTER KEY MANAGEMENT SET [ENCRYPTION] KEY
[FORCE KEYSTORE]
[USING TAG 'tag_name']
IDENTIFIED BY [EXTERNAL STORE | keystore_password]
[WITH BACKUP [USING 'backup_identifier']]
[CONTAINER = ALL | CURRENT];
```

   In this specification:

   - *tag* is the associated attributes and information that you define. Enclose this setting in single quotation marks (`' '`).
   - `FORCE KEYSTORE` temporarily opens the password-protected keystore for this operation. You must open the keystore for this operation.
   - `IDENTIFIED BY` can be one of the following settings:
     - `EXTERNAL STORE` uses the keystore password stored in the external store to perform the keystore operation.
     - *keystore_password* is the password that was created for this keystore.
   - `CONTAINER`: In the CDB root, set `CONTAINER` to either `ALL` or `CURRENT`. In a PDB, set it to `CURRENT`. In both cases, omitting `CONTAINER` defaults to `CURRENT`.

   For example:

```
ADMINISTER KEY MANAGEMENT SET KEY
FORCE KEYSTORE
IDENTIFIED BY keystore_password
WITH BACKUP USING 'emp_key_backup'
CONTAINER = CURRENT;

keystore altered.
```

Related Topics

- About Rekeying the TDE Master Encryption Key
  Oracle Database uses a unified TDE Master Encryption Key for both TDE column encryption and TDE tablespace encryption.

Finding the TDE Master Encryption Key That Is in Use in United Mode

A TDE master encryption key that is in use is the key that was activated most recently for the database.

In united mode, the TDE master encryption key in use of the PDB is the one that was activated most recently for that PDB.

- To find the TDE master encryption key that is in use, query the `V$ENCRYPTION_KEYS` dynamic view.

  For example:

```
SELECT KEY_ID
FROM V$ENCRYPTION_KEYS
WHERE ACTIVATION_TIME = (SELECT MAX(ACTIVATION_TIME)
       FROM V$ENCRYPTION_KEYS
       WHERE ACTIVATING_PDBID = SYS_CONTEXT('USERENV', 'CON_ID'));
```

Creating a Custom Attribute Tag in United Mode

To create a custom attribute tag in united mode, you must use the `SET TAG` clause of the `ADMINISTER KEY MANAGEMENT` statement.

1. Log in to the CDB root or the united mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. If necessary, query the `TAG` column of the `V$ENCRYPTION_KEY` dynamic view to find a listing of existing tags for the TDE master encryption keys.

   When you create a new tag for a TDE master encryption key, it overwrites the existing tag for that TDE master encryption key.
3. Create the custom attribute tag by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT SET TAG 'tag'
   FOR 'master_key_identifier'
   [FORCE KEYSTORE]
   IDENTIFIED BY [EXTERNAL STORE | keystore_password]
   WITH BACKUP [USING 'backup_identifier'];
   ```

   In this specification

   - *`tag`* is the associated attributes or information that you define. Enclose this information in single quotation marks (`' '`).
   - *`master_key_identifier`* identifies the TDE master encryption key for which the *`tag`* is set. To find a list of TDE master encryption key identifiers, query the `KEY_ID` column of the `V$ENCRYPTION_KEYS` dynamic view.
   - `FORCE KEYSTORE` temporarily opens the password-protected keystore for this operation. You must open the keystore for this operation.
   - `IDENTIFIED BY` can be one of the following settings:
     - `EXTERNAL STORE` uses the keystore password stored in the external store to perform the keystore operation.
     - *`keystore_password`* is the password that was created for this keystore.
   - *`backup_identifier`* defines the tag values. Enclose this setting in single quotation marks (`' '`) and separate each value with a colon.

   For example, to create a tag that uses two values, one to capture a specific session ID and the second to capture a specific terminal ID:

   ```
   ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY
   USING TAG 'sessionid=3205062574:terminal=xcvt'
   IDENTIFIED BY keystore_password
   WITH BACKUP;

   keystore altered.
   ```

Both the session ID (`3205062574`) and terminal ID (`xcvt`) can derive their values by using either the `SYS_CONTEXT` function with the `USERENV` namespace, or by using the `USERENV` function.

Related Topics

- About Creating Custom Attribute Tags
  Attribute tags enable you to monitor specific activities users perform, such as accessing a particular terminal ID.

Moving a TDE Master Encryption Key into a New Keystore in United Mode

In united mode, you can move an existing TDE master encryption key into a new keystore from an existing software password keystore.

This feature enables you to delete unused keys. After you move the key to a new keystore, you then can delete the old keystore.

1. Log in to the CDB root or the united mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Query the `KEY_ID` column of the `V$ENCRYPTION_KEYS` view to find the key identifier of the keystore to which you want to move the keys.

   For example:

   ```
   SELECT CREATION_TIME, KEY_ID FROM V$ENCRYPTION_KEYS;

   CREATION TIME
   ---------------------------------------------------
   22-SEP-17 08.55.12.956170 PM +00:00

   KEY_ID
   ---------------------------------------------------
   ARaHD762tUkkvyLgPzAi6hMAAAAAAAAAAAAAAAAAAAAAAAAAAAA
   ```

3. Move the key into a new keystore by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT
   MOVE [ENCRYPTION] KEYS
   TO NEW KEYSTORE 'keystore_location1'
   IDENTIFIED BY keystore1_password
   FROM [FORCE] KEYSTORE
   IDENTIFIED BY keystore_password
   [WITH IDENTIFIER IN
   { 'key_identifier' [, 'key_identifier' ]... | ( subquery ) } ]
   [WITH BACKUP [USING 'backup_identifier']];
   ```

   In this specification:

   - *keystore_location1* is the path to the `wallet` directory that will store the new keystore `.p12` file. By default, this directory is in `$ORACLE_BASE/admin/`*db_unique_name*`/wallet`.
   - `FORCE` temporarily opens the keystore for this operation.
   - *keystore_password* is the password for the keystore from which the key is moving.

   For example:

```
ADMINISTER KEY MANAGEMENT MOVE KEYS
TO NEW KEYSTORE '$ORACLE_BASE/admin/orcl/wallet'
IDENTIFIED BY keystore_password
FROM FORCE KEYSTORE
IDENTIFIED BY keystore_password
WITH IDENTIFIER IN
(SELECT KEY_ID FROM V$ENCRYPTION_KEYS WHERE ROWNUM < 2)
WITH BACKUP;
```

4. To delete the old keystore, go to the `wallet` directory and then manually delete the `.p12` file containing the keystore.

   To find the location of the keystore, open the keystores, and then query the `WRL_PARAMETER` column of the `V$ENCRYPTION_WALLET` view.

Related Topics

- Dangers of Deleting Keystores
  Oracle strongly recommends that you do not delete keystores.

Automatically Removing Inactive TDE Master Encryption Keys in United Mode

In united mode, the `REMOVE_INACTIVE_STANDBY_TDE_MASTER_KEY` initialization parameter can configure the automatic removal of inactive TDE master encryption keys.

1. Log in to the server where the CDB root or the united mode PDB of the Oracle standby database resides.
2. Locate the initialization parameter file for the database.

   By default, the initialization parameter fileis located in the `$ORACLE_HOME/dbs` directory.
3. Edit the initialization parameter file to include the `REMOVE_INACTIVE_STANDBY_TDE_MASTER_KEY` initialization parameter.

   For example:

   ```
   remove_inactive_standby_tde_master_key = true
   ```

   Setting this parameter to `TRUE` enables the automatic removal of inactive TDE master encryption keys; setting it to `FALSE` disables the automatic removal.

Isolating a Pluggable Database Keystore

Isolating a PDB keystore moves the master encryption key from the CDB root keystore into an isolated mode keystore in the a PDB.

This process enables the keystore to be managed as a separate keystore in isolated mode. This way, an administrator who has been locally granted the `ADMINISTER KEY MANAGEMENT` privilege for the PDB can manage the keystore.

1. Log in to the united mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.

   For example:

```
sqlplus sec_admin@hrpdb as syskm
Enter password: password
Connected.
```

2. Move the keys from the keystore of the CDB root into the isolated mode keystore of the PDB by using the following syntax:

```
ADMINISTER KEY MANAGEMENT [FORCE] ISOLATE KEYSTORE
IDENTIFIED BY isolated_keystore_password
FROM ROOT KEYSTORE
[FORCE KEYSTORE]
IDENTIFIED BY
[EXTERNAL STORE | united_keystore_password]
[WITH BACKUP [USING backup_id]];
```

   In this specification:

   - `FORCE` is used when a clone of the PDB is using the master encryption key that is being isolated. The `ADMINISTER KEY MANAGEMENT` statement then copies (rather than moves) the keys from the wallet of the CDB root into the isolated mode PDB.
   - `FORCE KEYSTORE` temporarily opens the password-protected keystore for this operation if an auto-login keystore is open (and in use) or if the keystore is closed.
   - *`united_keystore_password`*: Knowledge of this password does not enable the user who performs the `ISOLATE KEYSTORE` operation privileges to perform `ADMINISTER KEY MANAGEMENT UNITE KEYSTORE` operations on the CDB root. This password is the same as the keystore password in the CDB root.

   After the keystore of a CDB root has been united with that of a PDB, all of the previously active (historical) master encryption keys that were associated with the CDB are moved to the keystore of the PDB.

3. Confirm that the united mode PDB is now an isolated mode PDB.

```
SELECT KEYSTORE_MODE FROM V$ENCRYPTION_WALLET;
```

   The output should be `ISOLATED`.

After the united mode PDB has been converted to an isolated mode PDB, you can change the password of the keystore.

Administering Transparent Data Encryption in United Mode

You can perform general administrative tasks with Transparent Data Encryption in united mode.

- Moving PDBs from One CDB to Another in United Mode
  In united mode, you can automatically move a PDB from one CDB to another (for example, for load balancing or adding new functionality).
- Unplugging and Plugging a PDB with Encrypted Data in a CDB in United Mode
  In united mode, for a PDB that has encrypted data, you can plug it into a CDB. Conversely, you can unplug this PDB from the CDB.
- Managing Cloned PDBs with Encrypted Data in United Mode
  In united mode, you can clone a PDB that has encrypted data in a CDB.

- How Keystore Open and Close Operations Work in United Mode
  You should be aware of how keystore open and close operations work in united mode.
- Finding the Keystore Status for All of the PDBs in United Mode
  You can create a convenience function that uses the `V$ENCRYPTION_WALLET` view to find the status for keystores in all PDBs in a CDB.

Moving PDBs from One CDB to Another in United Mode

In united mode, you can automatically move a PDB from one CDB to another (for example, for load balancing or adding new functionality).

If the PDB has TDE-encrypted tables or tablespaces, then you can set the `ONE_STEP_PLUGIN_FOR_PDB_WITH_TDE` dynamic system parameter to `TRUE` to enable the PDB to include the TDE keys in the PDB move operation. This parameter avoids you having to manually provide a keystore password when you import the TDE keys into the PDB after it has moved to a different CDB. When `ONE_STEP_PLUGIN_FOR_PDB_WITH_TDE` is set to `TRUE`, the database caches the keystore password in memory, obfuscated at the system level, and then uses it for the import operation. The default for `ONE_STEP_PLUGIN_FOR_PDB_WITH_TDE` is `FALSE`.

1. Log in to the PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Set the `ONE_STEP_PLUGIN_FOR_PDB_WITH_TDE` dynamic initialization parameter `TRUE`.

   For example:

   ```
   ALTER SYSTEM SET ONE_STEP_PLUGIN_FOR_PDB_WITH_TDE = TRUE;
   ```

3. Move (relocate) the PDB.

Related Topics

- *Oracle Multitenant Administrator's Guide*

Unplugging and Plugging a PDB with Encrypted Data in a CDB in United Mode

In united mode, for a PDB that has encrypted data, you can plug it into a CDB. Conversely, you can unplug this PDB from the CDB.

- Unplugging a PDB That Has Encrypted Data in United Mode
  In united mode, you can unplug a PDB with encrypted data and export it into an XML metadata file or an archive file.
- Plugging a PDB That Has Encrypted Data into a CDB in United Mode
  To plug a PDB that has encrypted data into a CDB, you first plug in the PDB and then you can set the keystore in the PDB.
- Unplugging a PDB That Has Master Encryption Keys Stored in a Hardware Keystore in United Mode
  You can unplug a PDB from one CDB that has been configured with an HSM and then plug it into another CDB also configured with an hardware keystore.

- Plugging a PDB That Has Master Encryption Keys Stored in a Hardware Keystore in United Mode
  The `ADMINISTER KEY MANAGEMENT` statement can import a TDE master encryption key from a hardware keystore to a PDB that has been moved to another CDB.

Unplugging a PDB That Has Encrypted Data in United Mode

In united mode, you can unplug a PDB with encrypted data and export it into an XML metadata file or an archive file.

The database that is unplugged contains data files and other associated files. You can check if a PDB has been unplugged by querying the `STATUS` column of the `DBA_PDBS` data dictionary view.

1. In the CDB root, query the `V$ENCRYPTION_WALLET` dynamic view to ensure that the keystore is open.
2. Use the `ENCRYPT USING` *`transport_secret`* clause in the `ALTER PLUGGABLE DATABASE` statement when you unplug the PDB.

   You must use this clause if the PDB has encrypted data. Otherwise, an `ORA-46680: master keys of the container database must be exported` error is returned.

   - For example, to export the PDB data into an XML file:

     ```
     ALTER PLUGGABLE DATABASE CDB1_PDB2
     UNPLUG INTO '/tmp/cdb1_pdb2.xml'
     ENCRYPT USING transport_secret;
     ```

   - To export the PDB data into an archive file:

     ```
     ALTER PLUGGABLE DATABASE CDB1_PDB2
     UNPLUG INTO '/tmp/cdb1_pdb2.pdb'
     ENCRYPT USING transport_secret;
     ```

Related Topics

- Opening the Software Keystore in a United Mode PDB
  To open a software keystore in united mode, you must use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEYSTORE OPEN` clause.
- *Oracle Multitenant Administrator's Guide*
- *Oracle Database SQL Language Reference*

Plugging a PDB That Has Encrypted Data into a CDB in United Mode

To plug a PDB that has encrypted data into a CDB, you first plug in the PDB and then you can set the keystore in the PDB.

When you plug an unplugged PDB into another CDB, the key version is set to `0` because this operation invalidates the history of the previous keys. You can check the key version by querying the `KEY_VERSION` column of the `V$ENCRYPTED_TABLESPACES` dynamic view. Similarly, if a control file is lost and recreated, then the previous history of the keys is reset to `0`. You can check if a PDB has already been plugged in by querying the `STATUS` column of the `DBA_PDBS` data dictionary view.

1. From the CDB root, create the PDB by plugging the unplugged PDB into the CDB.

To perform this operation for united mode, include the `DECRYPT USING *transport_secret*` clause.

You must use this clause if the metadata or archive file for the PDB has encrypted data. Otherwise, an `ORA-46680: master keys of the container database must be exported` error is returned.

- For example, if you had exported the PDB data into a metadata XML file:

```
CREATE PLUGGABLE DATABASE CDB1_PDB2
USING '/tmp/cdb1_pdb2.xml'
NOCOPY KEYSTORE
IDENTIFIED BY password
DECRYPT USING transport_secret;
```

- If you had exported the PDB into an archive file:

```
CREATE PLUGGABLE DATABASE CDB1_PDB2
USING '/tmp/cdb1_pdb2.pdb'
DECRYPT USING transport_secret;
```

During the open operation of the PDB after the plug operation, Oracle Database determines if the PDB has encrypted data. If so, it opens the PDB in the `RESTRICTED` mode.

If you want to create the PDB by cloning another PDB or from a non-CDB, and if the source database has encrypted data or a TDE master encryption key that has been set, then you must provide the keystore password by including the `keystore identified by *keystore_password*` clause in the `CREATE PLUGGABLE DATABASE ... FROM` SQL statement. You must provide this password even if the source database is using an auto-login software keystore. You can find if the source database has encrypted data or a TDE master encryption key set in the keystore by querying the `V$ENCRYPTION_KEYS` dynamic view

2. Open the PDB.

   For example:

```
ALTER PLUGGABLE DATABASE CDB1_PDB2 OPEN;
```

3. Open the keystore in the CDB root.

   For example:

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
IDENTIFIED BY password;
```

   Optionally, open the keystore in the PDB.

4. In either the CDB root or the PDB (if the keystore is open in the PDB), set the TDE master encryption key for the PDB.

   For example:

```
ADMINISTER KEY MANAGEMENT SET KEY
IDENTIFIED BY keystore_password
WITH BACKUP USING 'emp_key_backup';
```

Related Topics

- Opening the Software Keystore in a United Mode PDB
  To open a software keystore in united mode, you must use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEYSTORE OPEN` clause.

- Step 3: Set the TDE Master Encryption Key in the Software Keystore in United Mode
  To set the TDE master encryption key in the keystore when the PDB is configured in united mode, use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEY` clause.

Unplugging a PDB That Has Master Encryption Keys Stored in a Hardware Keystore in United Mode

You can unplug a PDB from one CDB that has been configured with an HSM and then plug it into another CDB also configured with an hardware keystore.

1. Unplug the PDB.

   You can check if a PDB has already been unplugged by querying the `STATUS` column of the `DBA_PDBS` data dictionary view.

2. Move the master encryption keys of the unplugged PDB in the hardware keystore that was used at the source CDB to the hardware keystore that is in use at the destination CDB.

   Refer to the documentation for the hardware keystore for information about moving master encryption keys between hardware keystores.

Related Topics

- *Oracle Multitenant Administrator's Guide*

Plugging a PDB That Has Master Encryption Keys Stored in a Hardware Keystore in United Mode

The `ADMINISTER KEY MANAGEMENT` statement can import a TDE master encryption key from a hardware keystore to a PDB that has been moved to another CDB.

1. Plug the unplugged PDB into the destination CDB that has been configured with the hardware keystore.

   You can check if a PDB has already been plugged in by querying the `STATUS` column of the `DBA_PDBS` data dictionary view.

   After the plug-in operation, the PDB that has been plugged in will be in restricted mode.

2. Ensure that the master encryption keys from the hardware keystore that has been configured with the source CDB are available in the hardware keystore of the destination CDB.

3. Log in to the plugged PDB as a user who was granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.

   For example:

   ```
   sqlplus sec_admin@hr_pdb as syskm
   Enter password: password
   Connected.
   ```

   Contact your `SYSDBA` administrator for the correct PDB. To check the current container, run the `SHOW CON_NAME` command.

4. Open the master encryption key of the plugged PDB.

For example, for a PDB called PDB1:

```
ALTER SESSION SET CONTAINER = PDB1;
ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
IDENTIFIED BY keystore_passsword;
```

5. Import the hardware keystore master encryption key into the PDB.

```
ADMINISTER KEY MANAGEMENT IMPORT ENCRYPTION KEYS
WITH SECRET "HSM" FROM 'HSM'
IDENTIFIED BY keystore_password;
```

6. Restart the PDB.

```
ALTER PLUGGABLE DATABASE PDB1 CLOSE;
ALTER PLUGGABLE DATABASE PDB1 OPEN;
```

Related Topics

- *Oracle Multitenant Administrator's Guide*

Managing Cloned PDBs with Encrypted Data in United Mode

In united mode, you can clone a PDB that has encrypted data in a CDB.

- About Managing Cloned PDBs That Have Encrypted Data in United Mode
  When you clone a PDB, you must make the master encryption key of the source PDB available to cloned PDB.
- Cloning a PDB with Encrypted Data in a CDB in United Mode
  The CREATE PLUGGABLE DATABASE statement with the KEYSTORE IDENTIFIED BY clause can clone a PDB that has encrypted data.
- Performing a Remote Clone of PDB with Encrypted Data Between Two CDBs in United Mode
  The CREATE PLUGGABLE DATABASE statement with the KEYSTORE IDENTIFIED BY clause can remotely clone a PDB that has encrypted data.
- Relocating Across CDBs a Cloned PDB with Encrypted Data in United Mode
  The CREATE PLUGGABLE DATABASE statement with the KEYSTORE IDENTIFIED BY clause can relocate across CDBs a cloned PDB that has encrypted data.

About Managing Cloned PDBs That Have Encrypted Data in United Mode

When you clone a PDB, you must make the master encryption key of the source PDB available to cloned PDB.

This allows a cloned PDB to operate on the encrypted data. To perform the clone, you do not need to export and import the keys because Oracle Database transports the keys for you even if the cloned PDB is in a remote CDB. However, you will need to provide the keystore password of the CDB where you are creating the clone.

If the PDBs have encrypted data, then you can perform remote clone operations on PDBs between CDBs, and relocate PDBs across CDBs.

Cloning a PDB with Encrypted Data in a CDB in United Mode

The CREATE PLUGGABLE DATABASE statement with the KEYSTORE IDENTIFIED BY clause can clone a PDB that has encrypted data.

1. In the CDB root, query the STATUS column of the V$ENCRYPTION_WALLET dynamic view to ensure that the keystore is open in the CDB root.
2. Log in to the PDB as a user who has been granted the ADMINISTER KEY MANAGEMENT or SYSKM privilege.
3. Use the CREATE PLUGGABLE DATABASE statement with the KEYSTORE IDENTIFIED BY clause to clone the PDB.

   For example:

   ```
   CREATE PLUGGABLE DATABASE cdb1_pdb3
   FROM cdb1_pdb1
   FILE_NAME_CONVERT=('cdb1_pdb1', 'pdb3/cdb1_pdb3') KEYSTORE
   IDENTIFIED BY keystore_password;
   ```

   Replace *keystore_password* with the password of the keystore of the CDB where the cdb1_pdb3 clone is created.

   After you create the cloned PDB, encrypted data is still accessible by the clone using the master encryption key of the original PDB. After a PDB is cloned, there may be user data in the encrypted tablespaces. This encrypted data is still accessible because the master encryption key of the source PDB is copied over to the destination PDB. Because the clone is a copy of the source PDB but will eventually follow its own course and have its own data and security policies, you should rekey the master encrytion key of the cloned PDB.
4. Rekey the master encryption key of the cloned PDB.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT SET KEY
   FORCE KEYSTORE
   IDENTIFIED BY keystore_password
   WITH BACKUP USING 'emp_key_backup';
   ```

   In this example, FORCE KEYSTORE is included because the keystore must be open during the rekey operation.

   Before you rekey the master encryption key of the cloned PDB, the clone can still use master encryption keys that belong to the original PDB. However, these master encryption keys do not appear in the cloned PDB V$ dynamic views. Rekeying the master encryption key ensures that the cloned PDB uses its own unique keys, which will be viewable in the V$ views.

Related Topics

- Opening the Software Keystore in a United Mode PDB
  To open a software keystore in united mode, you must use the ADMINISTER KEY MANAGEMENT statement with the SET KEYSTORE OPEN clause.

Performing a Remote Clone of PDB with Encrypted Data Between Two CDBs in United Mode

The CREATE PLUGGABLE DATABASE statement with the KEYSTORE IDENTIFIED BY clause can remotely clone a PDB that has encrypted data.

1. In the CDB root, query the STATUS column of the V$ENCRYPTION_WALLET dynamic view to ensure that the keystore is open in the CDB root.

2. Log in to the PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
3. Create a database link for the PDB that you want to clone remotely.

   Use the `CREATE DATABASE LINK` SQL statement to create the database link. You must create the database link by following the database link prerequisites that are required for cloning a remote PDB or a non-CDB.
4. Use the `CREATE PLUGGABLE DATABASE` statement with the `KEYSTORE IDENTIFIED BY` clause to perform the clone of the PDB.

   For example:

   ```
   CREATE PLUGGABLE DATABASE cdb1_pdb3
   FROM cdb1_pdb1
   FILE_NAME_CONVERT=('cdb1_pdb1', 'pdb3/cdb1_pdb3') KEYSTORE
   IDENTIFIED BY keystore_password;
   ```

   Replace *`keystore_password`* with the password of the keystore of the CDB where the `cdb1_pdb3` clone is created.

   After you create the cloned PDB, encrypted data is still accessible by the clone using the master encryption key of the original PDB. After a PDB is cloned, there may be user data in the encrypted tablespaces. This encrypted data is still accessible because the master encryption key of the source PDB is copied over to the destination PDB. Because the clone is a copy of the source PDB but will eventually follow its own course and have its own data and security policies, you should rekey the master encrytion key of the cloned PDB.
5. Rekey the master encryption key of the remotely cloned PDB.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT SET KEY
   FORCE KEYSTORE
   IDENTIFIED BY keystore_password
   WITH BACKUP USING 'emp_key_backup';
   ```

   In this example, `FORCE KEYSTORE` is included because the keystore must be open during the rekey operation.

   Before you rekey the master encryption key of the cloned PDB, the clone can still use master encryption keys that belong to the original PDB. However, these master encryption keys do not appear in the cloned PDB `V$` dynamic views. Rekeying the master encryption key ensures that the cloned PDB uses its own unique keys, which will be viewable in the `V$` views.

Related Topics

- Opening the Software Keystore in a United Mode PDB
  To open a software keystore in united mode, you must use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEYSTORE OPEN` clause.
- *Oracle Multitenant Administrator's Guide*
- *Oracle Multitenant Administrator's Guide*
- *Oracle Database SQL Language Reference*

Relocating Across CDBs a Cloned PDB with Encrypted Data in United Mode

The `CREATE PLUGGABLE DATABASE` statement with the `KEYSTORE IDENTIFIED BY` clause can relocate across CDBs a cloned PDB that has encrypted data.

1. In the CDB root, query the `STATUS` column of the `V$ENCRYPTION_WALLET` dynamic view to ensure that the keystore is open in the CDB root.
2. Log in to the PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
3. Create a database link for the PDB that you want to clone.

   Use the `CREATE DATABASE LINK` SQL statement to create the database link. You must create the database link by following the database link prerequisites that are required for cloning a remote PDB or a non-CDB.
4. Use the `CREATE PLUGGABLE DATABASE` statement with the `KEYSTORE IDENTIFIED BY` clause to clone the PDB.

   For example:

   ```
   CREATE PLUGGABLE DATABASE cdb1_pdb3
   FROM cdb1_pdb1
   FILE_NAME_CONVERT=('cdb1_pdb1', 'pdb3/cdb1_pdb3') KEYSTORE
   IDENTIFIED BY keystore_password;
   ```

   Replace *`keystore_password`* with the password of the keystore of the CDB where the `cdb1_pdb3` clone is created.

   After you create the cloned PDB, encrypted data is still accessible by the clone using the master encryption key of the original PDB. After a PDB is cloned, there may be user data in the encrypted tablespaces. This encrypted data is still accessible because the master encryption key of the source PDB is copied over to the destination PDB. Because the clone is a copy of the source PDB but will eventually follow its own course and have its own data and security policies, you should rekey the master encrytion key of the cloned PDB.
5. Use the `ALTER PLUGGABLE DATABASE` statement to perform the relocation operation.

   For example:

   ```
   ALTER PLUGGABLE DATABASE cdb1_pdb3
   OPEN RELOCATE TO 'need instance name example';
   ```
6. Rekey the master encryption key of the remotely cloned PDB.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT SET KEY
   FORCE KEYSTORE
   IDENTIFIED BY keystore_password
   WITH BACKUP USING 'emp_key_backup';
   ```

   In this example, `FORCE KEYSTORE` is included because the keystore must be open during the rekey operation.

   Before you rekey the master encryption key of the cloned PDB, the clone can still use master encryption keys that belong to the original PDB. However, these

master encryption keys do not appear in the cloned PDB V$ dynamic views. Rekeying the master encryption key ensures that the cloned PDB uses its own unique keys, which will be viewable in the V$ views.

Related Topics

- Opening the Software Keystore in a United Mode PDB
  To open a software keystore in united mode, you must use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEYSTORE OPEN` clause.
- *Oracle Multitenant Administrator's Guide*
- *Oracle Multitenant Administrator's Guide*
- *Oracle Database SQL Language Reference*

How Keystore Open and Close Operations Work in United Mode

You should be aware of how keystore open and close operations work in united mode.

For each PDB in united mode, you must explicitly open the password-protected software keystore or hardware keystore in the PDB to enable the Transparent Data Encryption operations to proceed. (Auto-login and local auto-login software keystores open automatically.) Closing a keystore on a PDB blocks all of the Transparent Data Encryption operations on that PDB.

The open and close keystore operations in a PDB depend on the open and close status of the keystore in the CDB root.

Note the following:

- You can create a separate keystore password for each PDB in united mode.
- Before you can manually open a software password-protected or hardware keystore in an individual PDB, you must open the keystore in the CDB root.
- If an auto-login keystore is in use, or if the keystore is closed, then include the `FORCE KEYSTORE` clause in the `ADMINISTER KEY MANAGEMENT` statement when you open or close the keystore.
- If the keystore is a password-protected software keystore that uses an external store for passwords, then set the `IDENTIFIED BY` clause to `EXTERNAL STORE`.
- Before you can set a TDE master encryption key in an individual PDB, you must set the key in the CDB root.
- Auto-login and local auto-login software keystores open automatically. You do not need to manually open these from the CDB root first, or from the PDB.
- If you close the keystore in the CDB root, then the keystores in the dependent PDBs also close. A keystore close operation in the root is the equivalent of performing a keystore close operation with the `CONTAINER` clause set to `ALL`.
- If you perform an `ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN` statement in the CDB root and set the `CONTAINER` clause to `ALL`, then the keystore will only be opened in each PDB that is configured in united mode, and any PDB that is configured in isolated mode is not opened.

Finding the Keystore Status for All of the PDBs in United Mode

You can create a convenience function that uses the V$ENCRYPTION_WALLET view to find the status for keystores in all PDBs in a CDB.

The V$ENCRYPTION_WALLET view displays the status of the keystore in a PDB, whether it is open, closed, uses a software or hardware keystore, and so on.

- To create a function that uses theV$ENCRYPTION_WALLET view to find the keystore status, use the CREATE PROCEDURE PL/SQL statement.

Example 5-2 shows how to create this function.

Example 5-2 Function to Find the Keystore Status of All of the PDBs in a CDB

```
CREATE OR REPLACE PROCEDURE all_pdb_v$encryption_wallet
IS
    err_occ            BOOLEAN;
    curr_pdb           VARCHAR2(30);
    pdb_name           VARCHAR2(30);
    wrl_type           VARCHAR2(20);
    status             VARCHAR2(30);
    wallet_type        VARCHAR2(20);
    wallet_order       VARCHAR2(12);
    fully_backed_up    VARCHAR2(15);
    wrl_parameter      VARCHAR2(4000);
    cursor sel_pdbs IS SELECT NAME FROM V$CONTAINERS
                       WHERE NAME <> 'PDB$SEED' order by con_id desc;
  BEGIN

    -- Store the original PDB name
    SELECT sys_context('userenv', 'con_name') INTO curr_pdb FROM DUAL;
    IF curr_pdb <> 'CDB$ROOT' THEN
      dbms_output.put_line('Operation valid in ROOT only');
    END IF;

    err_occ := FALSE;
    dbms_output.put_line('---');
    dbms_output.put_line('PDB_NAME                       WRL_TYPE STATUS
');
    dbms_output.put_line('------------------------------ --------
------------------------------');
    dbms_output.put_line('WALLET_TYPE          WALLET_ORDER FULLY_BACKED_UP');
    dbms_output.put_line('-------------------- ------------ ---------------');
    dbms_output.put_line('WRL_PARAMETER');

dbms_output.put_line('--------------------------------------------------------------------------------'
);
    FOR pdbinfo IN sel_pdbs LOOP

      pdb_name := DBMS_ASSERT.ENQUOTE_NAME(pdbinfo.name, FALSE);
      EXECUTE IMMEDIATE 'ALTER SESSION SET CONTAINER = ' || pdb_name;

      BEGIN
        pdb_name := rpad(substr(pdb_name,1,30), 30, ' ');
        EXECUTE IMMEDIATE 'SELECT wrl_type from V$ENCRYPTION_WALLET' into wrl_type;
        wrl_type := rpad(substr(wrl_type,1,8), 8, ' ');
        EXECUTE IMMEDIATE 'SELECT status from V$ENCRYPTION_WALLET' into status;
        status := rpad(substr(status,1,30), 30, ' ');
        EXECUTE IMMEDIATE 'SELECT wallet_type from V$ENCRYPTION_WALLET' into wallet_type;
```

```
        wallet_type := rpad(substr(wallet_type,1,20), 20, ' ');
        EXECUTE IMMEDIATE 'SELECT wallet_order from V$ENCRYPTION_WALLET' into
wallet_order;
        wallet_order := rpad(substr(wallet_order,1,9), 12, ' ');
        EXECUTE IMMEDIATE 'SELECT fully_backed_up from V$ENCRYPTION_WALLET' into fully_backed_up;
        fully_backed_up := rpad(substr(fully_backed_up,1,9), 15, ' ');
        EXECUTE IMMEDIATE 'SELECT wrl_parameter from V$ENCRYPTION_WALLET' into wrl_parameter;
        wrl_parameter := rpad(substr(wrl_parameter,1,79), 79, ' ');
        dbms_output.put_line(pdb_name || ' ' || wrl_type || ' ' || status);
        dbms_output.put_line(wallet_type || ' ' || wallet_order || ' ' || fully_backed_up);
        dbms_output.put_line(wrl_parameter);

      EXCEPTION
        WHEN OTHERS THEN
        err_occ := TRUE;
      END;
    END LOOP;

    IF err_occ = TRUE THEN
       dbms_output.put_line('One or more PDB resulted in an error');
    END IF;
  END;
.
/
set serveroutput on
exec all_pdb_v$encryption_wallet;
```

6

Managing Keystores and TDE Master Encryption Keys in Isolated Mode

In an Oracle Cloud database (but not an on-premises database), isolated mode enables you to create a keystore for each pluggable database (PDB).

- About Managing Keystores and TDE Master Encryption Keys in Isolated Mode
  In isolated mode, where a pluggable database (PDB) has its own keystore, you manage the keystore and its TDE master encryption keys from the PDB only.
- Operations That Are Allowed in Isolated Mode
  You can perform many `ADMINISTER KEY MANAGEMENT` operations in isolated mode.
- Operations That Are Not Allowed in an Isolated Mode PDB
  There are several `ADMINISTER KEY MANAGEMENT` operations that you cannot perform in an isolated mode PDB.
- Configuring the Keystore Location and Type for Isolated Mode
  For isolated mode, you can configure the keystore location and type by using only parameters or a combination of parameters and the `ALTER SYSTEM` statement.
- Configuring a Keystore and TDE Master Encryption Key in Isolated Mode
  In isolated mode, the software keystore is associated with a PDB.
- Configuring a Hardware Keystore in Isolated Mode
  There are two different types of hardware keystores that Oracle Database supports: hardware security modules (`HSM`) or Oracle Key Vault (`OKV`) keystores.
- Administering Keystores and TDE Master Encryption Keys in Isolated Mode
  After you create a keystore and a TDE master encryption key in isolated mode, you can perform administration tasks such as rekeying or tagging encryption keys.
- Administering Transparent Data Encryption in Isolated Mode
  You can perform a number of general administrative tasks with Transparent Data Encryption in isolated mode.

About Managing Keystores and TDE Master Encryption Keys in Isolated Mode

In isolated mode, where a pluggable database (PDB) has its own keystore, you manage the keystore and its TDE master encryption keys from the PDB only.

Similar to united mode, you must first configure a PDB to use isolated mode by setting the `WALLET_ROOT` and `TDE_CONFIGURATION` parameters. After you set these parameters, you can create and manage the keystore from the PDB. In this way, you can have the following scenario:

- The united mode settings in the CDB root will apply to all PDBs that do not have isolated mode settings. For example, the keystore that you create in the CDB root will be used by the root's associated united mode PDBs.

- The PDBs that are configured in isolated mode are allowed to independently create and manage their own keystore. An isolated mode PDB can have its own keystore, independent of the keystore of the CDB root.

This scenario is useful in cases where you have many PDBs that must use one type of keystore, but you have a few PDBs that must use a different type. By different types of keystores, this refers to either a TDE software keystore or to one of the hardware keystores that Oracle supports (for example, Oracle Key Vault or Cloud Key Management Service). You cannot have a mixture of different hardware keystore types in one CDB environment because the Oracle server can load only one PKCS#11 vendor library. If necessary, you can configure these PDBs in isolated mode so that each PDB can use its own keystore.

An advantage of configuring a PDB in isolated mode is that it improves the performance of rekey operations in the PDB as compared to the rekey performance in united mode when there are a large number of encrypted PDBs.

In a CDB when the number of encrypted PDBs is large, configuring a PDB in isolated mode allows the performance of the rekey operation in that PDB to remain similar to the performance of a rekey operation in a standalone system, and remain constant as the number of encrypted PDBs in the overall system increases.

Operations That Are Allowed in Isolated Mode

You can perform many `ADMINISTER KEY MANAGEMENT` operations in isolated mode.

These operations include creating, backing up, opening keystores; changing keystore passwords, merging keystores, closing keystores; creating, activating, tagging, moving, exporting, importing, and migrating encryption keys; and adding, updating, and deleting client secrets.

Table 6-1 describes the `ADMINISTER KEY MANAGEMENT` operations that you can perform in an isolated mode PDB.

Table 6-1 ADMINISTER KEY MANAGEMENT Isolated Mode Operations

Operation	Syntax	Isolated Mode Notes
Creating a keystore	`ADMINISTER KEY MANAGEMENT CREATE KEYSTORE ['keystore_location'] IDENTIFIED BY keystore_password;`	You can create password-protected, local auto-login, and auto-login keystores in an isolated mode PDB. The *keystore_location* clause is optional only when you have set the `WALLET_ROOT` parameter. Otherwise, it is mandatory.
Creating an auto-login keystore	`ADMINISTER KEY MANAGEMENT CREATE [LOCAL] AUTO_LOGIN KEYSTORE FROM KEYSTORE ['keystore_location'] IDENTIFIED BY keystore_password;`	The *keystore_location* is optional if the `WALLET_ROOT` parameter is set.

Table 6-1 (Cont.) ADMINISTER KEY MANAGEMENT Isolated Mode Operations

Operation	Syntax	Isolated Mode Notes
Opening a keystore	`ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN [FORCE KEYSTORE] IDENTIFIED BY [EXTERNAL STORE | keystore_password];`	In this operation, the EXTERNAL_STORE clause uses the password in the wallet. In a non-multitenant (standalone) environment, the wallet is configured at the location set by the EXTERNAL_KEYSTORE_CREDENTIAL_LOCATION initialization parameter. In a multitenant environment the wallet is configured in the tde_seps directory in the WALLET_ROOT location. This wallet contains the password of the keystore. For a PDB in isolated mode, the wallet used by the EXTERNAL_STORE clause must be configured at the *WALLET_ROOT*/*PDB_GUID*/tde_seps location.
Changing a keystore password	`ADMINISTER KEY MANAGEMENT ALTER KEYSTORE PASSWORD IDENTIFIED BY old_keystore_password SET new_keystore_password WITH BACKUP [USING 'backup_identifier'];`	-
Backing up a keystore	`ADMINISTER KEY MANAGEMENT BACKUP KEYSTORE [USING 'backup_identifier'] IDENTIFIED BY [EXTERNAL STORE | keystore_password] [TO 'keystore_location'];`	-
Merging the contents of one keystore into an existing keystore	`ADMINISTER KEY MANAGEMENT MERGE KEYSTORE 'keystore_location1' [IDENTIFIED BY keystore1_password] INTO EXISTING KEYSTORE 'keystore_location2' IDENTIFIED BY keystore2_password WITH BACKUP [USING 'backup_identifier'];`	-

Table 6-1 (Cont.) ADMINISTER KEY MANAGEMENT Isolated Mode Operations

Operation	Syntax	Isolated Mode Notes
Merging the contents of two keystores to create a third keystore	`ADMINISTER KEY MANAGEMENT MERGE KEYSTORE 'keystore_location1' [IDENTIFIED BY keystore1_password] AND KEYSTORE 'keystore_location2' [IDENTIFIED BY keystore2_password] INTO NEW KEYSTORE 'keystore_location3' IDENTIFIED BY keystore3_password;`	-
Closing a keystore	`ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE [IDENTIFIED BY [EXTERNAL STORE | keystore_password]];`	-
Closing the keystore of the CDB root when a PDB in isolated mode has its keystore open	`ADMINISTER KEY MANAGEMENT FORCE KEYSTORE CLOSE [IDENTIFIED BY [EXTERNAL STORE | keystore_password]];`	The FORCE clause allows the keystore to be closed in the CDB root even when a PDB in isolated mode still has its keystore open
Creating and activating a new TDE master encryption key (rekeying)	`ADMINISTER KEY MANAGEMENT SET [ENCRYPTION] KEY [USING TAG 'tag_name'] [FORCE KEYSTORE] IDENTIFIED BY [EXTERNAL STORE | keystore_password ]WITH BACKUP [USING 'backup_identifier'];`	-
Creating a user-defined TDE master encryption key for either now (SET) or later on (CREATE	`ADMINISTER KEY MANAGEMENT [SET | CREATE] [ENCRYPTION] KEY 'mkid:mk | mk' [USING ALGORITHM 'algorithm'] [FORCE KEYSTORE] [USING TAG 'tag_name'] IDENTIFIED BY [EXTERNAL STORE | keystore_password] [WITH BACKUP [USING 'backup_identifier']] [CONTAINER = CURRENT];`	-

Table 6-1 (Cont.) ADMINISTER KEY MANAGEMENT Isolated Mode Operations

Operation	Syntax	Isolated Mode Notes
Activating an existing TDE master encryption key	`ADMINISTER KEY MANAGEMENT USE [ENCRYPTION] KEY 'key_id' [USING TAG 'tag'] IDENTIFIED BY [EXTERNAL STORE | keystore_password] WITH BACKUP [USING 'backup_identifier'];`	-
Tagging a TDE master encryption key	`ADMINISTER KEY MANAGEMENT SET TAG 'tag' FOR 'key_id' IDENTIFIED BY [EXTERNAL STORE | keystore_password] WITH BACKUP [USING 'backup_identifier'];`	-
Exporting a TDE master encryption key	`ADMINISTER KEY MANAGEMENT EXPORT [ENCRYPTION] KEYS WITH SECRET secret TO 'filename' IDENTIFIED BY keystore_password [WITH IDENTIFIER IN { 'key_id' [, 'key_id' ]... | ( subquery ) }];`	-
Importing a TDE master encryption key	`ADMINISTER KEY MANAGEMENT IMPORT [ENCRYPTION] KEYS WITH SECRET secret FROM 'filename' IDENTIFIED BY keystore_password [WITH BACKUP [USING 'backup_identifier']];`	-
Migrating a TDE master encryption key from a software keystore to an HSM	`ADMINISTER KEY MANAGEMENT SET [ENCRYPTION] KEY IDENTIFIED BY HSM_auth_string [FORCE KEYSTORE] MIGRATE USING software_keystore_password WITH BACKUP [USING 'backup_identifier'];`	-

Table 6-1 (Cont.) ADMINISTER KEY MANAGEMENT Isolated Mode Operations

Operation	Syntax	Isolated Mode Notes
Reverse-migrating a TDE master encryption key from an HSM to a software keystore	`ADMINISTER KEY MANAGEMENT SET [ENCRYPTION] KEY IDENTIFIED BY *software_keystore_password* REVERSE MIGRATE USING *HSM_auth_string*;`	-
Adding a client secret	`ADMINISTER KEY MANAGEMENT ADD SECRET *'secret'* FOR CLIENT *'client_identifier'* [USING TAG *'tag_name'* ] IDENTIFIED BY [EXTERNAL STORE | *keystore_password*] WITH BACKUP [USING *'backup_identifier'*];`	-
Updating a client secret	`ADMINISTER KEY MANAGEMENT UPDATE SECRET *'secret'* FOR CLIENT *'client_identifier'* [USING TAG *'tag_name'* ] IDENTIFIED BY [EXTERNAL STORE | *keystore_password*] WITH BACKUP [USING *'backup_identifier'*];`	-
Deleting a client secret	`ADMINISTER KEY MANAGEMENT DELETE SECRET FOR CLIENT *'client_identifier'* IDENTIFIED BY [EXTERNAL STORE | *keystore_password*] WITH BACKUP [USING *'backup_identifier'*];`	-

Table 6-1 (Cont.) ADMINISTER KEY MANAGEMENT Isolated Mode Operations

Operation	Syntax	Isolated Mode Notes
Isolate a PDB	`ADMINISTER KEY MANAGEMENT ISOLATE KEYSTORE IDENTIFIED BY *isolated_keystore_password* FROM ROOT KEYSTORE [FORCE KEYSTORE] IDENTIFIED BY [EXTERNAL STORE | *united_keystore_password*] WITH BACKUP [USING *backup_id*];`	This operation performs two actions. First, it changes the `TDE_CONFIGURATION` of the PDB so that it is in isolated mode. Second, it moves the TDE master encryption key and all previously active (historical) TDE master encryption keys from the keystore of the CDB root to a newly-created keystore for the PDB having its own password, where the PDB will be able to manage its own keys.

Operations That Are Not Allowed in an Isolated Mode PDB

There are several `ADMINISTER KEY MANAGEMENT` operations that you cannot perform in an isolated mode PDB.

These operations include the following:

- Using the `CONTAINER = ALL` clause to create a new TDE master encryption key for later use in each pluggable database (PDB)
- Moving encryption keys from the keystore of the CDB root into a keystore of a PDB that is configured in isolated mode

Configuring the Keystore Location and Type for Isolated Mode

For isolated mode, you can configure the keystore location and type by using only parameters or a combination of parameters and the `ALTER SYSTEM` statement.

- Configuring Isolated Mode
  You can configure isolated mode by setting `WALLET_ROOT` in the initialization parameter file in the CDB root and `TDE_CONFIGURATION` in the PDB you want to isolate.
- Example: Restoring an Older Version of a Control File
  You can set `TDE_CONFIGURATION` if you have an older version of a control file that must be restored and only a few PDBs were configured in isolated mode.
- Example: Addressing the Problem of a Lost Control File
  You can address the problem of a lost control file by using the `ALTER SYSTEM` statement.

- Example: Configuring Isolated Mode in an Oracle Real Application Clusters Environment
  You can use ALTER SYSTEM to configure isolated mode in an Oracle Real Application Clusters (Oracle RAC) environment.

Configuring Isolated Mode

You can configure isolated mode by setting WALLET_ROOT in the initialization parameter file in the CDB root and TDE_CONFIGURATION in the PDB you want to isolate.

Configuring the WALLET_ROOT and TDE_CONFIGURATION parameters for the CDB environment is a similar procedure as the one you would use to configure united mode, except rather than using the RESET clause of the ALTER SYSTEM statement, you use the SET clause. You can perform the configuration by adding the WALLET_ROOT and TDE_CONFIGURATION parameters to the initialization parameter file. To configure a PDB in isolated mode, you must set a value for the TDE_CONFIGURATION parameter of the PDB, which you can do either by using the ALTER SYSTEM statement or by issuing the ADMINISTER KEY MANAGEMENT ISOLATE KEYSTORE statement. This section focuses on the use of the ALTER SYSTEM statement.

Depending on whether your system uses pfile or spfile, you must set the SCOPE clause in the ALTER SYSTEM statement appropriately when setting the value of the TDE_CONFIGURATION parameter for the PDB. The value of the TDE_CONFIGURATION parameter is a list of attribute-value pairs, and it is the value of the KEYSTORE_CONFIGURATION attribute that specifies the type of the keystore, as follows:

- FILE specifies a software keystore
- OKV specifies an Oracle Key Vault hardware keystore)
- HSM specifies a hardware security module keystore
- FILE|OKV specifies a reverse-migration from the OKV keystore type to the FILE keystore type has occurred
- FILE|HSM specifies a reverse-migration from the HSM keystore type to the FILE keystore type has occurred
- OKV|FILE specifies a migration from the FILE keystore type to the OKV keystore type has occurred
- HSM|FILE specifies a migration from the FILE keystore type to the HSM keystore type has occurred. HSM|FILE has two meanings: it either means that you are migrating from FILE to HSM, or it means that the configuration started out as using an HSM but is now using an auto-login HSM configuration, where the credentials of the HSM reside in a cwallet.sso file on the file system.

After you have used ALTER SYSTEM to configure the TDE_CONFIGURATION value for the selected PDB, the PDB in the CDB environment is in isolated mode. The steps in this procedure explain in detail how to configure an individual PDB to be in isolated mode, using its own keystore type.

1. Connect to the PDB as a user who has been granted the ADMINISTER KEY MANAGEMENT or SYSKM privilege.
2. Run the ALTER SYSTEM statement to configure the WALLET_ROOT and TDE_CONFIGURATION parameters for the CDB environment.
   - If the CDB root and the PDB are open, then set SCOPE to both:

```
ALTER SYSTEM SET
TDE_CONFIGURATION="KEYSTORE_CONFIGURATION=keystore_type" SCOPE=both;
```

- If the CDB root is open and the PDB is in the mount state, then set `scope` to `spfile`:

```
ALTER SYSTEM SET
TDE_CONFIGURATION="KEYSTORE_CONFIGURATION=keystore_type" SCOPE=spfile;
```

3. Check the configuration.
   - To check the `TDE_CONFIGURATION` parameter setting:

     ```
     SHOW PARAMETER TDE_CONFIGURATION
     ```

     The output should reflect the keystore configuration that you set for the current PDB. If it shows a different keystore configuration (for example, `FILE` if you had set it to `OKV`), then the setting may be showing the keystore configuration that was set for the CDB root, in united mode.
   - To check the keystore mode:

     ```
     SELECT KEYSTORE_MODE FROM V$ENCRYPTION_WALLET;
     ```

     The output should be `ISOLATED`.

After you configure isolated mode, the CDB root keystore that was available to the PDB when it was in united mode is no longer available to this PDB. At this stage, the PDB is configured to use its own keystore. If the `KEYSTORE_CONFIGURATION` parameter was `FILE` (meaning that the PDB is configured to use a software keystore), then the keystore location configured for the PDB is *`WALLET_ROOT`*`/PDB-GUID/tde`. If a keystore exists at that location and contains a TDE master encryption key, then that key is only available to this PDB, not to any other PDB. If no keystore exists at that location, you now can now proceed to create a software keystore and set a TDE master encryption key. If you later decide that you want the isolated mode PDB to become a united mode PDB again, then you can use the `ADMINISTER KEY MANAGEMENT UNITE KEYSTORE` statement. When you run `ADMINISTER KEY MANAGEMENT UNITE KEYSTORE`, it moves the keys from the PDB's keystore to the keystore of the CDB root, but it leaves any client secrets behind. So if there were no client secrets in the first place, then it would leave the PDB's keystore essentially "empty". It can now be backed up, and removed. Always back up keystores before you remove them, even empty keystores.

Related Topics

- Configuring United Mode by Editing the Initialization Parameter File
  You can configure united mode by setting both the `WALLET_ROOT` and `TDE_CONFIGURATION` parameters in the initialization parameter file.
- Configuring United Mode with the Initialization Parameter File and ALTER SYSTEM
  If your environment relies on server parameter files (spfile) or parameter files (pfile), then you can set `TDE_CONFIGURATION` using `ALTER SYSTEM` with `SCOPE`.
- Uniting a Pluggable Database Keystore
  Uniting a PDB keystore moves the TDE master encryption keys from the PDB keystore into the keystore of the CDB root. This enables the administrator of the keystore of the CDB root to manage the keys.

Example: Restoring an Older Version of a Control File

You can set TDE_CONFIGURATION if you have an older version of a control file that must be restored and only a few PDBs were configured in isolated mode.

When the CDB root and the PDB are both in the mount state, then you can only change the PDB's keystore configuration from the CDB root.

1. Log in to the CDB root as a user who was granted the SYSDBA administrative privilege.
2. For each PDB that you want to change, use the following syntax:

   ```
   ALTER SYSTEM SET
   TDE_CONFIGURATION="CONTAINER=pdb_name;KEYSTORE_CONFIGURATION=keystore_type"
   SCOPE=memory;
   ```

   For example, for the hrpdb and salespdb PDBs using FILE (for software keystores) as the keystore type:

   ```
   ALTER SYSTEM SET
   TDE_CONFIGURATION="CONTAINER=hrpdb;KEYSTORE_CONFIGURATION=FILE" SCOPE=memory;
   ALTER SYSTEM SET
   TDE_CONFIGURATION="CONTAINER=salespdb;KEYSTORE_CONFIGURATION=FILE"
   SCOPE=memory;
   ```

3. After you set the TDE_CONFIGURATION parameter for each PDB, log in to the CDB root and then set TDE_CONFIGURATION for the CDB root itself.

   ```
   ALTER SYSTEM SET TDE_CONFIGURATION="KEYSTORE_CONFIGURATION=FILE";
   ```

   At this stage, CDB root is in the mounted state. The value of the TDE_CONFIGURATION parameter that was set using ALTER SYSTEM with the CONTAINER attribute is only present in the memory of the CDB root. To ensure that the configuration is properly applied to each PDB, you must close and then reopen the PDB. When an isolated mode PDB is opened, the configuration set by the ALTER SYSTEM statement that was issued in the CDB root is read from the control file and then is automatically applied to the PDB.

4. Connect to each PDB and then close and reopen the PDB.

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

Example: Addressing the Problem of a Lost Control File

You can address the problem of a lost control file by using the ALTER SYSTEM statement.

Running these statements with SCOPE set to memory will store the CONTAINER value in memory. When you open the isolated PDB, this configuration will automatically be updated for the PDB.

If you are using an Oracle Data Guard environment, then to correct the control file, run these statements on both the primary and the standby databases.

1. Log in to the CDB root as a user who was granted the SYSDBA administrative privilege.

2. If you are unsure of the exact state of the system, then you should run `ALTER SYSTEM` with `RESET`.

   For example:

   ```
   ALTER SYSTEM RESET TDE_CONFIGURATION SCOPE=memory;
   ```

3. For each PDB that you want to change, use the following syntax:

   ```
   ALTER SYSTEM SET
   TDE_CONFIGURATION="CONTAINER=pdb_name;KEYSTORE_CONFIGURATION=FILE"
   SCOPE=memory;
   ```

   For example, for the `hrpdb` and `salespdb` PDBs with `FILE` (for software keystores) as the keystore type:

   ```
   ALTER SYSTEM SET
   TDE_CONFIGURATION="CONTAINER=hrpdb;KEYSTORE_CONFIGURATION=FILE"
   SCOPE=memory;
   ALTER SYSTEM SET
   TDE_CONFIGURATION="CONTAINER=salespdb;KEYSTORE_CONFIGURATION=FILE"
   SCOPE=memory;
   ```

4. After you set the `TDE_CONFIGURATION` parameter for each PDB, log in to the CDB root and then set `TDE_CONFIGURATION` for the CDB root itself.

   ```
   ALTER SYSTEM SET TDE_CONFIGURATION="KEYSTORE_CONFIGURATION=FILE";
   ```

   At this stage, CDB root is in the mounted state. The value of the `TDE_CONFIGURATION` parameter that was set using `ALTER SYSTEM` with the `CONTAINER` attribute is only present in the memory of the CDB root. To ensure that the configuration is properly applied to each PDB, you must close and then reopen the PDB. When an isolated mode PDB is opened, the configuration set by the `ALTER SYSTEM` statement that was issued in the CDB root is read from the control file and then is automatically applied to the PDB.

5. Connect to each PDB and then close and reopen the PDB.

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

Example: Configuring Isolated Mode in an Oracle Real Application Clusters Environment

You can use `ALTER SYSTEM` to configure isolated mode in an Oracle Real Application Clusters (Oracle RAC) environment.

- To ensure that the effect of the `ALTER SYSTEM` statement is applied on each Oracle RAC node, specify the wildcard (*) in the `SID` clause of the `ALTER SYSTEM` statement, as follows. You can run this statement from either the CDB root or a PDB.

  ```
  ALTER SYSTEM SET TDE_CONFIGURATION="KEYSTORE_CONFIGURATION=keystore_type"
  SID='*';
  ```

Configuring a Keystore and TDE Master Encryption Key in Isolated Mode

In isolated mode, the software keystore is associated with a PDB.

- About Configuring a Software Keystore in Isolated Mode
  You can create all types of software keystores in isolated mode: password-protected, password protected with the credential provided from an external store, auto-login, local auto-login.
- Step 1: Create a Software Keystore in a PDB Configured in Isolated Mode
  A password-protected software keystore requires a password to protect the keystore keys and credentials.
- Step 2: Open the Software Keystore in an Isolated Mode PDB
  To open a software keystore in isolated mode, you must use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEYSTORE OPEN` clause.
- Step 3: Set the TDE Master Encryption Key in the Software Keystore of the Isolated Mode PDB
  To set the TDE master encryption key in a software keystore in an isolated mode PDB, use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEY` clause.
- Step 4: Encrypt Your Data in Isolated Mode
  Now that you have completed the keystore configuration and the PDB is configured in isolated mode, you can begin to encrypt data in the PDB.

About Configuring a Software Keystore in Isolated Mode

You can create all types of software keystores in isolated mode: password-protected, password protected with the credential provided from an external store, auto-login, local auto-login.

To enable encryption in the PDB after it is configured in isolated mode with the `KEYSTORE_CONFIGURATION` attribute set to `FILE` (that is, to use a software keystore), you must create a software keystore, open the software keystore, and then set a TDE master encryption key in the software keystore. Afterward, you can begin to encrypt data for tables and tablespaces that will be accessible in the PDB.

In a multitenant environment, you can create a secure external store to hold the credentials of the software keystore. This feature enables you to hide the keystore password: it removes the need for storing the keystore password in any script or tool that accesses the database without user intervention, such as an overnight batch script. When the `WALLET_ROOT` parameter is specified, the location of the external store for the CDB root is *`WALLET_ROOT`*`/tde_seps` and for the PDB it is *`WALLET_ROOT`*`/PDB-GUID/tde_seps`. When the `WALLET_ROOT` parameter is set, there is no longer a single central external store, so when a keystore password is updated, the corresponding external store must be updated as well. When the `WALLET_ROOT` parameter is not specified, then the location of the external store is the same for both the CDB root and for every PDB. The external store location must then be set by the `EXTERNAL_KEYSTORE_CREDENTIAL_LOCATION` initialization parameter. When the `WALLET_ROOT` parameter is not specified, then there is a single central external store, so when you update the keystore password, only the central external store at the `EXTERNAL_KEYSTORE_CREDENTIAL_LOCATION` must be updated.

In a multitenant environment, different PDBs can access this external store location when you run the ADMINISTER KEY MANAGEMENT statement using the IDENTIFIED BY EXTERNAL STORE clause. This way, you can centrally locate the password and then update it only once in the external store.

Related Topics

- Configuring an External Store for a Keystore Password
  An external store for a keystore password stores the keystore password in a centrally accessed and managed location.

Step 1: Create a Software Keystore in a PDB Configured in Isolated Mode

A password-protected software keystore requires a password to protect the keystore keys and credentials.

1. Log in to the isolated mode PDB as a user who has been granted the ADMINISTER KEY MANAGEMENT or SYSKM privilege.

   For example:

   ```
   sqlplus sec_admin@hrpdb as syskm
   Enter password: password
   ```

   Contact your SYSDBA administrator for the correct PDB. To check the current container, run the SHOW CON_NAME command.

2. Run the ADMINISTER KEY MANAGEMENT SQL statement to create the keystore using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT CREATE KEYSTORE
   ['keystore_location']
   IDENTIFIED BY software_keystore_password;
   ```

   In this specification:

   - *keystore_location* is the path to the keystore directory location of the password-protected keystore. If the path that is set by the WALLET_ROOT parameter is the path that you want to use, then you can omit the *keystore_location* setting.

     If you specify the *keystore_location*, then enclose it in single quotation marks (' '). To find this location, you can query the WRL_PARAMETER column of the V$ENCRYPTION_WALLET view. (If the keystore was not created in the default location, then the STATUS column of the V$ENCRYPTION_WALLET view is NOT_AVAILABLE.)

   - *software_keystore_password* is the password of the keystore that you, the security administrator, creates.

   For example, to create the keystore in the default location (assuming WALLET_ROOT is set):

   ```
   ADMINISTER KEY MANAGEMENT CREATE KEYSTORE
   IDENTIFIED BY password;
   keystore altered.
   ```

After you complete these steps, the ewallet.p12 file, which is the keystore, appears in the default keystore location under the WALLET_ROOT which you have configured.

For example, if you had set the WALLET_ROOT parameter to $ORACLE_BASE/wallet and set the TDE_CONFIGURATION parameter to FILE (indicating that a software keystore is configured for the PDB that is in isolated mode), then the keystore of the PDB will be created in the $ORACLE_BASE/wallet/PDB-GUID/tde directory.

Step 2: Open the Software Keystore in an Isolated Mode PDB

To open a software keystore in isolated mode, you must use the ADMINISTER KEY MANAGEMENT statement with the SET KEYSTORE OPEN clause.

1. Log in to the isolated mode PDB as a user who has been granted the ADMINISTER KEY MANAGEMENT or SYSKM privilege.

   Contact your SYSDBA administrator for the correct PDB. To check the current container, run the SHOW CON_NAME command.

2. Run the ADMINISTER KEY MANAGEMENT statement to open the keystore.

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
   IDENTIFIED BY password;
   keystore altered.
   ```

   To switch over to opening the password-protected software keystore when an auto-login keystore is configured and is currently open, specify the FORCE KEYSTORE clause as follows.

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
   FORCE KEYSTORE
   IDENTIFIED BY EXTERNAL STORE;
   keystore altered.
   ```

   Here, the IDENTIFIED BY EXTERNAL STORE clause is included in the statement because the keystore credentials exist in an external store. This enables the password-protected keystore to be opened without specifying the keystore password within the statement itself.

   If the WALLET_ROOT parameter has been set, then Oracle Database finds the external store by searching in this path: *WALLET_ROOT*/*PDB_GUID*/tde_seps.

3. Confirm that the keystore is open.

   ```
   SELECT STATUS FROM V$ENCRYPTION_WALLET;
   ```

Related Topics

- About Opening Software Keystores
  A password-protected software keystore must be open before any TDE master encryption keys can be created or accessed in the keystore.

Step 3: Set the TDE Master Encryption Key in the Software Keystore of the Isolated Mode PDB

To set the TDE master encryption key in a software keystore in an isolated mode PDB, use the ADMINISTER KEY MANAGEMENT statement with the SET KEY clause.

1. Log in to the isolated mode PDB as a user who has been granted the ADMINISTER KEY MANAGEMENT or SYSKM privilege.
2. Ensure that the database is open in READ WRITE mode.

To find the status, run the `show pdbs` command.

3. Run the `ADMINISTER KEY MANAGEMENT` SQL statement to set the key in the software keystore.

   For example, if the keystore of the PDB is password-protected, the PDB is open, and the keystore of the PDB is open:

   ```
   ADMINISTER KEY MANAGEMENT SET KEY
   IDENTIFIED BY keystore_password
   WITH BACKUP USING 'emp_key_backup';

   keystore altered.
   ```

   If the keystore is closed:

   ```
   ADMINISTER KEY MANAGEMENT SET KEY
   FORCE KEYSTORE
   IDENTIFIED BY keystore_password
   WITH BACKUP USING 'emp_key_backup';

   keystore altered.
   ```

   In this specification:

   - `FORCE KEYSTORE` should be included if the keystore is closed. This automatically opens the keystore before setting the TDE master encryption key. The `FORCE KEYSTORE` clause also switches over to opening the password-protected software keystore when an auto-login keystore is configured and is currently open.
   - `IDENTIFIED BY` specifies the keystore password. Alternatively, if the keystore password is in an external store, you can use the `IDENTIFIED BY EXTERNAL STORE` clause.

4. Confirm that the TDE master encryption key is set.

   ```
   SELECT MASTERKEY_ACTIVATED FROM V$DATABASE_KEY_INFO;
   ```

   The output should be `YES`.

Related Topics

- About Setting the Software Keystore TDE Master Encryption Key
  The TDE master encryption key is stored in the keystore.

Step 4: Encrypt Your Data in Isolated Mode

Now that you have completed the keystore configuration and the PDB is configured in isolated mode, you can begin to encrypt data in the PDB.

Related Topics

- Encrypting Columns in Tables
  You can use Transparent Data Encryption to encrypt individual columns in database tables.
- Encryption Conversions for Tablespaces and Databases
  You can perform encryption operations on both offline and online tablespaces and databases.

Configuring a Hardware Keystore in Isolated Mode

There are two different types of hardware keystores that Oracle Database supports: hardware security modules (HSM) or Oracle Key Vault (OKV) keystores.

- About Configuring a Hardware Keystore in Isolated Mode
  You can configure a hardware keystore for a PDB when the PDB is configured in isolated mode.
- Step 1: Configure the Hardware Security Module for the Isolated Mode PDB
  To configure a third-party hardware security module, you must copy your vendor's PKCS#11 library to the correct location and follow your vendor's instructions.
- Step 2: Open the Hardware Keystore in an Isolated Mode PDB
  To open a hardware keystore in an isolated mode PDB, you must use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEYSTORE OPEN` clause.
- Step 3: Set TDE Master Encryption Key in the Hardware Keystore of a PDB in Isolated Mode
  After you have opened the hardware keystore in an isolated mode PDB, you are ready to set the TDE master encryption key for the PDB.
- Step 4: Encrypt Your Data in Isolated Mode
  Now that you have completed the keystore configuration and the PDB is configured in isolated mode, you can begin to encrypt data in the PDB.

About Configuring a Hardware Keystore in Isolated Mode

You can configure a hardware keystore for a PDB when the PDB is configured in isolated mode.

To configure a hardware keystore for a PDB in isolated mode, you first must set the `WALLET_ROOT` parameter. This is necessary for two reasons: first, to have support for migrating to a software keystore in the future, and second, because the configuration file for Oracle Key Vault is retrieved from a location under `WALLET_ROOT`. Afterwards, you must set the `KEYSTORE_CONFIGURATION` attribute of the `TDE_CONFIGURATION` parameter to `HSM` or `OKV`, open the configured hardware keystore, and then set the TDE master encryption key for the PDB. After you complete these tasks, you can begin to encrypt data in the PDB.

How you specify the `IDENTIFIED BY` clause when you run the `ADMINISTER KEY MANAGEMENT` statement depends on the type of hardware keystore. For a hardware security module (`HSM`), you use the following syntax:

```
IDENTIFIED BY "user_name:password"
```

For an Oracle Key Vault keystore, you can omit the *user_name* and colon, but you must enclose the password in quotation marks:

```
IDENTIFIED BY "password"
```

Step 1: Configure the Hardware Security Module for the Isolated Mode PDB

To configure a third-party hardware security module, you must copy your vendor's PKCS#11 library to the correct location and follow your vendor's instructions.

Related Topics

- Step 2: Configure the Hardware Security Module
  To configure a third-party hardware security module, you must copy the PKCS#11 library to the correct location and follow your vendor's instructions.

Step 2: Open the Hardware Keystore in an Isolated Mode PDB

To open a hardware keystore in an isolated mode PDB, you must use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEYSTORE OPEN` clause.

1. Log in to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.

   For example:

   ```
   sqlplus sec_admin@hrpdb as syskm
   Enter password: password
   ```

   Contact your `SYSDBA` administrator for the correct PDB. To check the current container, run the `SHOW CON_NAME` command.

2. Open the hardware keystore by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
   IDENTIFIED BY "hardware_keystore_credentials";
   ```

   The type of hardware keystore determines how you specify the hardware keystore password. For hardware security modules, you must use the *`user_name`*`:`*`password`* syntax. For example:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
   IDENTIFIED BY "psmith:password";

   keystore altered.
   ```

   For an Oracle Key Vault keystore, you can only provide the password. No user name is allowed in the `IDENTIFIED BY` clause. Enclose the password in double quotation marks.

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
   IDENTIFIED BY "password";
   ```

3. Repeat this procedure each time you restart the database instance.

   You must open the keystore of the CDB root first.

Related Topics

- About Opening Hardware Keystores
  You must open the hardware keystore so that it is accessible to the database before you can perform any encryption or decryption.

Step 3: Set TDE Master Encryption Key in the Hardware Keystore of a PDB in Isolated Mode

After you have opened the hardware keystore in an isolated mode PDB, you are ready to set the TDE master encryption key for the PDB.

- Setting a New TDE Master Encryption Key in Isolated Mode
  You should complete this procedure if you have not previously configured a hardware keystore for Transparent Data Encryption.
- Migration of a Previously Configured Encryption Key in Isolated Mode
  You must migrate the previously configured master encryption key if you previously configured a software keystore.

Setting a New TDE Master Encryption Key in Isolated Mode

You should complete this procedure if you have not previously configured a hardware keystore for Transparent Data Encryption.

1. Log in to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.

   Contact your `SYSDBA` administrator for the correct PDB. To check the current container, run the `SHOW CON_NAME` command.

2. Ensure that the database is open in `READ WRITE` mode.

   You can set the TDE master encryption key if `OPEN_MODE` is set to `READ WRITE`. To find the status, run the `show pdbs` command.

3. Set the new TDE master encryption key by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT SET KEY
   [USING TAG 'tag']
   [FORCE KEYSTORE]
   IDENTIFIED BY [EXTERNAL STORE | "hardware_keystore_credentials"];
   ```

   In this specification:

   - `FORCE KEYSTORE` temporarily opens the password-protected keystore for this operation if the keystore is closed if an auto-login keystore is configured and is currently open, or if a password-protected keystore is configured and is currently closed.
   - `IDENTIFIED BY` can be one of the following settings:
     - `EXTERNAL STORE` uses the keystore password stored in the external store to perform the keystore operation.
     - *`hardware_keystore_credentials`* refers to the credentials for either an HSM or an Oracle Key Vault hardware keystore. For an HSM, specify the credentials using this format, enclosed in quotation marks and separating the components with a colon: *"user_name:password"*, with *`user_name`* being the user who created the HSM and password being this user's password. For Oracle Key Vault, enter only the password of the user who created the keystore. Enclose this password with quotation marks.

   For example:

```
ADMINISTER KEY MANAGEMENT SET KEY
IDENTIFIED BY "psmith:password";

keystore altered.
```

4. Confirm that the TDE master encryption key is set.

```
SELECT MASTERKEY_ACTIVATED FROM V$DATABASE_KEY_INFO;
```

The output should be `YES`.

Related Topics

- About Setting the Hardware Keystore TDE Master Encryption Key
  You must create a TDE master encryption key that is stored inside the hardware keystore.

Migration of a Previously Configured Encryption Key in Isolated Mode

You must migrate the previously configured master encryption key if you previously configured a software keystore.

Related Topics

- Migration of a Previously Configured TDE Master Encryption Key
  You must migrate the previously configured TDE master encryption key if you previously configured a software keystore.

Step 4: Encrypt Your Data in Isolated Mode

Now that you have completed the keystore configuration and the PDB is configured in isolated mode, you can begin to encrypt data in the PDB.

Related Topics

- Encrypting Columns in Tables
  You can use Transparent Data Encryption to encrypt individual columns in database tables.
- Encryption Conversions for Tablespaces and Databases
  You can perform encryption operations on both offline and online tablespaces and databases.

Administering Keystores and TDE Master Encryption Keys in Isolated Mode

After you create a keystore and a TDE master encryption key in isolated mode, you can perform administration tasks such as rekeying or tagging encryption keys.

To change the password of a hardware keystore, you must use the administrative interface of the hardware keystore. You cannot perform this operation by using the `ADMINISTER KEY MANAGEMENT` statement.

- Changing the Keystore Password in Isolated Mode
  You can change the password of a software keystore when the PDB is in isolated mode.

- Backing Up a Password-Protected Software Keystore in Isolated Mode
  The `BACKUP KEYSTORE` clause of the `ADMINISTER KEY MANAGEMENT` statement backs up a password-protected software keystore.
- Merging Software Keystores in Isolated Mode
  In isolated mode, you can merge software keystores.
- Closing Keystores in Isolated Mode
  You can close both software and hardware keystores in isolated mode, unless the system tablespace is encrypted.
- Creating a User-Defined TDE Master Encryption Key in Isolated Mode
  To create a user-defined TDE master encryption key, use the `ADMINISTER KEY MANAGEMENT` statement with the `SET | CREATE [ENCRYPTION] KEY` clause.
- Creating a TDE Master Encryption Key for Later Use in Isolated Mode
  A keystore must be open before you can create a TDE master encryption key for use later on in isolated mode.
- Activating a TDE Master Encryption Key in Isolated Mode
  To activate a TDE master encryption key in isolated mode, you must open the keystore and use `ADMINISTER KEY MANAGEMENT` with the `USE KEY` clause.
- Rekeying the TDE Master Encryption Key in Isolated Mode
  You can use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEY` clause to rekey a TDE master encryption key.
- Moving a TDE Master Encryption Key into a New Keystore in Isolated Mode
  In isolated mode, you can move an existing TDE master encryption key into a new keystore from an existing software password keystore.
- Creating a Custom Attribute Tag in Isolated Mode
  To create a custom attribute tag in isolated mode, you must use the `SET TAG` clause of the `ADMINISTER KEY MANAGEMENT` statement.
- Exporting and Importing the TDE Master Encryption Key in Isolated Mode
  You can export and import the TDE master encryption key in different ways in isolated mode.
- Storing Oracle Database Secrets in Isolated Mode
  Secrets are data that support internal Oracle Database features that integrate external clients such as Oracle GoldenGate into the database.
- Migrating Keystores in Isolated Mode
  You can perform migration and reverse migration operations between software keystores and hardware keystores in isolated mode.
- Automatically Removing Inactive TDE Master Encryption Keys in Isolated Mode
  In isolated mode, the `REMOVE_INACTIVE_STANDBY_TDE_MASTER_KEY` initialization parameter can configure the automatic removal of inactive master encryption keys.
- Uniting a Pluggable Database Keystore
  Uniting a PDB keystore moves the TDE master encryption keys from the PDB keystore into the keystore of the CDB root. This enables the administrator of the keystore of the CDB root to manage the keys.
- Creating a Keystore When the PDB Is Closed
  When you create a keystore in a PDB that is closed, the new keystore is empty and the PDB is converted to isolated mode.

Changing the Keystore Password in Isolated Mode

You can change the password of a software keystore when the PDB is in isolated mode.

To change the password of a hardware keystore, you must use the administrative interface of the hardware keystore. You cannot perform this operation by using the `ADMINISTER KEY MANAGEMENT` statement.

- Changing the Password-Protected Software Keystore Password in Isolated Mode
  To change the password of a password-protected software keystore in isolated mode, you must use the `ADMINISTER KEY MANAGEMENT` statement.
- Changing the Password of a Hardware Keystore in Isolated Mode
  To change the password of a hardware keystore, you must close the hardware keystore and then change the password from the hardware keystore's management interface.

Changing the Password-Protected Software Keystore Password in Isolated Mode

To change the password of a password-protected software keystore in isolated mode, you must use the `ADMINISTER KEY MANAGEMENT` statement.

1. Log in to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Change the password for the keystore by using the following syntax:

```
ADMINISTER KEY MANAGEMENT ALTER KEYSTORE PASSWORD
[FORCE KEYSTORE]
IDENTIFIED BY
old_password SET new_password
[WITH BACKUP [USING 'backup_identifier']];
```

   In this specification:

   - `FORCE KEYSTORE` temporarily opens the password-protected keystore for this operation if the keystore is closed if an auto-login keystore is configured and is currently open, or if a password-protected keystore is configured and is currently closed.
   - *old_password* is the current keystore password that you want to change.
   - *new_password* is the new password that you set for the keystore.

   The following example creates a backup of the keystore and then changes the keystore password:

```
ADMINISTER KEY MANAGEMENT ALTER KEYSTORE PASSWORD
IDENTIFIED BY
old_password SET new_password
WITH BACKUP USING 'pwd_change';

keystore altered.
```

   This example performs the same operation but uses the `FORCE KEYSTORE` clause in case the auto-login software keystore is in use or the password-protected software keystore is closed.

```
ADMINISTER KEY MANAGEMENT ALTER KEYSTORE PASSWORD
FORCE KEYSTORE
IDENTIFIED BY
old_password SET new_password
WITH BACKUP USING 'pwd_change';

keystore altered.
```

Related Topics

- Performing Operations That Require a Keystore Password
  Many `ADMINISTER KEY MANAGEMENT` operations require access to a keystore password, for both software and hardware keystores.
- Changing the Password of a Software Keystore
  Oracle Database enables you to easily change password-protected software keystore passwords.

Changing the Password of a Hardware Keystore in Isolated Mode

To change the password of a hardware keystore, you must close the hardware keystore and then change the password from the hardware keystore's management interface.

1. Log in to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.

   Contact your `SYSDBA` administrator for the correct PDB. To check the current container, run the `SHOW CON_NAME` command.

2. Close the hardware keystore.

   For example, for an HSM:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE
   IDENTIFIED BY "psmith:password";
   ```

   For a hardware keystore whose password is stored externally:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE
   IDENTIFIED BY EXTERNAL STORE;
   ```

3. From the hardware security module management interface, change the hardware security module password.

4. Update the credentials of the HSM in the external store to use "*user_name*:*password*".

   Currently, the external store contains the old HSM credentials, which would no longer work.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT
   UPDATE SECRET 'user_name:password'
   FOR CLIENT 'TDE_WALLET'
   TO LOCAL AUTO_LOGIN KEYSTORE '/etc/ORACLE/WALLETS/orcl/external_store';
   ```

5. In SQL*Plus, open the hardware keystore.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
   IDENTIFIED BY "psmith:new_password";
   ```

For a hardware keystore whose password is stored externally:

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
IDENTIFIED BY EXTERNAL STORE;
```

Related Topics

- Performing Operations That Require a Keystore Password
  Many `ADMINISTER KEY MANAGEMENT` operations require access to a keystore password, for both software and hardware keystores.
- Storing Oracle Database Secrets in Isolated Mode
  Secrets are data that support internal Oracle Database features that integrate external clients such as Oracle GoldenGate into the database.

Backing Up a Password-Protected Software Keystore in Isolated Mode

The `BACKUP KEYSTORE` clause of the `ADMINISTER KEY MANAGEMENT` statement backs up a password-protected software keystore.

1. Log in to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.

   For example:

   ```
   sqlplus sec_admin@hrpdb as syskm
   Enter password: password
   Connected.
   ```

   Contact your `SYSDBA` administrator for the correct PDB. To check the current container, run the `SHOW CON_NAME` command.

2. Back up the keystore by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT BACKUP KEYSTORE
   [USING 'backup_identifier']
   [FORCE KEYSTORE]
   IDENTIFIED BY [EXTERNAL STORE | software_keystore_password]
   [TO 'keystore_location'];
   ```

   In this specification:

   - `USING` *`backup_identifier`* is an optional string that you can provide to identify the backup. Enclose this identifier in single quotation marks (`' '`). This identifier is appended to the named keystore file (for example, `ewallet_`*`time-stamp`*`_emp_key_backup.p12`).
   - `FORCE KEYSTORE` temporarily opens the password-protected keystore for this operation. You must open the keystore for this operation.
   - `IDENTIFIED BY` is required for the `BACKUP KEYSTORE` operation on a password-protected keystore because although the backup is simply a copy of the existing keystore, the status of the TDE master encryption key in the password-protected keystore must be set to `BACKED UP` and for this change the keystore password is required.
   - *`keystore_location`* is the path at which the backup keystore is stored. This setting is restricted to the PDB when the PDB lockdown profile `EXTERNAL_FILE_ACCESS` setting is blocked in the PDB or when the `PATH_PREFIX` variable was set when the PDB was created. If you do not specify the

keystore_location, then the backup is created in the same directory as the original keystore. Enclose this location in single quotation marks (' ').

The following example backs up a software keystore in the same location as the source keystore.

```
ADMINISTER KEY MANAGEMENT BACKUP KEYSTORE
USING 'hr.emp_keystore'
FORCE KEYSTORE
IDENTIFIED BY
'software_keystore_password';

keystore altered.
```

In the following version, the password for the keystore is stored externally, so the `EXTERNAL STORE` clause is used.

```
ADMINISTER KEY MANAGEMENT BACKUP KEYSTORE
USING 'hr.emp_keystore'
FORCE KEYSTORE
IDENTIFIED BY EXTERNAL STORE;
```

After you run this statement, an `ewallet_`*identifier*`.p12` file (for example, `ewallet_`*time-stamp*`_hr.emp_keystore.p12`) appears in the keystore backup location.

Related Topics

- Backing Up Password-Protected Software Keystores
  When you back up a password-protected software keystore, you can create a backup identifier string to describe the backup type.

Merging Software Keystores in Isolated Mode

In isolated mode, you can merge software keystores.

- Merging One Software Keystore into an Existing Software Keystore in Isolated Mode
  In isolated mode, you can use the `ADMINISTER KEY MANAGEMENT` statement with the `MERGE KEYSTORE` clause to merge one software keystore into another existing software keystore.
- Merging Two Software Keystores into a Third New Keystore in Isolated Mode
  In isolated mode, you can merge two software keystores into a third new keystore, so that the two existing keystores are not changed and the new keystore contains the keys of both source keystores.

Merging One Software Keystore into an Existing Software Keystore in Isolated Mode

In isolated mode, you can use the `ADMINISTER KEY MANAGEMENT` statement with the `MERGE KEYSTORE` clause to merge one software keystore into another existing software keystore.

1. Log in to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.

Contact your SYSDBA administrator for the correct PDB. To check the current container, run the SHOW CON_NAME command.

2. Merge the software keystores by using the following syntax:

```
ADMINISTER KEY MANAGEMENT MERGE KEYSTORE 'keystore1_location'
[IDENTIFIED BY software_keystore1_password]
INTO EXISTING KEYSTORE 'keystore2_location'
IDENTIFIED BY software_keystore2_password
[WITH BACKUP [USING 'backup_identifier]];
```

In this specification:

- *keystore1_location* is the directory location of the first keystore, which will be left unchanged after the merge. Enclose this path in single quotation marks (' ').
- The IDENTIFIED BY clause is required for the first keystore if it is a password-protected keystore. *software_keystore1_password* is the password for the first keystore.
- *keystore2_location* is the directory location of the second keystore into which the first keystore is to be merged. Enclose this path in single quotation marks (' ').
- *software_keystore2_password* is the password for the second keystore.

The target keystore (*keystore2*) remains a password-protected keystore after the keystore merge operation.

Related Topics

- About Merging Software Keystores
  You can merge any combination of software keystores, but the merged keystore must be password-protected. It can have a password that is different from the constituent keystores.

Merging Two Software Keystores into a Third New Keystore in Isolated Mode

In isolated mode, you can merge two software keystores into a third new keystore, so that the two existing keystores are not changed and the new keystore contains the keys of both source keystores.

1. Log in to the isolated mode PDB as a user who has been granted the ADMINISTER KEY MANAGEMENT or SYSKM privilege.

   Contact your SYSDBA administrator for the correct PDB. To check the current container, run the SHOW CON_NAME command.

2. Merge the keystores by using the following syntax:

```
ADMINISTER KEY MANAGEMENT MERGE KEYSTORE 'keystore1_location'
[IDENTIFIED BY software_keystore1_password] AND KEYSTORE
'keystore2_location'
[IDENTIFIED BY software_keystore2_password]
INTO NEW KEYSTORE 'keystore3_location'
IDENTIFIED BY software_keystore3_password;
```

In this specification:

- *keystore1_location* is the directory location of the first keystore, which will be left unchanged after the merge. Enclose this path in single quotation marks (' ').
- The IDENTIFIED BY clause is required for the first keystore if it is a password-protected keystore. *software_keystore1_password* is the current password for the first keystore.
- *keystore2_location* is the directory location of the second keystore. Enclose this path in single quotation marks (' ').
- The IDENTIFIED BY clause is required for the second keystore if it is a password-protected keystore. *software_keystore2_password* is the current password for the second keystore.
- *keystore3_location* specifies the directory location of the new, merged keystore. Enclose this path in single quotation marks (' '). If there is already an existing keystore at this location, the command exits with an error.
- *software_keystore3_password* is the new password for the merged keystore.

The following example merges an auto-login software keystore with a password-protected keystore to create a merged password-protected keystore at a new location:

```
ADMINISTER KEY MANAGEMENT MERGE KEYSTORE '/etc/ORACLE/KEYSTORE/DB1'
AND KEYSTORE '/etc/ORACLE/KEYSTORE/DB2'
IDENTIFIED BY existing_password_for_keystore_2
INTO NEW KEYSTORE '/etc/ORACLE/KEYSTORE/DB3'
IDENTIFIED BY new_password_for_keystore_3;

keystore altered.
```

Related Topics

- About Merging Software Keystores
  You can merge any combination of software keystores, but the merged keystore must be password-protected. It can have a password that is different from the constituent keystores.

Closing Keystores in Isolated Mode

You can close both software and hardware keystores in isolated mode, unless the system tablespace is encrypted.

- Closing a Software Keystore in Isolated Mode
  You can close password-protected keystores, auto-login keystores, and local auto-login software keystores in isolated mode.
- Closing a Hardware Keystore in Isolated Mode
  To close a hardware keystore, you must use the ADMINISTER KEY MANAGEMENT statement with the SET KEYSTORE CLOSE clause.

Closing a Software Keystore in Isolated Mode

You can close password-protected keystores, auto-login keystores, and local auto-login software keystores in isolated mode.

In the case of an auto-login keystore, which opens automatically when it is accessed, you must first move it to a new location where it cannot be automatically opened, then

you must manually close it. You must do this if you are changing your configuration from an auto-login keystore to a password-protected keystore: you change the configuration to stop using the auto-login keystore (by moving the auto-login keystore to another location where it cannot be automatically opened), and then closing the auto-login keystore.

1. Log in to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Close the software keystore by using the following syntax.

   Note that the only difference between the following two `ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE` statements is that a password must be provided for a password-protected keystore.

   - For a password-protected software keystore:

     ```
     ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE
     IDENTIFIED BY [EXTERNAL STORE | software_keystore_password];
     ```

     Closing a password-protected keystore disables all encryption and decryption operations. Any attempt to encrypt or decrypt data or access encrypted data results in an error.

   - For an auto-login or local auto-login software keystore:

     ```
     ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE;
     ```

     The result of this statement will not necessarily be that the keystore status will change to `CLOSED`, because unless you also moved the `cwallet.sso` file to a location that Oracle Database cannot find, then a background job or background process could automatically re-open the auto-login keystore. This can cause the status to potentially always appear to be `OPEN` even after the `ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE` statement completed successfully.

Related Topics

- About Closing Keystores
  After you open a keystore, it remains open until you shut down the database instance.

Closing a Hardware Keystore in Isolated Mode

To close a hardware keystore, you must use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEYSTORE CLOSE` clause.

For an Oracle Key Vault keystore, you can only provide the password. No user name is allowed in the `IDENTIFIED BY` clause. Enclose the password in double quotation marks.

1. Log in to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.

   Contact your `SYSDBA` administrator for the correct PDB. To check the current container, run the `SHOW CON_NAME` command.

2. Close the hardware keystore by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE
   IDENTIFIED BY [EXTERNAL STORE | "hardware_keystore_credentials"];
   ```

For example, for an HSM:

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE
IDENTIFIED BY "psmith:password";
```

Closing a keystore disables all encryption and decryption operations. Any attempt to encrypt or decrypt data or access encrypted data results in an error.

Related Topics

- About Closing Keystores
  After you open a keystore, it remains open until you shut down the database instance.

Creating a User-Defined TDE Master Encryption Key in Isolated Mode

To create a user-defined TDE master encryption key, use the `ADMINISTER KEY MANAGEMENT` statement with the `SET | CREATE [ENCRYPTION] KEY` clause.

1. Log in to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Create the user-defined TDE master encryption key by using the following syntax:

```
ADMINISTER KEY MANAGEMENT SET | CREATE [ENCRYPTION] KEY
'mkid:mk | mk'
[USING ALGORITHM 'algorithm']
[FORCE KEYSTORE]
IDENTIFIED BY [EXTERNAL STORE | keystore_password]
[WITH BACKUP [USING 'backup_identifier']];
```

In this specification:

- `SET | CREATE` : Enter `SET` if you want to create the master and activate the TDE master encryption key now, or enter `CREATE` if you want to create the key for later use, without activating it yet.
- *mkid* and *mk*:
  - *mkid*, the TDE master encryption key ID, is a 16–byte hex-encoded value that you can specify or have Oracle Database generate.
  - *mk*, the TDE master encryption key, is a hex-encoded value that you can specify or have Oracle Database generate, either 32 bytes (for the for `AES256`, `ARIA256`, and `GOST256` algorithms) or 16 bytes (for the `SEED128` algorithm).

  If you omit the *mkid*:*mk*|*mkid* clause but include the *mk* value, then Oracle Database generates the *mkid* for the *mk*.

  If you omit the entire *mkid*:*mk*|*mkid* clause, then Oracle Database generates these values for you.
- `USING ALGORITHM`: Specify one of the following supported algorithms:
  - `AES256`
  - `ARIA256`
  - `SEED128`
  - `GOST256`

  If you omit the algorithm, then the default, `AES256`, is used.

- `FORCE KEYSTORE` temporarily opens the password-protected keystore for this operation. You must open the keystore for this operation.

The following example includes a user-created TDE master encryption key but no TDE master encryption key ID, so that the TDE master encryption key ID is generated:

```
ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY
'3D432109DF88967A541967062A6F4E460E892318E307F017BA048707B402493C'
USING ALGORITHM 'GOST256'
FORCE KEYSTORE
IDENTIFIED BY keystore_password;
```

The next example creates user-defined keys for both the master encryption ID and the TDE master encryption key. It omits the algorithm specification, so the default algorithm `AES256` is used.

```
ADMINISTER KEY MANAGEMENT CREATE ENCRYPTION KEY
'10203040506070801112131415161718:3D432109DF88967A541967062A6F4E460E892318E30
7F017BA048707B402493C'
IDENTIFIED BY keystore_password;
```

Related Topics

- Supported Encryption and Integrity Algorithms
  The supported Advanced Encryption Standard cipher keys, including tablespace and database encryption keys, can be either 128, 192, or 256 bits long. Tablespace and database encryption use the 128–bit length cipher key.

Creating a TDE Master Encryption Key for Later Use in Isolated Mode

A keystore must be open before you can create a TDE master encryption key for use later on in isolated mode.

1. Log in to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.

   Contact your `SYSDBA` administrator for the correct PDB. To check the current container, run the `SHOW CON_NAME` command.

2. Create the TDE master encryption key by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT CREATE KEY
   [USING TAG 'tag']
   [FORCE KEYSTORE]
   IDENTIFIED BY [EXTERNAL STORE | keystore_password]
   [WITH BACKUP [USING 'backup_identifier']];
   ```

   In this specification:

   - `FORCE KEYSTORE` temporarily opens the password-protected keystore for this operation. You must open the keystore for this operation.
   - `IDENTIFIED BY` is required for the `BACKUP KEYSTORE` operation on a password-protected keystore because although the backup is simply a copy of the existing keystore, the status of the TDE master encryption key in the password-protected keystore must be set to `BACKED UP` and for this change the keystore password is required.
   - *keystore_location* is the path at which the backup keystore is stored. This setting is restricted to the PDB when the PDB lockdown profile

EXTERNAL_FILE_ACCESS setting is blocked in the PDB or when the PATH_PREFIX variable was not set when the PDB was created. If you do not specify the *keystore_location*, then the backup is created in the same directory as the original keystore. Enclose this location in single quotation marks (' ').

For example:

```
ADMINISTER KEY MANAGEMENT CREATE KEY
FORCE KEYSTORE
IDENTIFIED BY keystore_password
WITH BACKUP;
```

3. If necessary, activate the TDE master encryption key.

   a. Find the key ID.

   ```
   SELECT KEY_ID FROM V$ENCRYPTION_KEYS;

   KEY_ID
   ------------------------------------------------
   AWsHwVYC2U+Nv3RVphn/yAIAAAAAAAAAAAAAAAAAAAAAAAAAAA
   ```

   b. Use this key ID to activate the key.

   ```
   ADMINISTER KEY MANAGEMENT USE KEY
   'AWsHwVYC2U+Nv3RVphn/yAIAAAAAAAAAAAAAAAAAAAAAAAAAAA'
   USING TAG 'quarter:second;description:Activate Key on standby'
   IDENTIFIED BY password
   WITH BACKUP;
   ```

Related Topics

- Creating TDE Master Encryption Keys for Later Use
  You can create a TDE master encryption key that can be activated at a later date.

Activating a TDE Master Encryption Key in Isolated Mode

To activate a TDE master encryption key in isolated mode, you must open the keystore and use ADMINISTER KEY MANAGEMENT with the USE KEY clause.

1. Log in to the isolated mode PDB as a user who has been granted the ADMINISTER KEY MANAGEMENT or SYSKM privilege.
2. Query the KEY_ID column of the V$ENCRYPTION_KEYS view to find the key identifier.

   For example:

   ```
   SELECT KEY_ID FROM V$ENCRYPTION_KEYS;

   KEY_ID
   ------------------------------------------------
   ARaHD762tUkkvyLgPzAi6hMAAAAAAAAAAAAAAAAAAAAAAAAAAA
   ```

3. Use this key identifier to activate the TDE master encryption key by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT USE KEY 'key_identifier_from_V$ENCRYPTION_KEYS'
   [USING TAG 'tag']
   [FORCE KEYSTORE]
   IDENTIFIED BY [EXTERNAL STORE | keystore_password]
   [WITH BACKUP [USING 'backup_identifier']];
   ```

   In this specification:

- `FORCE KEYSTORE` temporarily opens the password-protected keystore for this operation. You must open the keystore for this operation.

For example:

```
ADMINISTER KEY MANAGEMENT USE KEY
'ARaHD762tUkkvyLgPzAi6hMAAAAAAAAAAAAAAAAAAAAAAAAAAAAA'
FORCE KEYSTORE
IDENTIFIED BY keystore_password
WITH BACKUP;
```

Related Topics

- About Activating TDE Master Encryption Keys
  You can activate a previously created or imported TDE master encryption key by using the `USE KEY` clause of `ADMINSTER KEY MANAGEMENT`.

Rekeying the TDE Master Encryption Key in Isolated Mode

You can use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEY` clause to rekey a TDE master encryption key.

1. Log in to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.

   Contact your `SYSDBA` administrator for the correct PDB. To check the current container, run the `SHOW CON_NAME` command.

2. If you are rekeying the TDE master encryption key for a keystore that has auto login enabled, then ensure that both the auto login keystore, identified by the `.sso` file, and the encryption keystore, identified by the `.p12` file, are present.

   You can find the location of these files by querying the `WRL_PARAMETER` column of the `V$ENCRYPTION_WALLET` view. To find the `WRL_PARAMETER` values for all of the database instances, query the `GV$ENCRYPTION_WALLET` view.

3. Rekey the TDE master encryption key by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT SET [ENCRYPTION] KEY
   [FORCE KEYSTORE]
   [USING TAG 'tag_name']
   IDENTIFIED BY [EXTERNAL STORE | keystore_password]
   [WITH BACKUP [USING 'backup_identifier']];
   ```

   In this specification:

   - *tag* is the associated attributes and information that you define. Enclose this setting in single quotation marks (`' '`).
   - `FORCE KEYSTORE` temporarily opens the password-protected keystore for this operation. You must open the keystore for this operation.
   - `IDENTIFIED BY` can be one of the following settings:
     - `EXTERNAL STORE` uses the keystore password stored in the external store to perform the keystore operation.
     - *keystore_password* is the password that was created for this keystore.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT SET KEY
   FORCE KEYSTORE
   ```

```
IDENTIFIED BY keystore_password
WITH BACKUP USING 'emp_key_backup';

keystore altered.
```

Related Topics

- About Rekeying the TDE Master Encryption Key
  Oracle Database uses a unified TDE Master Encryption Key for both TDE column encryption and TDE tablespace encryption.

Moving a TDE Master Encryption Key into a New Keystore in Isolated Mode

In isolated mode, you can move an existing TDE master encryption key into a new keystore from an existing software password keystore.

This feature enables you to move a subset, or all, of a keystore's keys into a new keystore. After you move the key or keys to the new keystore and then back up the old keystore, optionally you then can delete this old keystore.

1. Log in to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Query the `CREATION_TIME` and `KEY_ID` columns of the `V$ENCRYPTION_KEYS` view to find the key identifier of the key that you want to move.

   For example:

   ```
   SELECT CREATION_TIME, KEY_ID FROM V$ENCRYPTION_KEYS;

   CREATION TIME
   ---------------------------------------------------
   22-SEP-17 08.55.12.956170 PM +00:00

   KEY_ID
   ---------------------------------------------------
   ARaHD762tUkkvyLgPzAi6hMAAAAAAAAAAAAAAAAAAAAAAAAAAAA
   ```

3. Move the key into a new keystore by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT
   MOVE [ENCRYPTION] KEYS
   TO NEW KEYSTORE 'keystore_location1'
   IDENTIFIED BY keystore1_password
   FROM [FORCE] KEYSTORE
   IDENTIFIED BY keystore_password
   [WITH IDENTIFIER IN
   { 'key_identifier' [, 'key_identifier' ]... | ( subquery ) }]
   [WITH BACKUP [USING 'backup_identifier'] ];
   ```

   In this specification:

   - *keystore_location1* is the path to the `wallet` directory that will store the new keystore `.p12` file. By default, this directory is `$ORACLE_BASE/admin/`*db_unique_name*`/wallet`.
   - `FORCE` temporarily opens the keystore for this operation.
   - *keystore_password* is the password for the keystore from which the key is moving.

For example:

```
ADMINISTER KEY MANAGEMENT MOVE KEYS
TO NEW KEYSTORE '$ORACLE_BASE/admin/orcl/wallet'
IDENTIFIED BY keystore_password
FROM FORCE KEYSTORE
IDENTIFIED BY keystore_password
WITH IDENTIFIER IN
(SELECT KEY_ID FROM V$ENCRYPTION_KEYS WHERE ROWNUM < 2)
WITH BACKUP;
```

After the keys are moved to the new keystore, they no longer exist in the old keystore.

4. To delete the old keystore, go to the `wallet` directory and do the following:
   a. Back up the `.p12` file containing the keystore that you want to delete.
   b. Manually delete the `.p12` file containing the keystore.

   To find the location of the keystore, open the keystore, and then query the `WRL_PARAMETER` column of the `V$ENCRYPTION_WALLET` view.

Related Topics

- Dangers of Deleting Keystores
  Oracle strongly recommends that you do not delete keystores.

Creating a Custom Attribute Tag in Isolated Mode

To create a custom attribute tag in isolated mode, you must use the `SET TAG` clause of the `ADMINISTER KEY MANAGEMENT` statement.

1. Log in to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. If necessary, query the `TAG` column of the `V$ENCRYPTION_KEY` dynamic view to find a listing of existing tags for the TDE master encryption keys.

   When you create a new tag for a TDE master encryption key, it overwrites the existing tag for that TDE master encryption key.
3. Create the custom attribute tag by using the following syntax:

```
ADMINISTER KEY MANAGEMENT SET TAG 'tag'
FOR 'master_key_identifier'
[FORCE KEYSTORE]
IDENTIFIED BY [EXTERNAL STORE | keystore_password]
[WITH BACKUP [USING 'backup_identifier']];
```

   In this specification:

   - *tag* is the associated attributes or information that you define. Enclose this information in single quotation marks (`' '`).
   - *master_key_identifier* identifies the TDE master encryption key for which the *tag* is set. To find a list of TDE master encryption key identifiers, query the `KEY_ID` column of the `V$ENCRYPTION_KEYS` dynamic view.
   - `FORCE KEYSTORE` temporarily opens the password-protected keystore for this operation. You must open the keystore for this operation.
   - `IDENTIFIED BY` can be one of the following settings:

- `EXTERNAL STORE` uses the keystore password stored in the external store to perform the keystore operation.
- *`keystore_password`* is the password that was created for this keystore.
- *`backup_identifier`* defines the tag values. Enclose this setting in single quotation marks (`' '`) and separate each value with a colon.

For example, to create a tag that uses two values, one to capture a specific session ID and the second to capture a specific terminal ID:

```
ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY
USING TAG 'sessionid=3205062574:terminal=xcvt'
IDENTIFIED BY keystore_password
WITH BACKUP;

keystore altered.
```

Both the session ID (`3205062574`) and terminal ID (`xcvt`) can derive their values by using either the `SYS_CONTEXT` function with the `USERENV` namespace, or by using the `USERENV` function.

Related Topics

- About Creating Custom Attribute Tags
  Attribute tags enable you to monitor specific activities users perform, such as accessing a particular terminal ID.

Exporting and Importing the TDE Master Encryption Key in Isolated Mode

You can export and import the TDE master encryption key in different ways in isolated mode.

- Exporting a TDE Master Encryption Key in Isolated Mode
  In isolated mode, you can use the `ADMINISTER KEY MANAGEMENT` statement to export a TDE master encryption key.
- Importing a TDE Master Encryption Key in Isolated Mode
  The `ADMINISTER KEY MANAGEMENT` statement with the `IMPORT [ENCRYPTION] KEYS WITH SECRET` clause can import a TDE master encryption key.

Exporting a TDE Master Encryption Key in Isolated Mode

In isolated mode, you can use the `ADMINISTER KEY MANAGEMENT` statement to export a TDE master encryption key.

1. Log in to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Export the TDE master encryption keystore by using the following syntax:

```
ADMINISTER KEY MANAGEMENT EXPORT [ENCRYPTION] KEYS
WITH SECRET "export_secret"
TO 'file_path'
[FORCE KEYSTORE]
IDENTIFIED BY [EXTERNAL STORE | keystore_password]
[WITH IDENTIFIER IN 'key_id1', 'key_id2', 'key_idn' | (SQL_query)];
```

In this specification:

- *export_secret* is a password that you can specify to encrypt the export the file that contains the exported keys. Enclose this secret in double quotation marks (" "), or you can omit the quotation marks if the secret has no spaces.
- *file_path* is the complete path and name of the file to which the keys must be exported. Enclose this path in single quotation marks (' '). You can export to regular file systems only.
- FORCE KEYSTORE temporarily opens the password-protected keystore for this operation. You must open the keystore for this operation.
- *key_id1*, *key_id2*, *key_idn* is a string of one or more TDE master encryption key identifiers for the TDE master encryption key being exported. Separate each key identifier with a comma and enclose each of these key identifiers in single quotation marks (' '). To find TDE master encryption key identifiers, query the KEY_ID column of the V$ENCRYPTION_KEYS dynamic view.
- *SQL_query* is a query that fetches a list of the TDE master encryption key identifiers. It should return only one column that contains the TDE master encryption key identifiers. This query is executed with current user rights.

Related Topics

- Exporting and Importing the TDE Master Encryption Key
  You can export and import the TDE master encryption key in different ways.

Importing a TDE Master Encryption Key in Isolated Mode

The ADMINISTER KEY MANAGEMENT statement with the IMPORT [ENCRYPTION] KEYS WITH SECRET clause can import a TDE master encryption key.

1. Log in to the isolated mode PDB as a user who has been granted the ADMINISTER KEY MANAGEMENT or SYSKM privilege.

   Contact your SYSDBA administrator for the correct PDB. To check the current container, run the SHOW CON_NAME command.

2. Run the following SQL statement:

```
ADMINISTER KEY MANAGEMENT IMPORT [ENCRYPTION] KEYS
WITH SECRET "import_secret"
FROM 'file_name'
[FORCE KEYSTORE]
IDENTIFIED BY [EXTERNAL STORE | keystore_password]
[WITH BACKUP [USING 'backup_identifier']];
```

   In this specification:

   - *import_secret* is the same password that was used to encrypt the keys during the export operation. Enclose this secret in double quotation marks (" "), or you can omit the quotation marks if the secret has no spaces.
   - *file_name* is the complete path and name of the file from which the keys need to be imported. Enclose this setting in single quotation marks (' ').
   - FORCE KEYSTORE temporarily opens the password-protected keystore for this operation. You must open the keystore for this operation.

Related Topics

- Exporting and Importing the TDE Master Encryption Key
  You can export and import the TDE master encryption key in different ways.

Storing Oracle Database Secrets in Isolated Mode

Secrets are data that support internal Oracle Database features that integrate external clients such as Oracle GoldenGate into the database.

- Storing Oracle Database Secrets in a Software Keystore in Isolated Mode
  The `ADMINISTER KEY MANAGEMENT ADD SECRET|UPDATE SECRET|DELETE SECRET` statements can add secrets, update secrets, and delete secrets in a keystore.
- Storing Oracle Database Secrets in a Hardware Keystore in Isolated Mode
  The `ADMINISTER KEY MANAGEMENT ADD SECRET|UPDATE SECRET|DELETE SECRET` statements can add secrets, update secrets, and delete secrets in a keystore.

Storing Oracle Database Secrets in a Software Keystore in Isolated Mode

The `ADMINISTER KEY MANAGEMENT ADD SECRET|UPDATE SECRET|DELETE SECRET` statements can add secrets, update secrets, and delete secrets in a keystore.

1. Log in to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Add, update, or delete a database secret in a software keystore by using the following syntax:
   - To add a secret:

     ```
     ADMINISTER KEY MANAGEMENT
     ADD SECRET 'secret' FOR CLIENT 'client_identifier'
     [USING TAG 'tag']
     [TO [[LOCAL] AUTO_LOGIN] KEYSTORE keystore_location
     [WITH BACKUP [USING backup_id]]
     | [FORCE KEYSTORE]
     IDENTIFIED BY [EXTERNAL STORE | keystore_password]
     [WITH BACKUP [USING backup_id]]];
     ```

   - To update a secret:

     ```
     ADMINISTER KEY MANAGEMENT
     UPDATE SECRET 'secret' FOR CLIENT 'client_identifier'
     [USING TAG 'tag']
     [TO [[LOCAL] AUTO_LOGIN] KEYSTORE keystore_location
     [WITH BACKUP [USING backup_id]]
     | [FORCE KEYSTORE]
     IDENTIFIED BY [EXTERNAL STORE | keystore_password]
     [WITH BACKUP [USING backup_id]]];
     ```

   - To delete a secret:

     ```
     ADMINISTER KEY MANAGEMENT
     DELETE SECRET FOR CLIENT 'client_identifier'
     [FROM [[LOCAL] AUTO_LOGIN] KEYSTORE keystore_location
     [FORCE KEYSTORE]
     IDENTIFIED BY [EXTERNAL STORE | keystore_password]
     [WITH BACKUP [USING backup_id]]];
     ```

   The specification is as follows:

- *secret* is the client secret key to be stored, updated, or deleted. To find information about existing secrets and their client identifiers, query the V$CLIENT_SECRETS dynamic view.
- *client_identifier* is an alphanumeric string used to identify the secret key.
- FORCE KEYSTORE temporarily opens the password-protected keystore for this operation. You must open the keystore for this operation.

Related Topics

- Storing Oracle Database Secrets
  Secrets are data that support internal Oracle Database features that integrate external clients such as Oracle GoldenGate into the database.

Storing Oracle Database Secrets in a Hardware Keystore in Isolated Mode

The ADMINISTER KEY MANAGEMENT ADD SECRET|UPDATE SECRET|DELETE SECRET statements can add secrets, update secrets, and delete secrets in a keystore.

1. Log in to the isolated mode PDB as a user who has been granted the ADMINISTER KEY MANAGEMENT or SYSKM privilege.
2. Add, update, or delete a database secret in a hardware keystore by using the following syntax:
   - To add a secret:

     ```
     ADMINISTER KEY MANAGEMENT
     ADD SECRET 'secret' FOR CLIENT 'client_identifier'
     [USING TAG 'tag']
     [TO [[LOCAL] AUTO_LOGIN] KEYSTORE keystore_location
     [FORCE KEYSTORE]
     IDENTIFIED BY "hardware_keystore_credentials"]
     [WITH BACKUP [USING backup_id]];
     ```

   - To update a secret:

     ```
     ADMINISTER KEY MANAGEMENT
     UPDATE SECRET 'secret' FOR CLIENT 'client_identifier'
     [USING TAG 'tag']
      [TO [[LOCAL] AUTO_LOGIN] KEYSTORE keystore_location
     [FORCE KEYSTORE]
     IDENTIFIED BY "hardware_keystore_credentials"
     [WITH BACKUP [USING backup_id]];
     ```

   - To delete a secret:

     ```
     ADMINISTER KEY MANAGEMENT
     DELETE SECRET FOR CLIENT 'client_identifier'
     [FROM [[LOCAL] AUTO_LOGIN] KEYSTORE keystore_location
     [FORCE KEYSTORE]
     IDENTIFIED BY "hardware_keystore_credentials";
     ```

     The specification is as follows:

     - *secret* is the client secret key to be stored, updated, or deleted. To find information about existing secrets and their client identifiers, query the V$CLIENT_SECRETS dynamic view.
     - *client_identifier* is an alphanumeric string used to identify the secret key.
     - FORCE KEYSTORE temporarily opens the password-protected keystore for this operation. You must open the keystore for this operation.

- *hardware_keystore_credentials* refers to the credentials for either an HSM or an Oracle Key Vault hardware keystore. For an HSM, specify the credentials using this format, enclosed in quotation marks and separating the components with a colon: *"user_name:password"*, with *user_name* being the user who created the HSM and password being this user's password. For Oracle Key Vault, enter only the password of the user who created the keystore. Enclose this password with quotation marks.

Related Topics

- Storing Oracle Database Secrets
  Secrets are data that support internal Oracle Database features that integrate external clients such as Oracle GoldenGate into the database.

Migrating Keystores in Isolated Mode

You can perform migration and reverse migration operations between software keystores and hardware keystores in isolated mode.

- Migrating from a Password-Protected Software Keystore to a Hardware Keystore in Isolated Mode
  In isolated mode, you can migrate from a password-protected software keystore to a hardware keystore.
- Migrating from a Hardware Keystore to a Password-Protected Software Keystore in Isolated Mode
  In isolated mode, you can migrate from a hardware keystore to a password-protected software keystore.

Migrating from a Password-Protected Software Keystore to a Hardware Keystore in Isolated Mode

In isolated mode, you can migrate from a password-protected software keystore to a hardware keystore.

For both the software keystore and the hardware keystore to open at the same time, either the software keystore must have the same password as the hardware keystore, or alternatively, after the migration has completed you can create an auto-login keystore for the software keystore.

1. Log in to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Set the password of the hardware keystore so that it matches that of the software keystore.

   Some Oracle tools, such as Oracle Data Pump and Oracle Recovery Manager, require access to the old software keystore to decrypt data that was exported or backed up using the TDE master encryption key from the old software keystore.

   - To set the software keystore password so that it is the same as that of the hardware keystore, use the following syntax:

     ```
     ADMINISTER KEY MANAGEMENT ALTER KEYSTORE PASSWORD
     [FORCE KEYSTORE]
     IDENTIFIED BY software_keystore_password
     SET "hardware_keystore_credentials"
     WITH BACKUP [USING 'backup_identifier'];
     ```

In this specification:

- *software_keystore_password* is the password that was assigned to this keystore when it was created.
- *hardware_keystore_credentials* refers to the credentials for either an HSM or an Oracle Key Vault hardware keystore. For an HSM, specify the credentials using this format, enclosed in quotation marks and separating the components with a colon: "*user_name*:*password*", with *user_name* being the user who created the HSM and password being this user's password. For Oracle Key Vault, enter only the password that was used when the database endpoint was registered with Oracle Key Vault. Enclose this password with quotation marks.

- Alternatively, to create an auto-login keystore for a software keystore, use the following syntax:

```
ADMINISTER KEY MANAGEMENT CREATE [LOCAL] AUTO_LOGIN KEYSTORE
FROM KEYSTORE 'keystore_location'
IDENTIFIED BY software_keystore_password;
```

3. Provision Oracle Key Vault for the isolated mode PDB by following the instructions in *Oracle Key Vault Administrator's Guide* to install the Oracle Key Vault software onto the endpoint.
4. Set the configuration of the keystore so that the hardware keystore becomes the new primary, and the password-protected software keystore becomes the secondary, as follows:

```
ALTER SYSTEM SET TDE_CONFIGURATION="KEYSTORE_CONFIGURATION=OKV|FILE";
```

5. Migrate the hardware keystore by using the following syntax:

```
ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY
IDENTIFIED BY "hardware_keystore_credentials"
[FORCE KEYSTORE]
MIGRATE USING software_keystore_password
[WITH BACKUP [USING 'backup_identifier']];
```

After you complete the migration, you do not need to restart the database, nor do you need to manually re-open the hardware keystore.

Related Topics

- Keystore Order After a Migration
  After you perform a migration, keystores can be either primary or secondary in their order.
- Migration of Keystores to and from Oracle Key Vault
  You can use Oracle Key Vault to migrate both software and hardware keystores to and from Oracle Key Vault.
- *Oracle Key Vault Administrator's Guide*

Migrating from a Hardware Keystore to a Password-Protected Software Keystore in Isolated Mode

In isolated mode, you can migrate from a hardware keystore to a password-protected software keystore.

1. Log in to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Ensure that you have enrolled the PDB endpoint and that the Oracle Key Vault configuration is present at the *$WALLET_ROOT/PDB_GUID*`/okv` location.

   For both the software keystore and the hardware keystore to open at the same time, either the software keystore must have the same password as the hardware keystore, or alternatively, after the reverse migration has completed, you can create an auto-login keystore for the software keystore.
3. Set the `TDE_CONFIGURATION` parameter as follows, so that `FILE` becomes the new primary keystore, and `OKV` becomes the secondary keystore.

   ```
   ALTER SYSTEM SET TDE_CONFIGURATION="KEYSTORE_CONFIGURATION=FILE|OKV";
   ```
4. Now that the keystore configuration has been completed, issue the following statement to reverse migrate from the hardware keystore to the password-protected software keystore:

   ```
   ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY
   IDENTIFIED BY software_keystore_password
   REVERSE MIGRATE USING "hardware_keystore_credentials"
   [WITH BACKUP [USING 'backup_identifier']];
   ```
5. Optionally, change the password of the newly migrated software keystore.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT ALTER KEYSTORE PASSWORD
   IDENTIFIED BY
   old_password SET new_password
   WITH BACKUP USING 'pwd_change';
   ```

After you complete these steps, the migration process automatically reloads the keystore keys in memory. You do not need to restart the database, nor do you need to manually re-open the software keystore. The hardware keystore may still be required after reverse migration because the old keys are likely to have been used for encrypted backups or by tools such as Oracle Data Pump and Oracle Recovery Manager. You should create an auto-login keystore and put the `HSM_PASSWORD` client secret into it. For example:

```
ADMINISTER KEY MANAGEMENT ADD SECRET 'user_name:password'
FOR CLIENT 'HSM_PASSWORD'
TO LOCAL AUTO_LOGIN KEYSTORE software_keystore_location
WITH BACKUP;
```

Related Topics

- About Migrating Back from a Hardware Keystore
  To switch from using a hardware keystore solution to a software keystore, you can use reverse migration of the keystore.
- Keystore Order After a Migration
  After you perform a migration, keystores can be either primary or secondary in their order.
- Migration of Keystores to and from Oracle Key Vault
  You can use Oracle Key Vault to migrate both software and hardware keystores to and from Oracle Key Vault.

Automatically Removing Inactive TDE Master Encryption Keys in Isolated Mode

In isolated mode, the `REMOVE_INACTIVE_STANDBY_TDE_MASTER_KEY` initialization parameter can configure the automatic removal of inactive master encryption keys.

1. Log in to the server where the isolated mode PDB standby database resides.
2. Locate the `init.ora` file for the database.

   By default, `init.ora` file is located in the `$ORACLE_HOME/dbs` directory.
3. Edit the `init.ora` file to include the `REMOVE_INACTIVE_STANDBY_TDE_MASTER_KEY` initialization parameter.

   For example:

   ```
   remove_inactive_standby_tde_master_key = true
   ```

   Setting this parameter to `TRUE` enables the automatic removal of inactive master encryption keys; setting it to `FALSE` disables the automatic removal.

Uniting a Pluggable Database Keystore

Uniting a PDB keystore moves the TDE master encryption keys from the PDB keystore into the keystore of the CDB root. This enables the administrator of the keystore of the CDB root to manage the keys.

The client secrets are not moved. Instead, they are left behind in the keystore that the PDB used while it was configured in isolated mode. Oracle recommends that you delete client secrets from that keystore before you unite the PDB keystore. Similarly, when a PDB becomes isolated, no client secret contained in the keystore of the CDB root is moved.

1. Log in to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Unite the PDB keystore, which moves the TDE master encryption keys from the PDB keystore into the keystore of the CDB root, by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT UNITE KEYSTORE
   IDENTIFIED BY isolated_keystore_password
   WITH ROOT KEYSTORE [FORCE KEYSTORE]
   IDENTIFIED BY
   [EXTERNAL STORE | keystore_password_of_cdb_root]
   [WITH BACKUP [USING backup_id]];
   ```

   In this specification:

   - `FORCE KEYSTORE` temporarily opens the password-protected keystore for this operation if an auto-login keystore is open (and in use) or if the keystore is closed.
   - *`united_keystore_password`*: Knowledge of this password does not enable the user who performs the `UNITE KEYSTORE` operation privileges to perform `ADMINISTER KEY MANAGEMENT UNITE KEYSTORE` operations on the PDB.

When the keystore of a PDB is united with a keystore in the CDB root, all of the previously active (historical) TDE master encryption keys that were associated with the PDB are moved to the keystore of the CDB root.

3. Confirm that the isolated mode PDB is now a united mode PDB.

```
SELECT KEYSTORE_MODE FROM V$ENCRYPTION_WALLET;
```

The output should be `UNITED`.

The keystore no longer exists but its master encryption key is now in the keystore in the CDB root. If you later decide that you want the united mode PDB to be an isolated mode PDB again, then you can use the `ADMINISTER KEY MANAGEMENT ISOLATE KEYSTORE` statement.

Related Topics

- Isolating a Pluggable Database Keystore
  Isolating a PDB keystore moves the master encryption key from the CDB root keystore into an isolated mode keystore in the a PDB.

Creating a Keystore When the PDB Is Closed

When you create a keystore in a PDB that is closed, the new keystore is empty and the PDB is converted to isolated mode.

- About Creating a Keystore When the PDB Is Closed
  Creating a keystore in a PDB that is closed could inadvertently cause problems in rekey operations, but the keystore creation can be reverted.
- Reverting a Keystore Creation Operation When a PDB Is Closed
  If you have inadvertently created a keystore in a PDB (and thereby caused it to become configured in isolated mode), then you should reverse the keystore creation operation.

About Creating a Keystore When the PDB Is Closed

Creating a keystore in a PDB that is closed could inadvertently cause problems in rekey operations, but the keystore creation can be reverted.

In previous releases, if you tried to create a keystore in a closed PDB, you were prevented and an `ORA-65040: operation not allowed from within a pluggable database` error would appear. Starting in Oracle Database release 18c, for convenience, when the keystore of the PDB is closed and if you run the `ADMINISTER KEY MANAGEMENT CREATE KEYSTORE` statement in the PDB, Oracle Database allows the operation.

If the closed PDB has not been configured to use encryption (that is, it has never had an `ADMINISTER KEY MANAGEMENT SET KEY` statement performed in it), after you execute `ADMINISTER KEY MANAGEMENT CREATE KEYSTORE`, resulting in an empty keystore and the configuration of the PDB being changed to isolated mode, then you can create a TDE master encryption key in this empty keystore.

If, however, the PDB was already configured to use encryption, then the PDB may be configured in united mode (and thus have its TDE master encryption key being managed in the keystore of the CDB root).

Mistakenly running an `ADMINISTER KEY MANAGEMENT CREATE KEYSTORE` statement on such a closed PDB will create an additional keystore (which will be empty), and will

then configure the PDB to be in isolated mode. This effectively misconfigures the PDB, because the PDB is now in isolated mode (whereas it should be in united mode), yet its TDE master encryption key is still in the keystore of the CDB root. This misconfiguration can cause problems later on, if you try to rekey the TDE master encryption key by using the `ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY` statement . An `ORA-28362: master key not found` error will appear, because when encryption has already been enabled and a key has been set, Oracle Database treats the `ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY` statement as a rekey operation. In order to perform a rekey operation, Oracle Database must locate the currently active TDE master encryption key of the PDB. But in this misconfigured PDB, Oracle Database cannot locate the TDE master encryption key, because the PDB is now in isolated mode and the necessary key is in the keystore of the CDB root. Hence, the PDB is no longer configured to search in the keystore of the PDB, and the rekey operation fails.

To remedy the misconfiguration of the PDB, you must reconfigure the PDB to united mode and you must remove the empty keystore. (Always make a backup before removing any keystore.) When the PDB is configured back to united mode, then the currently active TDE master encryption key is once again available for rekey and other TDE master encryption key operations.

If later on you want to configure the PDB to be in isolated mode, then you can open the PDB and run the `ADMINISTER KEY MANAGEMENT ISOLATE KEYSTORE` statement, which isolates the PDB and moves its TDE master encryption key and previously-active (historical) keys from the keystore of the CDB root to a newly-created keystore for the isolated PDB.

Related Topics

- Reverting a Keystore Creation Operation When a PDB Is Closed
  If you have inadvertently created a keystore in a PDB (and thereby caused it to become configured in isolated mode), then you should reverse the keystore creation operation.

Reverting a Keystore Creation Operation When a PDB Is Closed

If you have inadvertently created a keystore in a PDB (and thereby caused it to become configured in isolated mode), then you should reverse the keystore creation operation.

Use this procedure if you created a keystore in a closed PDB that already had encryption enabled (that is, it already had a TDE master encryption key).

1. Log in to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Confirm the mode of the PDB by querying the `KEYSTORE_MODE` column of the `V$ENCRYPTION_WALLET` dynamic view.
3. If the `V$ENCRYPTION_WALLET` output is `ISOLATED`, then execute the `ALTER SYSTEM` statement to reconfigure the PDB to united mode.
   - When pfile is in use, clear the `TDE_CONFIGURATION` parameter by using the following statement:

     ```
     ALTER SYSTEM RESET TDE_CONFIGURATION SCOPE=MEMORY;
     ```

     In an Oracle Real Application Clusters environment, include the `SID` parameter:

```
ALTER SYSTEM RESET TDE_CONFIGURATION SCOPE=MEMORY SID='*';
```

- When spfile is in use, clear the `TDE_CONFIGURATION` parameter by using this statement:

  ```
  ALTER SYSTEM RESET TDE_CONFIGURATION SCOPE=BOTH;
  ```

  In an Oracle Real Application Clusters environment, include the `SID` parameter:

  ```
  ALTER SYSTEM RESET TDE_CONFIGURATION SCOPE=BOTH SID='*';
  ```

4. In the *`WALLET_ROOT/PDB_GUID`*`/tde` directory, find and back up the `ewallet.p12` keystore file that was mistakenly created.
5. Delete the mistakenly-created empty keystore file.

At this stage, the PDB will be in united mode and the correct keystore and TDE master encryption key will be available for any future rekey operations.

Administering Transparent Data Encryption in Isolated Mode

You can perform a number of general administrative tasks with Transparent Data Encryption in isolated mode.

- Moving PDBs from One CDB to Another in Isolated Mode
  In isolated mode, you can automatically move a PDB from one CDB to another (for example, for load balancing or adding new functionality).
- Unplugging and Plugging a PDB with Encrypted Data in a CDB in Isolated Mode
  In isolated mode, for a PDB that has encrypted data, you can plug it into a CDB. Conversely, you can unplug this PDB from the CDB.
- Cloning a PDB with Encrypted Data in a CDB in Isolated Mode
  The `CREATE PLUGGABLE DATABASE` statement with the `KEYSTORE IDENTIFIED BY` clause can clone a PDB that has encrypted data.
- Performing a Remote Clone of PDB with Encrypted Data Between Two CDBs in Isolated Mode
  The `CREATE PLUGGABLE DATABASE` statement with the `KEYSTORE IDENTIFIED BY` clause can remotely clone a PDB that has encrypted data.
- Relocating Across CDBs a Cloned PDB with Encrypted Data in Isolated Mode
  The `CREATE PLUGGABLE DATABASE` statement with the `KEYSTORE IDENTIFIED BY` clause can relocate across CDBs a cloned PDB that has encrypted data.
- How Keystore Open and Close Operations Work in Isolated Mode
  You should be aware of how keystore open and close operations work in isolated mode.
- Exporting and Importing Master Encryption Keys for a PDB in Isolated Mode
  In isolated mode, the `EXPORT` and `IMPORT` clauses of `ADMINISTER KEY MANAGEMENT EXPORT` can export or import master encryption keys for a PDB.

Moving PDBs from One CDB to Another in Isolated Mode

In isolated mode, you can automatically move a PDB from one CDB to another (for example, for load balancing or adding new functionality).

If the PDB has TDE-encrypted tables or tablespaces, then you can set the `ONE_STEP_PLUGIN_FOR_PDB_WITH_TDE` dynamic system parameter to `TRUE` to enable the PDB to include the TDE keys in the PDB move operation. This parameter avoids you having to manually provide a keystore password when you import the TDE keys into the PDB after it has moved to a different CDB. When `ONE_STEP_PLUGIN_FOR_PDB_WITH_TDE` is set to `TRUE`, the database caches the keystore password in memory, obfuscated at the system level, and then uses it for the import operation. The default for `ONE_STEP_PLUGIN_FOR_PDB_WITH_TDE` is `FALSE`.

1. Before you begin the PDB move operation, log in to the CDB root as a user who has been granted the `SYSDBA` administrative privilege.

   For example:

   ```
   sqlplus c##sec_admin as sysdba
   Enter password: password
   ```

2. Set the `ONE_STEP_PLUGIN_FOR_PDB_WITH_TDE` dynamic initialization parameter `TRUE`.

   For example:

   ```
   ALTER SYSTEM SET ONE_STEP_PLUGIN_FOR_PDB_WITH_TDE = TRUE;
   ```

3. Move (relocate) the PDB.

Related Topics

- *Oracle Multitenant Administrator's Guide*

Unplugging and Plugging a PDB with Encrypted Data in a CDB in Isolated Mode

In isolated mode, for a PDB that has encrypted data, you can plug it into a CDB. Conversely, you can unplug this PDB from the CDB.

- Unplugging a PDB That Has Encrypted Data in Isolated Mode
  You can unplug a PDB (that has encrypted data) from one CDB and then optionally plug it into another CDB.
- Plugging a PDB That Has Encrypted Data into a CDB in Isolated Mode
  After you plug a PDB that has encrypted data into a CDB, you can set the encryption key in the PDB.
- Unplugging a PDB That Has Master Encryption Keys Stored in a Hardware Keystore in Isolated Mode
  You can unplug a PDB from one CDB that has been configured with a hardware keystore and then plug it into another CDB also configured with a hardware keystore.
- Plugging a PDB That Has Master Keys Stored in an HSM in Isolated Mode
  The `ADMINISTER KEY MANAGEMENT` statement can import a hardware keystore master encryption key to a PDB that has been moved to another CDB.

Unplugging a PDB That Has Encrypted Data in Isolated Mode

You can unplug a PDB (that has encrypted data) from one CDB and then optionally plug it into another CDB.

Unlike united mode, you do not need to specify the `ENCRYPT` clause in the `ALTER PLUGGABLE DATABASE` statement. The database that is unplugged contains data files and other associated files. Because each PDB can have its own unique keystore, you do not need to export the TDE master encryption key of the PDB that you want to unplug. You can check if a PDB has already been unplugged by querying the `STATUS` column of the `DBA_PDBS` data dictionary view.

- Unplug the isolated mode PDB as you normally unplug PDBs.

  For example:

```
ALTER PLUGGABLE DATABASE pdb1
  UNPLUG INTO '/oracle/data/pdb1.xml';
```

Related Topics

- *Oracle Multitenant Administrator's Guide*
- *Oracle Database SQL Language Reference*

Plugging a PDB That Has Encrypted Data into a CDB in Isolated Mode

After you plug a PDB that has encrypted data into a CDB, you can set the encryption key in the PDB.

Unlike united mode, you do not need to specify the `DECRYPT` clause in the `CREATE PLUGGABLE DATABASE` statement. When you plug an unplugged PDB into another CDB, the key version is set to `0` because this operation invalidates the history of the previous keys. You can check the key version by querying the `KEY_VERSION` column of the `V$ENCRYPTED_TABLESPACES` dynamic view. Similarly, if a control file is lost and recreated, then the previous history of the keys is reset to `0`. You can check if a PDB has already been plugged in by querying the `STATUS` column of the `DBA_PDBS` data dictionary view.

1. Create the PDB by plugging the unplugged PDB into the CDB.
   - For example, if you had exported the PDB data into a metadata XML file:

```
CREATE PLUGGABLE DATABASE CDB1_PDB2
USING '/tmp/cdb1_pdb2.xml'
NOCOPY KEYSTORE
IDENTIFIED BY password;
```

   - If you had exported the PDB into an archive file:

```
CREATE PLUGGABLE DATABASE CDB1_PDB2
USING '/tmp/cdb1_pdb2.pdb';
```

   During the open operation of the PDB after the plug operation, Oracle Database determines if the PDB has encrypted data. If so, it opens the PDB in the `RESTRICTED` mode.

   You can find if the source database has encrypted data or a TDE master encryption key set in the keystore by querying the `V$ENCRYPTION_KEYS` dynamic view.

2. Open the PDB.

   For example:

   ```
   ALTER PLUGGABLE DATABASE CDB1_PDB2 OPEN;
   ```

3. Open the keystore in the CDB root.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
   IDENTIFIED BY password;
   ```

   Optionally, open the keystore in the PDB.

4. In the PDB, open the keystore and set the TDE master encryption key for the PDB.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT SET KEY
   IDENTIFIED BY keystore_password
   WITH BACKUP USING 'emp_key_backup';
   ```

Unplugging a PDB That Has Master Encryption Keys Stored in a Hardware Keystore in Isolated Mode

You can unplug a PDB from one CDB that has been configured with a hardware keystore and then plug it into another CDB also configured with a hardware keystore.

1. Unplug the PDB.

   You can check if a PDB has already been unplugged by querying the `STATUS` column of the `DBA_PDBS` data dictionary view.

2. Move the master encryption keys of the unplugged PDB from the hardware that was used at the source CDB to the hardware that is in use at the destination CDB.

   Refer to the documentation for the hardware keystore for information about moving master keys between hardware keystores.

Related Topics

- *Oracle Multitenant Administrator's Guide*

Plugging a PDB That Has Master Keys Stored in an HSM in Isolated Mode

The `ADMINISTER KEY MANAGEMENT` statement can import a hardware keystore master encryption key to a PDB that has been moved to another CDB.

1. Plug the unplugged isolated mode PDB into the destination CDB that has been configured with the hardware keystore.

   You can check if a PDB has already been plugged in by querying the `STATUS` column of the `DBA_PDBS` data dictionary view.

   After the plug-in operation, the PDB that has been plugged in will be in restricted mode.

2. Ensure that the master keys from the HSM that has been configured with the source CDB are available in the hardware keystore of the destination CDB.

3. Log in to the plugged PDB as a user who was granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
4. Open the keystore of the plugged PDB.

   For example, for a PDB called `PDB1`:

   ```
   ALTER SESSION SET CONTAINER = PDB1;
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
   IDENTIFIED BY "harware_keystore_credentials";
   ```
5. Import the hardware keystore master encryption key into the PDB.

   ```
   ADMINISTER KEY MANAGEMENT IMPORT ENCRYPTION KEYS
   WITH SECRET "hardware_keystore" FROM 'hardware_keystore'
   IDENTIFIED BY "hardware_keystore_credentials";
   ```
6. Close and re-open the PDB.

   ```
   ALTER PLUGGABLE DATABASE PDB1 CLOSE;
   ALTER PLUGGABLE DATABASE PDB1 OPEN;
   ```

Related Topics

- *Oracle Multitenant Administrator's Guide*

Cloning a PDB with Encrypted Data in a CDB in Isolated Mode

The `CREATE PLUGGABLE DATABASE` statement with the `KEYSTORE IDENTIFIED BY` clause can clone a PDB that has encrypted data.

1. Log in to the isolated mode PDB as a user who was granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Ensure that the software keystore of the PDB that you plan to clone is open.

   You can query the `STATUS` column of the `V$ENCRYPTION_WALLET` view to find if the software keystore is open.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN FORCE KEYSTORE IDENTIFIED BY
   keystore_password;
   ```
3. Clone the PDB.

   For example:

   ```
   CREATE PLUGGABLE DATABASE cdb1_pdb3 FROM cdb1_pdb1
   FILE_NAME_CONVERT=('cdb1_pdb1', 'pdb3/cdb1_pdb3') KEYSTORE
   IDENTIFIED BY keystore_password;
   ```

   Replace *`keystore_password`* with the password of the keystore of the CDB where the `cdb1_pdb3` clone is created.

   After you create the cloned PDB, encrypted data is still accessible by the clone using the master encryption key of the original PDB. After a PDB is cloned, there may be user data in the encrypted tablespaces. This encrypted data is still accessible because the master encryption key of the source PDB is copied over to the destination PDB. Because the clone is a copy of the source PDB but will eventually follow its own course and have its own data and security policies, you should rekey the master encryption key of the cloned PDB.
4. Rekey the master encryption key of the cloned PDB.

For example:

```
ADMINISTER KEY MANAGEMENT SET KEY
IDENTIFIED BY keystore_password
WITH BACKUP USING 'emp_key_backup';
```

Before you rekey the master encryption key of the cloned PDB, the clone can still use master encryption keys that belong to the original PDB. However, these master encryption keys do not appear in the cloned PDB V$ dynamic views. Rekeying the master encryption key ensures that the cloned PDB uses its own unique keys, which will be viewable in the V$ views.

Related Topics

- About Managing Cloned PDBs That Have Encrypted Data in United Mode
  When you clone a PDB, you must make the master encryption key of the source PDB available to cloned PDB.

Performing a Remote Clone of PDB with Encrypted Data Between Two CDBs in Isolated Mode

The CREATE PLUGGABLE DATABASE statement with the KEYSTORE IDENTIFIED BY clause can remotely clone a PDB that has encrypted data.

1. Log in to the isolated mode PDB as a user who was granted the ADMINISTER KEY MANAGEMENT or SYSKM privilege.
2. Query the STATUS column of the V$ENCRYPTION_WALLET view to ensure that the software keystore of the PDB that you plan to clone is open.
3. Create a database link for the PDB that you want to clone remotely.

   Use the CREATE DATABASE LINK SQL statement to create the database link. You must create the database link by following the database link prerequisites that are required for cloning a remote PDB or a non-CDB.
4. Use the CREATE PLUGGABLE DATABASE statement with the KEYSTORE IDENTIFIED BY clause to perform the clone of the PDB.

   For example:

   ```
   CREATE PLUGGABLE DATABASE cdb1_pdb3
   FROM cdb1_pdb1
   FILE_NAME_CONVERT=('cdb1_pdb1', 'pdb3/cdb1_pdb3') KEYSTORE
   IDENTIFIED BY keystore_password;
   ```

   Replace *keystore_password* with the password of the keystore of the CDB where the cdb1_pdb3 clone is created.

   After you create the cloned PDB, encrypted data is still accessible by the clone using the master encryption key of the original PDB. After a PDB is cloned, there may be user data in the encrypted tablespaces. This encrypted data is still accessible because the master encryption key of the source PDB is copied over to the destination PDB. Because the clone is a copy of the source PDB but will eventually follow its own course and have its own data and security policies, you should rekey the master encrytion key of the cloned PDB.
5. Rekey the master encryption key of the remotely cloned PDB.

For example:

```
ADMINISTER KEY MANAGEMENT SET KEY
FORCE KEYSTORE
IDENTIFIED BY keystore_password
WITH BACKUP USING 'emp_key_backup';
```

In this example, `FORCE KEYSTORE` is included because the keystore must be open during the rekey operation.

Before you rekey the master encryption key of the cloned PDB, the clone can still use master encryption keys that belong to the original PDB. However, these master encryption keys do not appear in the cloned PDB `V$` dynamic views. Rekeying the master encryption key ensures that the cloned PDB uses its own unique keys, which will be viewable in the `V$` views.

Related Topics

- About Managing Cloned PDBs That Have Encrypted Data in United Mode
  When you clone a PDB, you must make the master encryption key of the source PDB available to cloned PDB.
- Opening the Software Keystore in a United Mode PDB
  To open a software keystore in united mode, you must use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEYSTORE OPEN` clause.
- *Oracle Multitenant Administrator's Guide*
- *Oracle Multitenant Administrator's Guide*
- *Oracle Database SQL Language Reference*

Relocating Across CDBs a Cloned PDB with Encrypted Data in Isolated Mode

The `CREATE PLUGGABLE DATABASE` statement with the `KEYSTORE IDENTIFIED BY` clause can relocate across CDBs a cloned PDB that has encrypted data.

1. Log in to the isolated mode PDB as a user who was granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Query the `STATUS` column of the `V$ENCRYPTION_WALLET` view to ensure that the the software keystore of the PDB that you plan to clone is open.
3. Create a database link for the PDB that you want to clone remotely.

   Use the `CREATE DATABASE LINK` SQL statement to create the database link. You must create the database link by following the database link prerequisites that are required for cloning a remote PDB or a non-CDB.
4. Use the `CREATE PLUGGABLE DATABASE` statement with the `KEYSTORE IDENTIFIED BY` clause to clone the PDB.

   For example:

   ```
   CREATE PLUGGABLE DATABASE cdb1_pdb3
   FROM cdb1_pdb1
   FILE_NAME_CONVERT=('cdb1_pdb1', 'pdb3/cdb1_pdb3') KEYSTORE
   IDENTIFIED BY keystore_password;
   ```

Replace *keystore_password* with the password of the keystore of the CDB where the cdb1_pdb3 clone is created.

After you create the cloned PDB, encrypted data is still accessible by the clone using the master encryption key of the original PDB. After a PDB is cloned, there may be user data in the encrypted tablespaces. This encrypted data is still accessible because the master encryption key of the source PDB is copied over to the destination PDB. Because the clone is a copy of the source PDB but will eventually follow its own course and have its own data and security policies, you should rekey the master encrytion key of the cloned PDB.

5. Use the ALTER PLUGGABLE DATABASE statement to perform the relocation operation.

   For example:

```
ALTER PLUGGABLE DATABASE cdb1_pdb3
OPEN RELOCATE TO 'need instance name example';
```

6. Rekey the master encryption key of the remotely cloned PDB.

   For example:

```
ADMINISTER KEY MANAGEMENT SET KEY
FORCE KEYSTORE
IDENTIFIED BY keystore_password
WITH BACKUP USING 'emp_key_backup';
```

   In this example, FORCE KEYSTORE is included because the keystore must be open during the rekey operation.

   Before you rekey the master encryption key of the cloned PDB, the clone can still use master encryption keys that belong to the original PDB. However, these master encryption keys do not appear in the cloned PDB V$ dynamic views. Rekeying the master encryption key ensures that the cloned PDB uses its own unique keys, which will be viewable in the V$ views.

Related Topics

- About Managing Cloned PDBs That Have Encrypted Data in United Mode
  When you clone a PDB, you must make the master encryption key of the source PDB available to cloned PDB.
- Opening the Software Keystore in a United Mode PDB
  To open a software keystore in united mode, you must use the ADMINISTER KEY MANAGEMENT statement with the SET KEYSTORE OPEN clause.
- *Oracle Multitenant Administrator's Guide*
- *Oracle Multitenant Administrator's Guide*
- *Oracle Database SQL Language Reference*

How Keystore Open and Close Operations Work in Isolated Mode

You should be aware of how keystore open and close operations work in isolated mode.

For each PDB in isolated mode, you must explicitly open the password-protected software keystore or hardware keystore in the PDB to enable the Transparent Data

Encryption operations to proceed. (Auto-login and local auto-login software keystores open automatically.) Closing a keystore on a PDB blocks all of the Transparent Data Encryption operations on that PDB.

The open and close keystore operations in a PDB depend on the open and close status of the keystore in the PDB.

Note the following:

- You can create a separate keystore password for each PDB in the multitenant environment.
- Before you can manually open a software password-protected or hardware keystore in an individual PDB, you must open the keystore in the CDB root.
- If an auto-login keystore is in use, or if the keystore is closed, then include the `FORCE KEYSTORE` clause in the `ADMINISTER KEY MANAGEMENT` statement when you open or close the keystore.
- If the keystore is a password-protected software keystore that uses an external store for passwords, then set the `IDENTIFIED BY` clause to `EXTERNAL STORE`.
- Before you can set a TDE master encryption key in an individual PDB, you must set the key in the CDB root.
- Auto-login and local auto-login software keystores open automatically. You do not need to manually open these from the root first, or from the PDB.
- If there is any PDB configured in isolated mode that has its keystore open, then an attempt to close the keystore in the CDB root would fail with an `ORA-46692 cannot close wallet` error. Use the `FORCE CLOSE` clause in the `ADMINISTER KEY MANAGEMENT` statement to override this behavior.
- If you perform an `ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN` statement in the CDB root and set the `CONTAINER` clause to `ALL`, then the keystore will only be opened in each PDB that is configured in united mode, and the keystore of any PDB that is configured in isolated mode is not opened.

Exporting and Importing Master Encryption Keys for a PDB in Isolated Mode

In isolated mode, the `EXPORT` and `IMPORT` clauses of `ADMINISTER KEY MANAGEMENT EXPORT` can export or import master encryption keys for a PDB.

- About Exporting and Importing Master Encryption Keys for a PDB in Isolated Mode
  In isolated mode, you can export and import master encryption keys from the CDB root in the same way that you export and import this key for a non-CDB database.
- Exporting or Importing a Master Encryption Key for a PDB in Isolated Mode
  In isolated mode, the `ADMINISTER KEY MANAGEMENT` statement can export or import a master encryption key for a PDB.
- Example: Exporting a Master Encryption Key from a PDB in Isolated Mode
  The `ADMINISTER KEY MANAGEMENT EXPORT ENCRYPTION KEYS` statement can export master encryption keys for a PDB.
- Example: Importing a Master Encryption Key into a PDB in Isolated Mode
  The `ADMINISTER KEY MANAGEMENT IMPORT ENCRYPTION KEYS` statement can import a master encryption key into a PDB.

About Exporting and Importing Master Encryption Keys for a PDB in Isolated Mode

In isolated mode, you can export and import master encryption keys from the CDB root in the same way that you export and import this key for a non-CDB database.

You can export and import all of the master encryption keys that belong to the PDB by exporting and importing the master encryption keys from within a PDB. Export and import operations of master encryption keys in a PDB supports the PDB unplug and plug operations. During a PDB unplug and plug operations, all the master encryption keys that belong to a PDB, as well as the metadata, are involved. Therefore, the `WITH IDENTIFIER` clause of the `ADMINISTER KEY MANAGEMENT EXPORT` statement is not allowed when you export keys from within a PDB. The `WITH IDENTIFIER` clause is only permitted in the CDB root.

You should include the `FORCE KEYSTORE` clause if the CDB root has an auto-login keystore or if the keystore is closed. If the keystore has been configured to use an external store for the password, then use the `IDENTIFIED BY EXTERNAL STORE` clause. For example, to perform an export operation for this scenario:

```
ADMINISTER KEY MANAGEMENT EXPORT KEYS
WITH SECRET "my_secret"
TO '/etc/TDE/export.exp'
FORCE KEYSTORE
IDENTIFIED BY EXTERNAL STORE;
```

This `ADMINISTER KEY MANAGEMENT EXPORT` operation exports not only the keys but creates metadata that is necessary for PDB environments (as well as for cloning operations).

Inside a PDB, the export operation of master encryption keys exports the keys that were created or activated by a PDB with the same GUID as the PDB where the keys are being exported. Essentially, all of the keys that belong to a PDB where the export is being performed will be exported.

The importing of master encryption keys from an export file within a PDB takes place only if the master encryption key was exported from another PDB with the same GUID. To support the plug-in of a PDB into a CDB, the import will also import the master encryption keys from an export file that contains the master encryption keys of a non-CDB exported without the `WITH IDENTIFIER` clause. Because the PDB-specific details, such as the PDB name and database ID, can change from one CDB to the next, the PDB-specific information is modified during the import to reflect the updated PDB information.

> **Note:**
>
> Within a PDB, you can only export the keys of a PDB as a whole. The ability to export them selectively based on a query or an identifier is restricted to the root.

Exporting or Importing a Master Encryption Key for a PDB in Isolated Mode

In isolated mode, the `ADMINISTER KEY MANAGEMENT` statement can export or import a master encryption key for a PDB.

1. Log in to the isolated mode PDB as a user who was granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Perform the export or import operation.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT EXPORT ENCRYPTION KEYS
   WITH SECRET "hr_secret" TO '/tmp/export.p12'
   FORCE KEYSTORE
   IDENTIFIED BY password;
   ```

   Ensure that you include the `FORCE KEYSTORE` clause because the keystore must be open for this operation.

Related Topics

- Example: Exporting a Master Encryption Key from a PDB in Isolated Mode
  The `ADMINISTER KEY MANAGEMENT EXPORT ENCRYPTION KEYS` statement can export master encryption keys for a PDB.

Example: Exporting a Master Encryption Key from a PDB in Isolated Mode

The `ADMINISTER KEY MANAGEMENT EXPORT ENCRYPTION KEYS` statement can export master encryption keys for a PDB.

Example 6-1 shows how to export a master encryption key from the PDB `hrpdb`. In this example, the `FORCE KEYSTORE` clause is included in case the auto-login keystore is in use, or if the keystore is closed.

Example 6-1 Exporting a Master Encryption Key from a PDB

```
ADMINISTER KEY MANAGEMENT EXPORT ENCRYPTION KEYS
WITH SECRET "my_secret" TO '/tmp/export.p12'
FORCE KEYSTORE
IDENTIFIED BY password;
```

Example: Importing a Master Encryption Key into a PDB in Isolated Mode

The `ADMINISTER KEY MANAGEMENT IMPORT ENCRYPTION KEYS` statement can import a master encryption key into a PDB.

Example 6-2 shows how to import a master encryption key into the PDB `hrpdb`.

Example 6-2 Importing a Master Encryption Key into a PDB

```
ADMINISTER KEY MANAGEMENT IMPORT ENCRYPTION KEYS
WITH SECRET "my_secret"
FROM '/tmp/export.p12'
FORCE KEYSTORE
IDENTIFIED BY password
WITH BACKUP;
```

7

General Considerations of Using Transparent Data Encryption

When you use Transparent Data Encryption, you should consider factors such as security, performance, and storage overheads.

- Compression and Data Deduplication of Encrypted Data
  With tablespace encryption, Oracle Database compresses tables and indexes before encrypting the tablespace.
- Security Considerations for Transparent Data Encryption
  As with all Oracle Database features, you should consider security when you create TDE policies.
- Performance and Storage Overhead of Transparent Data Encryption
  The performance of Transparent Data Encryption can vary.
- Modifying Your Applications for Use with Transparent Data Encryption
  You can modify your applications to use Transparent Data Encryption.
- How ALTER SYSTEM and orapki Map to ADMINISTER KEY MANAGEMENT
  Many of the clauses from the `ALTER SYSTEM` statement correspond to the `ADMINISTER KEY MANAGEMENT` statement.
- Using Transparent Data Encryption with PKI Encryption
  PKI encryption is deprecated, but if you are still using it, then there are several issues you must consider.
- Data Loads from External Files to Tables with Encrypted Columns
  You can use SQL*Loader to perform data loads from files to tables that have encrypted columns.
- Transparent Data Encryption and Database Close Operations
  You should ensure that the software or hardware keystore is open before you close the database.

Compression and Data Deduplication of Encrypted Data

With tablespace encryption, Oracle Database compresses tables and indexes before encrypting the tablespace.

This ensures that you receive the maximum space and performance benefits from compression, while also receiving the security of encryption at rest. In the `CREATE TABLESPACE` SQL statement, include both the `COMPRESS` and `ENCRYPT` clauses.

With column encryption, Oracle Database compresses the data after it encrypts the column. This means that compression will have minimal effectiveness on encrypted columns. There is one notable exception: if the column is a SecureFiles LOB, and the encryption is implemented with SecureFiles LOB Encryption, and the compression (and possibly deduplication) are implemented with SecureFiles LOB Compression & Deduplication, then compression is performed before encryption. Similar to the

`CREATE TABLESPACE` statement for tablespace encryption, include both the `COMPRESS` and `ENCRYPT` clauses.

See Also:

- *Oracle Database Backup and Recovery User's Guide* for more information about the Advanced Compression Option
- *Oracle Database SecureFiles and Large Objects Developer's Guide* for information about SecureFiles LOB storage
- *Oracle Database SecureFiles and Large Objects Developer's Guide* for information about SecureFiles Compression

Security Considerations for Transparent Data Encryption

As with all Oracle Database features, you should consider security when you create TDE policies.

- Transparent Data Encryption General Security Advice
  Security considerations for Transparent Data Encryption (TDE) operate within the broader area of total system security.
- Transparent Data Encryption Column Encryption-Specific Advice
  Additional security considerations apply to normal database and network operations when using TDE.
- Managing Security for Plaintext Fragments
  You should remove old plaintext fragments that can appear over time.

Transparent Data Encryption General Security Advice

Security considerations for Transparent Data Encryption (TDE) operate within the broader area of total system security.

Follow these general guidelines:

- Identify the degrees of sensitivity of data in your database, the protection that they need, and the levels of risk to be addressed. For example, highly sensitive data requiring stronger protection can be encrypted with the AES256 algorithm. A database that is not as sensitive can be protected with no salt or the `nomac` option to enable performance benefits.
- Evaluate the costs and benefits that are acceptable to data and keystore protection. Protection of keys determines the type of keystore to be used: auto-login software keystores, password-based software keystores, or hardware keystores.
- Consider having separate security administrators for TDE and for the database.
- Consider having a separate and exclusive keystore for TDE.
- Implement protected back-up procedures for your encrypted data.

Transparent Data Encryption Column Encryption-Specific Advice

Additional security considerations apply to normal database and network operations when using TDE.

Encrypted column data stays encrypted in the data files, undo logs, redo logs, and the buffer cache of the system global area (SGA). However, data is decrypted during expression evaluation, making it possible for decrypted data to appear in the swap file on the disk. Privileged operating system users can potentially view this data.

Column values encrypted using TDE are stored in the data files in encrypted form. However, these data files may still contain some plaintext fragments, called ghost copies, left over by past data operations on the table. This is similar to finding data on the disk after a file was deleted by the operating system.

Managing Security for Plaintext Fragments

You should remove old plaintext fragments that can appear over time.

Old plaintext fragments may be present for some time until the database overwrites the blocks containing such values. If privileged operating system users bypass the access controls of the database, then they might be able to directly access these values in the data file holding the tablespace.

To minimize this risk:

1. Create a new tablespace in a new data file.

   You can use the `CREATE TABLESPACE` statement to create this tablespace.

2. Move the table containing encrypted columns to the new tablespace. You can use the `ALTER TABLE....MOVE` statement.

   Repeat this step for all of the objects in the original tablespace.

3. Drop the original tablespace.

   You can use the `DROP TABLESPACE` *`tablespace`* `INCLUDING CONTENTS KEEP DATAFILES` statement. Oracle recommends that you securely delete data files using platform-specific utilities.

4. Use platform-specific and file system-specific utilities to securely delete the old data file. Examples of such utilities include `shred` (on Linux) and `sdelete` (on Windows).

Performance and Storage Overhead of Transparent Data Encryption

The performance of Transparent Data Encryption can vary.

- Performance Overhead of Transparent Data Encryption
  Transparent Data Encryption tablespace encryption has small associated performance overhead.
- Storage Overhead of Transparent Data Encryption
  TDE tablespace encryption has no storage overhead, but TDE column encryption has some associated storage overhead.

Performance Overhead of Transparent Data Encryption

Transparent Data Encryption tablespace encryption has small associated performance overhead.

The actual performance impact on applications can vary. TDE column encryption affects performance only when data is retrieved from or inserted into an encrypted column. No reduction in performance occurs for operations involving unencrypted columns, even if these columns are in a table containing encrypted columns. Accessing data in encrypted columns involves small performance overhead, and the exact overhead you observe can vary.

The total performance overhead depends on the number of encrypted columns and their frequency of access. The columns most appropriate for encryption are those containing the most sensitive data.

Enabling encryption on an existing table results in a full table update like any other `ALTER TABLE` operation that modifies table characteristics. Keep in mind the potential performance and redo log impact on the database server before enabling encryption on a large existing table.

A table can temporarily become inaccessible for write operations while encryption is being enabled, TDE table keys are being rekeyed, or the encryption algorithm is being changed. You can use online table redefinition to ensure that the table is available for write operations during such procedures.

If you enable TDE column encryption on a very large table, then you may need to increase the redo log size to accommodate the operation.

Encrypting an indexed column takes more time than encrypting a column without indexes. If you must encrypt a column that has an index built on it, you can try dropping the index, encrypting the column with `NO SALT`, and then re-creating the index.

If you index an encrypted column, then the index is created on the encrypted values. When you query for a value in the encrypted column, Oracle Database transparently encrypts the value used in the SQL query. It then performs an index lookup using the encrypted value.

Note:

If you must perform range scans over indexed, encrypted columns, then use TDE tablespace encryption in place of TDE column encryption.

See Also:

- Creating an Encrypted Column in an External Table
- *Oracle Database Administrator's Guide* for information about redefining tables online

Storage Overhead of Transparent Data Encryption

TDE tablespace encryption has no storage overhead, but TDE column encryption has some associated storage overhead.

Encrypted column data must have more storage space than plaintext data. In addition, TDE pads out encrypted values to multiples of 16 bytes. This means that if a credit card number requires nine bytes for storage, then an encrypted credit card value will require an additional seven bytes.

Each encrypted value is also associated with a 20-byte integrity check. This does not apply if you have encrypted columns using the `NOMAC` parameter. If data was encrypted with salt, then each encrypted value requires an additional 16 bytes of storage.

The maximum storage overhead for each encrypted value is from one to 52 bytes.

Related Topics

- Creating an Encrypted Column in an External Table
  The external table feature enables you to access data in external sources as if the data were in a database table.

Modifying Your Applications for Use with Transparent Data Encryption

You can modify your applications to use Transparent Data Encryption.

1. Configure the software or hardware keystore for TDE, and then set the master encryption key.

   See the following sections for more information:

   - Configuring a Software Keystore
   - Configuring a Hardware Keystore
2. Verify that the master encryption key was created by querying the `KEY_ID` column of the `V$ENCRYPTION_KEYS` view.
3. Identify the sensitive columns (such as those containing credit card data) that require Transparent Data Encryption protection.
4. Decide whether to use TDE column encryption or TDE tablespace encryption.

   See the following sections for more information:

   - How Transparent Data Encryption Column Encryption Works
   - How Transparent Data Encryption Tablespace Encryption Works
5. Open the keystore.

   See the following sections for more information:

   - Step 3: Open the Software Keystore
   - Step 3: Open the Hardware Keystore
6. Encrypt the columns or tablespaces.

   See the following sections for more information:

- Encrypting Columns in Tables
- Encryption Conversions for Tablespaces and Databases

How ALTER SYSTEM and orapki Map to ADMINISTER KEY MANAGEMENT

Many of the clauses from the `ALTER SYSTEM` statement correspond to the `ADMINISTER KEY MANAGEMENT` statement.

Table 7-1 compares the Transparent Data Encryption usage of the `ALTER SYSTEM` statement and the `orapki` utility from previous releases with the `ADMINISTER KEY MANAGEMENT` statement.

Table 7-1 How ALTER SYSTEM and orapki Map to ADMINISTER KEY MANAGEMENT

Behavior	ALTER SYSTEM or orapki	ADMINISTER KEY MANAGEMENT
Creating a keystore	For software keystores (called wallets in previous releases): `ALTER SYSTEM SET ENCRYPTION KEY ["certificate_ID"] IDENTIFIED BY keystore_password;` For hardware keystores, the keystore is available after you configure the hardware security module.	For software keystores: `ADMINISTER KEY MANAGEMENT CREATE KEYSTORE 'keystore_location' IDENTIFIED BY software_keystore_password` For hardware keystores, the keystore is available after you configure the hardware security module.
Creating an auto-login keystore	`orapki wallet create -wallet wallet_location -auto_login [-pwd password]`	For software keystores: `ADMINISTER KEY MANAGEMENT CREATE [LOCAL] AUTO_LOGIN KEYSTORE FROM KEYSTORE 'keystore_location' IDENTIFIED BY software_keystore_password;` This type of keystore applies to software keystores only.
Opening a keystore	`ALTER SYSTEM SET ENCRYPTION WALLET OPEN IDENTIFIED BY password;`	`ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY keystore_password [CONTAINER = ALL | CURRENT];`
Closing a keystore	`ALTER SYSTEM SET ENCRYPTION WALLET CLOSE IDENTIFIED BY password;`	For both software and hardware keystores: `ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE IDENTIFIED BY keystore_password [CONTAINER = ALL | CURRENT];`
Migrating from a hardware keystore to a software keystore	Not available	`ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY IDENTIFIED BY software_keystore_password REVERSE MIGRATE USING "user_name:password" [WITH BACKUP [USING 'backup_identifier']];`

Table 7-1 (Cont.) How ALTER SYSTEM and orapki Map to ADMINISTER KEY MANAGEMENT

Behavior	ALTER SYSTEM or orapki	ADMINISTER KEY MANAGEMENT
Migrating from a software keystore to a hardware keystore	`ALTER SYSTEM SET ENCRYPTION KEY IDENTIFIED BY "user_name:password" MIGRATE USING wallet_password;`	`ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY IDENTIFIED BY "user_name:password" MIGRATE USING software_keystore_password;`
Changing a keystore password	`orapki wallet change_pwd -wallet wallet_location [-oldpwd password ] [-newpwd password]`	For password-based software keystores: `ADMINISTER KEY MANAGEMENT ALTER KEYSTORE PASSWORD IDENTIFIED BY software_keystore_old_password SET software_keystore_new_password [WITH BACKUP [USING 'backup_identifier']];` For hardware keystores, you close the keystore, change it in the hardware security module interface, and then reopen the keystore.
Backing up a password-based software keystore	Not available	`ADMINISTER KEY MANAGEMENT BACKUP KEYSTORE [USING 'backup_identifier'] IDENTIFIED BY software_keystore_password [TO 'keystore_location'];`
Merging two software keystores into a third new keystore	Not available	`ADMINISTER KEY MANAGEMENT MERGE KEYSTORE 'keystore1_location' [IDENTIFIED BY software_keystore1_password] AND KEYSTORE 'keystore2_location' [IDENTIFIED BY software_keystore2_password] INTO NEW KEYSTORE 'keystore3_location' IDENTIFIED BY software_keystore3_password;`
Merging one software keystore into another existing keystore	Not available	`ADMINISTER KEY MANAGEMENT MERGE KEYSTORE 'keystore1_location' [IDENTIFIED BY software_keystore1_password] INTO EXISTNG KEYSTORE 'keystore2_location' IDENTIFIED BY software_keystore2_password [WITH BACKUP [USING 'backup_identifier']];`

Setting or rekeying the master encryption key	For software wallets:
	`ALTER SYSTEM SET ENCRYPTION KEY ["certificate_ID"] IDENTIFIED BY keystore_password;`
	For hardware security modules:
	`ALTER SYSTEM SET ENCRYPTION KEY IDENTIFIED BY "user_name:password"`
	Note: The `ALTER SYSTEM SET ENCRYPTION KEY` statement does not update the `V$ENCRYPTION_KEYS` dynamic view after you rekey the encryption key.
		`ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY [USING TAG 'tag'] IDENTIFIED BY keystore_password WITH BACKUP [USING 'backup_identifier'] [CONTAINER = ALL | CURRENT];`
		After you rekey the encryption key, the `V$ENCRYPTION_KEYS` dynamic view is updated.

Table 7-1 (Cont.) How ALTER SYSTEM and orapki Map to ADMINISTER KEY MANAGEMENT

Behavior	ALTER SYSTEM or orapki	ADMINISTER KEY MANAGEMENT
Creating a master encryption key for later user	Not available	`ADMINISTER KEY MANAGEMENT CREATE KEY [USING TAG 'tag'] IDENTIFIED BY keystore_password [WITH BACKUP [USING 'backup_identifier']] [CONTAINER = (ALL|CURRENT)];`
Activating a master encryption key	Not available	`ADMINISTER KEY MANAGEMENT USE KEY 'key_identifier' [USING TAG 'tag'] IDENTIFIED BY keystore_password [WITH BACKUP [USING 'backup_identifier']];`
Creating custom tags for master encryption keys	Not available	`ADMINISTER KEY MANAGEMENT SET TAG 'tag' FOR 'master_key_identifier' IDENTIFIED BY keystore_password [WITH BACKUP [USING 'backup_identifier']];`
Exporting a master encryption key	Not available	`ADMINISTER KEY MANAGEMENT EXPORT [ENCRYPTION] KEYS WITH SECRET "export_secret" TO 'file_path' IDENTIFIED BY software_keystore_password [WITH IDENTIFIER IN 'key_id1', 'key_id2', 'key_idn' | (SQL_query)]`
Importing a master encryption key	Not available	`ADMINISTER KEY MANAGEMENT IMPORT [ENCRYPTION] KEYS WITH SECRET "import_secret" | FROM 'file_name' IDENTIFIED BY software_keystore_password [WITH BACKUP [USING 'backup_identifier']];`

Storing Oracle Database secrets in a keystore	Not available	For software keystores:
		`ADMINISTER KEY MANAGEMENT ADD SECRET|UPDATE SECRET|DELETE SECRET "secret" FOR CLIENT 'client_identifier' [USING TAG'tag'] IDENTIFIED BY keystore_password [WITH BACKUP [USING 'backup_identifier'];`
		For hardware keystores:
		`ADMINISTER KEY MANAGEMENT ADD SECRET|UPDATE SECRET|DELETE SECRET "secret" FOR CLIENT 'client_identifier' [USING TAG 'tag'] IDENTIFIED BY "user_name:password" [WITH BACKUP [USING 'backup_identifier'];`

Using Transparent Data Encryption with PKI Encryption

PKI encryption is deprecated, but if you are still using it, then there are several issues you must consider.

> **Note:**
>
> The use of PKI encryption with Transparent Data Encryption is deprecated. To configure Transparent Data Encryption, use the `ADMINISTER KEY MANAGEMENT` SQL statement.

- Software Master Encryption Key Use with PKI Key Pairs
  A master encryption key can be an existing key pair from a PKI certificate designated for encryption.
- TDE Tablespace and Hardware Keystores with PKI Encryption
  PKI encryption is a cryptographic system that uses two keys, a public key and a private key, to encrypt data.
- Backup and Recovery of a PKI Key Pair
  For software keystores, Transparent Data Encryption supports the use of PKI asymmetric key pairs as master encryption keys for column encryption.

Software Master Encryption Key Use with PKI Key Pairs

A master encryption key can be an existing key pair from a PKI certificate designated for encryption.

Note the following:

- If you have already deployed PKI in your organization, then you can use PKI services such as key escrow and recovery. However, encryption using current PKI algorithms requires significantly more system resources than symmetric key encryption. Using a PKI key pair as a master encryption key may result in greater performance degradation when accessing encrypted columns in the database.
- For PKI-based keys, certificate revocation lists are not enforced because enforcing certificate revocation may lead to losing access to all of the encrypted information in the database. However, you cannot use the same certificate to create the master encryption key again.

TDE Tablespace and Hardware Keystores with PKI Encryption

PKI encryption is a cryptographic system that uses two keys, a public key and a private key, to encrypt data.

You cannot use PKI-based encryption with TDE tablespace encryption or with hardware keystores.

Backup and Recovery of a PKI Key Pair

For software keystores, Transparent Data Encryption supports the use of PKI asymmetric key pairs as master encryption keys for column encryption.

This enables the database to use existing key backup, escrow, and recovery facilities from leading certificate authority vendors.

In current key escrow or recovery systems, the certificate authority with key recovery capabilities typically stores a version of the private key, or a piece of information that helps recover the private key. If the private key is lost, then you can recover the original key and certificate by contacting the certificate authority and initiating a key recovery process.

Typically, the key recovery process is automated and requires the user to present certain authenticating credentials to the certificate authority. TDE puts no restrictions on the key recovery process other than that the recovered key and its associated certificate be a PKCS#12 file that can be imported into an keystore. This requirement is consistent with the key recovery mechanisms of leading certificate authorities.

After obtaining the PKCS#12 file with the original certificate and private key, you must create an empty keystore in the same location as the previous keystore. You can then import the PKCS#12 file into the new keystore by using the same utility. Choose a strong password to protect the keystore.

After you use the `ADMINISTER KEY MANAGEMENT` statements to create the keystore and import the correct encryption keys, log in to the database and run the following `ALTER SYSTEM` statement at the SQL prompt to complete the recovery process:

```
ALTER SYSTEM SET ENCRYPTION KEY "cert_id" IDENTIFIED BY keystore_password;
```

In this specification:

- *`cert_id`* is the certificate ID of the certificate to be used as the master encryption key.
- *`keystore_password`* is a password that you create.

> **Note:**
>
> You must use the `ALTER SYSTEM` statement to regenerate encryption keys for PKI key pairs only. This restriction does not apply to non-PKI encryption keys.

Data Loads from External Files to Tables with Encrypted Columns

You can use SQL*Loader to perform data loads from files to tables that have encrypted columns.

Be aware that with SQL*Loader, you cannot include the `ENCRYPT` clause in the column definition of an external table of the type `ORACLE_LOADER`, but you can include it in the column definitions of external tables of type `ORACLE_DATAPUMP`.

- External tables of type `ORACLE_LOADER`

  The reason that you cannot include the `ENCRYPT` clause in the column definitions of external tables of the type `ORACLE_LOADER` is because the contents of an external table with the `ORACLE_LOADER` type must come from a user-specified plaintext "backing file," and such plaintext files cannot contain any TDE encrypted data.

  If you use the `ENCRYPT` clause in the definition of an external table of type `ORACLE_LOADER`, then when you query the TDE-encrypted column in this external table, the query fails. This is because TDE expects the external data to have been encrypted, and automatically tries to decrypt it on load. This action fails because the "backing file" only contains plaintext.

- External tables of type `ORACLE_DATAPUMP`

  You can use TDE column encryption with external tables of type `ORACLE_DATAPUMP`. This is because for external tables of `ORACLE_DATAPUMP` type, the "backing file" is always created by Oracle Database(during an unload operation) and thus does have support for being populated with encrypted data.

Transparent Data Encryption and Database Close Operations

You should ensure that the software or hardware keystore is open before you close the database.

The master encryption keys may be required during the database close operation. The database close operation automatically closes the software or hardware keystore.

Related Topics

- Step 3: Open the Software Keystore
  Depending on the type of keystore you create, you must manually open the keystore before you can use it.
- Step 3: Open the Hardware Keystore
  After you have configured the hardware security module, you must open the hardware keystore before it can be used.

8

Using Transparent Data Encryption with Other Oracle Features

You can use Oracle Data Encryption with other Oracle features, such as Oracle Data Guard or Oracle Real Application Clusters.

- How Transparent Data Encryption Works with Export and Import Operations
  Oracle Data Pump can export and import tables that contain encrypted columns, as well as encrypt entire dump sets.
- How Transparent Data Encryption Works with Oracle Data Guard
  An Oracle Data Guard primary database and secondary secondary database can share both a software keystore and a hardware security module.
- How Transparent Data Encryption Works with Oracle Real Application Clusters
  Oracle Real Application Clusters (Oracle RAC) nodes can share both a software keystore and a hardware security module.
- How Transparent Data Encryption Works with SecureFiles
  SecureFiles, which stores LOBS, has three features: compression, deduplication, and encryption.
- How Transparent Data Encryption Works with Oracle Call Interface
  Transparent Data Encryption does not have any effect on the operation of Oracle Call Interface (OCI).
- How Transparent Data Encryption Works with Editions
  Transparent Data Encryption does not have any effect on the Editions feature of Oracle Database.
- Configuring Transparent Data Encryption to Work in a Multidatabase Environment
  Each Oracle database on the same server (such as databases sharing the same Oracle binary but using different data files) must access its own TDE keystore.

How Transparent Data Encryption Works with Export and Import Operations

Oracle Data Pump can export and import tables that contain encrypted columns, as well as encrypt entire dump sets.

- About Exporting and Importing Encrypted Data
  You can use Oracle Data Pump to export and import tables that have encrypted columns.
- Exporting and Importing Tables with Encrypted Columns
  You can export and import tables with encrypted columns using the `ENCRYPTION=ENCRYPTED_COLUMNS_ONLY` setting.
- Using Oracle Data Pump to Encrypt Entire Dump Sets
  Oracle Data Pump can encrypt entire dump sets, not just Transparent Data Encryption columns.

- Using Oracle Data Pump with Encrypted Data Dictionary Data
  Oracle Data Pump operations provide protections for encrypted passwords and other encrypted data.

About Exporting and Importing Encrypted Data

You can use Oracle Data Pump to export and import tables that have encrypted columns.

For both software and hardware keystores, the following points are important when you must export tables containing encrypted columns:

- Sensitive data should remain unintelligible during transport.
- Authorized users should be able to decrypt the data after it is imported at the destination.

When you use Oracle Data Pump to export and import tables containing encrypted columns, it uses the `ENCRYPTION` parameter to enable encryption of data in dump file sets. The `ENCRYPTION` parameter allows the following values:

- `ENCRYPTED_COLUMNS_ONLY`: Writes encrypted columns to the dump file set in encrypted format
- `DATA_ONLY`: Writes all of the data to the dump file set in encrypted format
- `METADATA_ONLY`: Writes all of the metadata to the dump file set in encrypted format
- `ALL`: Writes all of the data and metadata to the dump file set in encrypted format
- `NONE`: Does not use encryption for dump file sets

Exporting and Importing Tables with Encrypted Columns

You can export and import tables with encrypted columns using the `ENCRYPTION=ENCRYPTED_COLUMNS_ONLY` setting.

1. Ensure that the keystore is open before you attempt to export tables containing encrypted columns.

   In a multitenant environment, if you are exporting data in a pluggable database (PDB), then ensure that the wallet is open in the PDB. If you are exporting into the root, then ensure that the wallet is open in the root.

   To find if the keystore is open, query the `STATUS` column of the `V$ENCRYPTION_WALLET` view. If you must open the keystore, then run the following SQL statement:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
   IDENTIFIED BY software_keystore_password
   [CONTAINER = ALL | CURRENT];
   ```

   The *`software_keystore_password`* setting is the password for the keystore. The keystore must be open because the encrypted columns must be decrypted using the TDE table keys, which requires access to the TDE master encryption key. The columns are reencrypted using a password, before they are exported.
2. Run the `EXPDP` command, using the `ENCRYPTION_PASSWORD` parameter to specify a password that is used to encrypt column data in the export dump file set.

The following example exports the `employee_data` table. The `ENCRYPTION_PWD_PROMPT = YES` setting enables you to prompt for the password interactively, which is a recommended security practice.

```
expdp hr TABLES=employee_data DIRECTORY=dpump_dir
DUMPFILE=dpcd2be1.dmp ENCRYPTION=ENCRYPTED_COLUMNS_ONLY
ENCRYPTION_PWD_PROMPT = YES

Password: password_for_hr
```

3. To import the exported data into the target database, ensure that you specify the same password that you used for the export operation, as set by the `ENCRYPTION_PASSWORD` parameter.

   The password is used to decrypt the data. Data is reencrypted with the new TDE table keys generated in the target database. The target database must have the keystore open to access the TDE master encryption key. The following example imports the `employee_data` table:

   ```
   impdp hr TABLES=employee_data DIRECTORY=dpump_dir
   DUMPFILE=dpcd2be1.dmp
   ENCRYPTION_PWD_PROMPT = YES

   Password: password_for_hr
   ```

Using Oracle Data Pump to Encrypt Entire Dump Sets

Oracle Data Pump can encrypt entire dump sets, not just Transparent Data Encryption columns.

While importing, you can use either the password or the keystore TDE master encryption key to decrypt the data. If the password is not supplied, then the TDE master encryption key in the keystore is used to decrypt the data. The keystore must be present and open at the target database. The open keystore is also required to reencrypt column encryption data at the target database.

You can use the `ENCRYPTION_MODE=TRANSPARENT` setting to transparently encrypt the dump file set with the TDE master encryption key stored in the keystore. A password is not required in this case. The keystore must be present and open at the target database, and it must *contain* the TDE master encryption key from the *source* database for a successful decryption of column encryption metadata during an import operation.

The open keystore is also required to reencrypt column encryption metadata at the target database. If a keystore already exists on the target database, then you can export the current TDE master encryption key *from* the keystore of the source database and import it *into* the keystore of the target database.

- Use the `ENCRYPTION_MODE` parameter to specify the encryption mode. `ENCRYPTION_MODE=DUAL` encrypts the dump set using the TDE master encryption key stored in the keystore and the password provided.

For example, to use dual encryption mode to export encrypted data:

```
expdp hr DIRECTORY=dpump_dir1
DUMPFILE=hr_enc.dmp
ENCRYPTION=all
ENCRYPTION_PASSWORD=encryption_password
ENCRYPTION_PWD_PROMPT=yes
ENCRYPTION_ALGORITHM=AES256
```

```
ENCRYPTION_MODE=dual

Password: password_for_hr
Encryption Password: password_for_encryption
```

Related Topics

- Exporting and Importing the TDE Master Encryption Key
  You can export and import the TDE master encryption key in different ways.
- *Oracle Database Utilities*
- Creating an Encrypted Column in an External Table
  The external table feature enables you to access data in external sources as if the data were in a database table.

Using Oracle Data Pump with Encrypted Data Dictionary Data

Oracle Data Pump operations provide protections for encrypted passwords and other encrypted data.

When you enable the encryption of fixed-user database passwords in a source database, then an Oracle Data Pump export operation dump stores a known invalid password for the database link password. This password is in place instead of the encrypted password that the export operation extracts from the database. An `ORA-39395: Warning: object <database link name> requires password reset after import` warning message is displayed as a result. If you import data into an Oracle Database 18*c* or later database, then this same warning appears when the database link object with its invalid password is created in the target database. When this happens, you must reset the database link password, as follows:

```
ALTER DATABASE LINK database_link_name CONNECT TO schema_name IDENTIFIED BY
password;
```

To find information about the database link, you can query the `V$DBLINK` dynamic view.

When the encryption of fixed-user database passwords has been disabled in a source database, then there are no changes to Data Pump. The obfuscated database link passwords are exported and imported as in previous releases.

In this case, Oracle recommends the following:

- Set the `ENCRYPTION_PASSWORD` parameter on the `expdp` command so that you can further protect the obfuscated database link passwords.
- Set the `ENCRYPTION_PWD_PROMPT` parameter to `YES` so that the password can be entered interactively from a prompt, instead of being echoed on the screen.

Both the `ENCRYPTION_PASSWORD` and the `ENCRYPTION_PWD_PROMPT` parameters are available in import operations. `ENCRYPTION_PWD_PROMPT` is only available with the `expdp` and `impdp` command-line clients, whereas `ENCRYPTION_PASSWORD` is available in both the command-line clients and the `DBMS_DATAPUMP` PL/SQL package.

During an import operation, whether the keystore is open or closed affects the behavior of whether or not an encryption password must be provided. If the keystore was open during the export operation and you provided an encryption password, then you do not need to provide the password during the import operation. If the keystore is closed during the export operation, then you must provide the password during the import operation.

Related Topics

- *Oracle Database Reference*
- *Oracle Database Utilities*

How Transparent Data Encryption Works with Oracle Data Guard

An Oracle Data Guard primary database and secondary secondary database can share both a software keystore and a hardware security module.

- About Using Transparent Data Encryption with Oracle Data Guard
  For both software keystores and hardware keystores, Oracle Data Guard supports Transparent Data Encryption (TDE).
- Configuring TDE and Oracle Key Vault in an Oracle Data Guard Environment
  You can configure Oracle Data Guard in a multitenant environment so that it can work with TDE and Oracle Key Vault.

About Using Transparent Data Encryption with Oracle Data Guard

For both software keystores and hardware keystores, Oracle Data Guard supports Transparent Data Encryption (TDE).

If the primary database uses TDE, then each standby database in a Data Guard configuration must have a copy of the encryption keystore from the primary database. If you reset the TDE master encryption key in the primary database, then you must copy the keystore from the primary database that contains the TDE master encryption key to each standby database.

Note the following:

- Re-key operations with wallet-based TDE will cause the Managed Recovery Process (MRP) on the standby databases to fail because the new TDE master encryption key is not yet available. In order to circumvent this problem, use the `ADMINISTER KEY MANAGEMENT CREATE KEY` statement on the primary database to insert new TDE master encryption keys into the wallet. Copy the wallet to the standby databases, and then execute a `ADMINISTER KEY MANAGEMENT USE KEY` statement on the primary.
- Encrypted data in log files remains encrypted when data is transferred to the standby database. Encrypted data also stays encrypted during transit.

Related Topics

- Merging Software Keystores
  You can merge software keystores in a variety of ways.

Configuring TDE and Oracle Key Vault in an Oracle Data Guard Environment

You can configure Oracle Data Guard in a multitenant environment so that it can work with TDE and Oracle Key Vault.

The following scenario shows the configuration with Oracle Key Vault in a single-instance, multitenant Oracle Data Guard environment with one physical standby database. The version for the primary and standby databases must be release 19.6 or later. To complete this procedure, you must perform each step in the sequence shown. After you complete the procedure, Oracle Data Guard will use Oracle Key Vault for TDE key management exclusively, and there will be no TDE wallet on your database servers. Oracle recommends that you monitor the alert logs of both primary and standby databases.

1. On both the primary and standby databases, execute the `opatch lspatches` command to check the patch release.

   ```
   $ORACLE_HOME/OPatch/opatch lspatches
   ```

   Output similar to the following appears:

   ```
   31281355;Database Release Update : 19.7.0.0.200414 (30869156)
   29585399;OCW RELEASE UPDATE 19.3.0.0.0 (29585399)
   ```

2. Download the Oracle Key Vault deployment script that the Oracle Key Vault administrators prepared to enable database administrators to automatically register their Oracle databases with Oracle Key Vault.

   Oracle Key Vault Administrator's Guide has an example of how to create a script to automatically enroll Oracle databases as endpoints. The deployment scripts reside on a shared file system from which database administrators can download. There are two different versions of these deployment scripts. The `primary.zip` file is for the primary database, and the `secondary.zip` file is for all standby databases. You can use these scripts for an Oracle Data Guard or an Oracle RAC environment.

   Another component that the Oracle Key Vault administrators prepare and add to the deployment script is a configuration file that contains all details for the deployment scripts to connect to Oracle Key Vault.

3. Copy the two deployment scripts (`primary.zip` and `secondary.zip`) that an Oracle Key Vault administrator created for database administrators to download from a shared location.

   a. Copy the `primary.zip` file to the primary database.

      ```
      $ scp user@ip_address:/path/to/file/primary.zip .
      ```

   b. Copy the `secondary.zip` file to the standby database.

      ```
      $ scp user@ip_address:/path/to/file/secondary.zip .
      ```

4. On their respective servers, extract the zip files.

```
$ unzip primary.zip

$ unzip secondary.zip
```

5. Execute the `primary-run-me.sh` and `secondary-run-me.sh` scripts, which contain the commands for the RESTful API to execute in Oracle Key Vault.

   The Oracle Key Vault RESTful services will execute these commands in order to register this database in Oracle Key Vault with unique wallet and endpoint names.

   a. **Primary database:** For example:

   ```
   $ ./primary-run-me.sh

   create_wallet -w $ORACLE_SID
   create_endpoint -e $ORACLE_SID_on_short_host_name -t ORACLE_DB -q
   LINUX64 -d "long_host_name, 192.0.2.101"
   set_default_wallet -e $ORACLE_SID_on_short_host_name -w $ORACLE_SID
   provision -v default_password -e $ORACLE_SID_on_short_host_name -y
   Oracle_Key_Vault_installation_directory
   ```

   b. **Standby database:** For example:

   ```
   $ ./secondary-run-me.sh

   create_endpoint -e $ORACLE_SID_on_short_host_name -t ORACLE_DB -q
   LINUX64 -d "long_host_name, 192.0.2.102"
   set_default_wallet -e $ORACLE_SID_on_short_host_name -w $ORACLE_SID
   provision -v default_password -e $ORACLE_SID_on_short_host_name -y
   Oracle_Key_Vault_installation_directory
   ```

6. Create the following directories on the primary database and the standby database.

   For example:

   ```
   $ mkdir -pv /u01/opt/oracle/product/okv
   $ mkdir -pv /u01/opt/oracle/product/tde
   $ mkdir -pv /u01/opt/oracle/product/tde_seps
   ```

   In this specification:

   - The `/u01/opt/oracle/product` directory will be defined as `WALLET_ROOT` in a later step.
   - `/u01/opt/oracle/product/okv` is the installation directory for the Oracle Key Vault client software. Depending on how the `TDE_CONFIGURATION` parameter is set, the Oracle Database will look for the Oracle Key Vault client software in *`wallet_root/okv`*.
   - `/u01/opt/oracle/product/tde` will store an auto-login wallet, which only contains the future Oracle Key Vault password, enabling an auto-login Oracle Key Vault configuration. Depending on how `TDE_CONFIGURATION` is set, the Oracle Database will look for the TDE wallet or an auto-open wallet for Oracle Key Vault, in *`wallet_root/tde`*.
   - `/u01/opt/oracle/product/tde_seps` will store an auto-login wallet, which only contains the future Oracle Key Vault password. This will hide the Oracle Key Vault password from the SQL*Plus command line and potentially from the database administrator to enforce separation of duties between Oracle database administrators and Oracle Key Vault administrators.

7. Execute the RESTful API on the primary database first, because the deployment script on the standby databases depends on the presence of the shared virtual wallet in Oracle Key Vault that the script on the primary database creates.

   ```
   $ java -jar okvrestservices.jar -c config.ini
   ```

   The following message should appear:

   ```
   The endpoint software for Oracle Key Vault installed successfully
   ```

8. On the primary and standby databases, execute the `root.sh` script to deploy the PKCS#11 library.

   ```
   # /u01/opt/oracle/product/okv/bin/root.sh
   ```

   The following output should appear:

   ```
   Creating directory: /opt/oracle/extapi/64/hsm/oracle/1.0.0/
   Copying PKCS library to /opt/oracle/extapi/64/hsm/oracle/1.0.0/
   Setting PKCS library file permissions
   ```

9. Execute the `okvutil changepwd` command to change the password for the wallet that you installed, starting from the primary database and then to the standby.

   Because all database administrators downloaded the same deployment script, all databases have the same password into Oracle Key Vault. This step enables each database to have a unique password.

   ```
   $ /u01/opt/oracle/product/okv/bin/okvutil changepwd -t wallet -l /u01/opt/
   oracle/product/okv/ssl/

   Enter wallet password: default_password
   Enter new wallet password: Oracle_Key_Vault_password
   Confirm new wallet password: Oracle_Key_Vault_password
   Wallet password changed successfully
   ```

10. On the primary and standby databases, execute the following statements.

    a. Execute the following statement to add the Oracle Key Vault password as a secret into an auto-open wallet to replace the Oracle Key Vault password in the SQL*Plus command line with `EXTERNAL STORE`.

       ```
       ADMINISTER KEY MANAGEMENT ADD SECRET 'Oracle_Key_Vault_password'
       FOR CLIENT 'OKV_PASSWORD'
       TO LOCAL AUTO_LOGIN KEYSTORE '/u01/opt/oracle/product/tde_seps';
       ```

    b. Execute the following statement to add the Oracle Key Vault password as a secret into an auto-open wallet to enable auto-open Oracle Key Vault.

       ```
       ADMINISTER KEY MANAGEMENT ADD SECRET 'Oracle_Key_Vault_password'
       FOR CLIENT 'HSM_PASSWORD'
       TO LOCAL AUTO_LOGIN KEYSTORE '/u01/opt/oracle/product/tde';
       ```

    c. Configure the primary and standby databases to always encrypt new tablespaces:

       ```
       ALTER SYSTEM SET ENCRYPT_NEW_TABLESPACES = ALWAYS SCOPE = BOTH;
       ```

    d. Optionally, if patch 30398099 is installed, then change the database default algorithm from `AES128` to either `AES192` or `AES256`.

       Note that the following parameter is preceded by an underscore.

       ```
       ALTER SYSTEM SET "_tablespace_encryption_default_algorithm" = 'AES256'
       SCOPE = BOTH;
       ```

e. In the primary and standby databases, define the WALLET_ROOT static initialization parameter:

```
ALTER SYSTEM SET WALLET_ROOT = '/u01/opt/oracle/product' SCOPE = SPFILE;
```

f. Restart the primary and standby databases so that the preceding ALTER SYSTEM SET WALLET_ROOT statement takes effect.

g. After the database restarts, configure TDE to use Oracle Key Vault as the first keystore and the auto-open wallet in WALLET_ROOT/tde as the secondary keystore.

Execute the following statement in both the primary and standby databases:

```
ALTER SYSTEM SET TDE_CONFIGURATION = "KEYSTORE_CONFIGURATION=OKV|FILE"
SCOPE = BOTH;
```

11. In the primary database, create your first TDE master encryption keys in Oracle Key Vault.

Check the alert.log of the standby database. The managed recovery process (MRP) should not be stopped, since the standby database finds the correct master key in the shared virtual wallet in Oracle Key Vault.

a. **Primary root container:** Set the first master encryption key.

For all ADMINISTER KEY MANAGEMENT statements that do not change the TDE configuration, the password will be replaced by EXTERNAL STORE. This enables separation of duties between the database administrators and the Oracle Key Vault administrators because the Oracle Key Vault administrators do not need to share the Oracle Key Vault password with the database administrators.

```
sqlplus sys as syskm
Enter password: password

ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY EXTERNAL STORE
CONTAINER = CURRENT;
```

b. **All primary PDBs:** Set the first, tagged, master key for each open PDB. The benefit of tagging the PDB keys is that they can later be easily identified to belong to a certain PDB.

```
SELECT ' ADMINISTER KEY MANAGEMENT SET KEY
USING TAG '''||SYS_CONTEXT('USERENV', 'CON_NAME')||' '||
TO_CHAR (SYSDATE, 'YYYY-MM-DD HH24:MI:SS')||'''
IDENTIFIED BY EXTERNAL STORE;' "SET KEY COMMAND" FROM DUAL;
```

c. Execute the generated output of this SELECT statement.

For example:

```
ADMINISTER KEY MANAGEMENT SET KEY
USING TAG 'pdb_name date time'
IDENTIFIED BY EXTERNAL STORE;
```

12. Perform the following steps in the root container.

a. Optionally, encrypt the USERS tablespace in the root container. While technically possible, you should not encrypt the SYSTEM, SYSAUX, TEMP, and UNDO tablespaces of the root container.

For example:

```
ALTER TABLESPACE USERS ENCRYPTION ONLINE USING 'AES256' ENCRYPT;
```

b. Observe the `alert.log` of the standby database to confirm that the `USERS` tablespace there is also encrypted.

c. As a user with the `SYSKM` administrative privilege, encrypt the data dictionary with the `AES256` algorithm.

```
ALTER DATABASE DICTIONARY ENCRYPT CREDENTIALS CONTAINER =
CURRENT;
```

13. Optionally, encrypt the `USERS`, `SYSTEM`, and `SYSAUX` tablespaces in the PDBs.

    Encrypting the `TEMP` and `UNDO` tablespaces is optional because data from encrypted tablespaces is tracked and automatically encrypted before being written into `TEMP` or `UNDO`.

```
ALTER TABLESPACE USERS ENCRYPTION ONLINE USING 'AES256' ENCRYPT;

ALTER TABLESPACE SYSTEM ENCRYPTION ONLINE ENCRYPT;

ALTER TABLESPACE SYSAUX ENCRYPTION ONLINE USING 'AES256' ENCRYPT;
```

    The `SYSTEM` tablespace can only be encrypted with the database default algorithm, which is `AES128` unless it has been changed after you applied patch 30398099. If you do not have this patch and want to encrypt the `SYSTEM` tablespace with `AES256`, you must rekey the `SYSTEM` tablespace to use `AES256`. For example:

```
ALTER TABLESPACE SYSTEM ENCRYPTION ONLINE USING 'AES256' REKEY;
```

    Observe the `alert.log` of the standby database to confirm the encryption and rekey operations are applied there as well.

14. Create a tablespace and table in the primary database PDB.

    When you create the tablespace in the primary database even though there are no encryption keywords in that statement. It will be encrypted with `AES128` by default unless the database default algorithm has been changed after patch 30398099 has been applied. Observe the `alert.log` of the standby database to confirm the encrypted tablespace is created there as well.

```
CREATE TABLESPACE protected DATAFILE SIZE 50M;

CREATE TABLE SYSTEM.TEST TABLESPACE protected
AS SELECT * FROM DBA_OBJECTS;
```

15. Confirm that you can select from the table that is stored in an encrypted tablespace.

```
SELECT COUNT(*), OWNER FROM SYSTEM.TEST
GROUP BY OWNER
ORDER BY 1 DESC;
```

16. On the **standby** database, execute the following query to list the encrypted tablespaces for the root container, all PDBs, and the encryption algorithm.

```
SELECT C.NAME AS PDB_NAME, T.NAME AS TBS_NAME, E.ENCRYPTIONALG AS ALG
FROM V$TABLESPACE T, V$ENCRYPTED_TABLESPACES E, V$CONTAINERS C
WHERE E.TS# = T.TS# AND E.CON_ID = T.CON_ID AND E.CON_ID = C.CON_ID ORDER BY
E.CON_ID, T.NAME;

PDB_NAME        TBS_NAME             ALG
--------------- -------------------- -------
CDB$ROOT        USERS                AES256
```

```
FINPDB          PROTECTED01          AES256
FINPDB          SYSAUX               AES256
FINPDB          SYSTEM               AES256
FINPDB          USERS                AES256
```

17. Optionally, validate the configuration.

    a. Perform an Oracle Data Guard switchover between the primary and standby databases.

       See *Oracle Data Guard Concepts and Administration*.

       Perform the following steps in the new primary database.

    b. Select from the encrypted table in your PDB.

       Because there is an auto-open connection into Oracle Key Vault, the following query does not require that you enter the Oracle Key Vault password.

       ```
       SELECT COUNT(*), OWNER FROM SYSTEM.TEST
       GROUP BY OWNER
       ORDER BY 1 DESC;

       24 rows selected.
       ```

    c. Rekey the PDB.

       ```
       SELECT ' ADMINISTER KEY MANAGEMENT SET KEY
       USING TAG '''||SYS_CONTEXT('USERENV', 'CON_NAME')||' '||
       TO_CHAR (SYSDATE, 'YYYY-MM-DD HH24:MI:SS')||'''
       IDENTIFIED BY EXTERNAL STORE;' "REKEY COMMAND" FROM DUAL;
       ```

    d. Execute the generated output of this `SELECT` statement.

       For example:

       ```
       ADMINISTER KEY MANAGEMENT SET KEY
       USING TAG 'pdb_name date time'
       IDENTIFIED BY EXTERNAL STORE;
       ```

    e. In the root container, as a user who has the `SYSKM` administrative privilege, rekey the data dictionary.

       ```
       ALTER DATABASE DICTIONARY REKEY CREDENTIALS CONTAINER = CURRENT;
       ```

    f. Perform another Oracle Data Guard switchover.

       See *Oracle Data Guard Concepts and Administration*.

    g. Select from the encrypted table in your PDB.

       Because there is an auto-open connection into Oracle Key Vault, the following query does not require that you enter the Oracle Key Vault password.

       ```
       SELECT COUNT(*), OWNER FROM SYSTEM.TEST
       GROUP BY OWNER
       ORDER BY 1 DESC;

       24 rows selected.
       ```

    h. Rekey the PDB.

       ```
       SELECT ' ADMINISTER KEY MANAGEMENT SET KEY
       USING TAG '''||SYS_CONTEXT('USERENV', 'CON_NAME')||' '||
       TO_CHAR (SYSDATE, 'YYYY-MM-DD HH24:MI:SS')||'''
       IDENTIFIED BY EXTERNAL STORE;' "REKEY COMMAND" FROM DUAL;
       ```

    i. Execute the generated output of this `SELECT` statement.

For example:

```
ADMINISTER KEY MANAGEMENT SET KEY
USING TAG 'pdb_name date time'
IDENTIFIED BY EXTERNAL STORE;
```

How Transparent Data Encryption Works with Oracle Real Application Clusters

Oracle Real Application Clusters (Oracle RAC) nodes can share both a software keystore and a hardware security module.

- About Using Transparent Data Encryption with Oracle Real Application Clusters
  Oracle Database enables Oracle Real Application Clusters (Oracle RAC) nodes to share a software keystore.
- Configuring TDE in Oracle Real Application Clusters for Oracle Key Vault
  You can configure TDE in Oracle Real Application Clusters (Oracle RAC) on Oracle Exadata Cloud at Customer (ExaCC) and other servers for centralized key management provided by Oracle Key Vault.

About Using Transparent Data Encryption with Oracle Real Application Clusters

Oracle Database enables Oracle Real Application Clusters (Oracle RAC) nodes to share a software keystore.

A TDE configuration with Oracle Key Vault or a PKCS11-compatible hardware security module uses a network connection from each instance of the database to the external key manager. In Oracle Key Vault, you must create one endpoint for each instance of the Oracle RAC-enabled database, and one virtual wallet for each Oracle RAC-enabled database. Then, make that virtual wallet the default wallet of all endpoints that belong to that database. In an Oracle RAC-enabled Data Guard configuration, all instances (primary and all standby databases) share that one virtual wallet. With this configuration, set key and re-key operations are completely transparent because all participating instances are automatically synchronized. This eliminates the need to manually copy the software keystore to each of the other nodes in the cluster. Oracle recommends that you create the software keystore on a shared file system.

Oracle does not support the use of individual TDE wallets for each Oracle RAC node. Instead, use shared wallets for TDE in the Oracle RAC environment. This enables all of the instances to access the same shared software keystore. If your site uses Oracle Automatic Storage Management Cluster File System (Oracle ACFS), then this is the preferred location for a shared wallet. Directly sharing the wallet in Oracle Automatic Storage Management (Oracle ASM) (for example, `+DATA/$ORACLE_UNQNAME/WALLETS`) is an alternative if Oracle ACFS is not available.

Keystore operations (such as opening or closing the keystore, or rekeying the TDE master encryption key) can be issued on any one Oracle RAC instance. Internally, the Oracle database takes care of synchronizing the keystore context on each Oracle RAC node, so that the effect of the keystore operation is visible to all of the other Oracle RAC instances in the cluster. Similarly, when a TDE master encryption key rekey operation takes place, the new key becomes available to each of the Oracle RAC instances. You can perform other keystore operations, such as exporting TDE

master encryption keys, rotating the keystore password, merging keystores, or backing up keystores, from a single instance only.

When using a shared file system, ensure that the `WALLET_ROOT` static system parameter for all of the Oracle RAC instances point to the same shared software keystore location, as follows:

```
ALTER SYSTEM SET WALLET_ROOT = '+DATA/$ORACLE_UNQNAME/WALLETS' SCOPE = SPFILE
SID ='*';

ALTER SYSTEM SET TDE_CONFIGURATION = "KEYSTORE_CONFIGURATION=FILE" SCOPE = BOTH
SID = '*';
```

Note:

- If you have the `ENCRYPTION_WALLET_LOCATION` parameter set, then be aware this parameter is deprecated. Oracle recommends that you use the `WALLET_ROOT` static initialization parameter and `TDE_CONFIGURATION` dynamic initialization parameter instead.
- Storing TDE master encryption keys in individual wallets per Oracle Real Application Clusters (Oracle RAC) instance is not supported. As an alternative, use Oracle Key Vault for centralized key management across your on-premises or Cloud-based database deployments, or Oracle Automatic Storage Management (Oracle ASM), or Oracle ASM Cluster File System (Oracle ACFS) to provide local shared wallets.

Configuring TDE in Oracle Real Application Clusters for Oracle Key Vault

You can configure TDE in Oracle Real Application Clusters (Oracle RAC) on Oracle Exadata Cloud at Customer (ExaCC) and other servers for centralized key management provided by Oracle Key Vault.

The following scenario assumes that you have a multitenant two-node Oracle RAC configuration. In this procedure, you must complete the following steps in the order shown. After you have completed this procedure, the Oracle RAC environment will exclusively use Oracle Key Vault for key management for Transparent Data Encryption. This procedure assumes that you have installed the January 2020 (19.6) or later upgrade for Oracle Database release 19c.

Before you begin, monitor the alert logs of your running Oracle RAC database. The Java version that is included in the default Oracle Database release 19c installation can be used to install the Oracle Key Vault client with the RESTful services. The `provision` command of the RESTful API requires a symbolic link in `/usr/bin/java` that points to the Java home in the Oracle database. For example:

```
# ln -sv $ORACLE_HOME/jdk/jre/java /usr/bin/java
```

1. Download the Oracle Key Vault deployment script that the Oracle Key Vault administrators prepared to enable database administrators to automatically register their Oracle databases with Oracle Key Vault.

Oracle Key Vault Administrator's Guide has an example of how to create a script to automatically enroll Oracle databases as endpoints. The deployment scripts reside on a shared file system from with database administrators can download. There are two different versions of these deployment scripts. One script is for only the first node (which is called lead node in this procedure) and the other script is for all other nodes (which are called secondary nodes in this procedure). You can use these scripts for an Oracle RAC or an Oracle Data Guard environment.

Another component that the Oracle Key Vault administrators prepare and add to the deployment script is a configuration file that contains all details for the deployment scripts to connect to Oracle Key Vault.

2. Copy the two deployment scripts (`primary.zip` and `secondary.zip`) that an Oracle Key Vault administrator created for database administrators to download from a shared location.

   a. Copy the `primary.zip` file to the lead node.

   ```
   $ scp user@ip_address:/path/to/file/primary.zip .
   ```

   b. Copy the `secondary.zip` file to each secondary node.

   ```
   $ scp user@ip_address:/path/to/file/secondary.zip .
   ```

3. Extract the zip files.

   a. **On the lead node:** Extract the `primary.zip` file.

   ```
   $ unzip primary.zip
   ```

   b. **On the secondary nodes:** Extract the `secondary.zip` file.

   ```
   $ unzip secondary.zip
   ```

4. Execute the `primary-run-me.sh` and `secondary-run-me.sh` scripts, which contain the commands for the RESTful API to execute in Oracle Key Vault.

   The Oracle Key Vault RESTful services will execute these commands in order to register this database in Oracle Key Vault with unique wallet and endpoint names.

   a. **On the lead node:** This script creates a shared wallet (for the lead and all secondary nodes) and an endpoint in Oracle Key Vault, associates this endpoint for the lead node with the shared wallet, and downloads and installs the Oracle Key Vault client into an **existing** installation directory. With the `WALLET_ROOT` configuration, this directory is *`wallet_root`*`/okv`.

   ```
   $ more primary-run-me.sh
   #!/bin/bash
   export EP_NAME=${ORACLE_SID^^}_on_${HOSTNAME/.*}
   cat > /home/oracle/script.txt << EOF
   create_wallet -w $ORACLE_UNQNAME
   create_endpoint -e $EP_NAME -t ORACLE_DB -q LINUX64 -d "$HOSTNAME, $
   (hostname -i)"
   set_default_wallet -e $EP_NAME -w $ORACLE_UNQNAME
   provision -v default_password -e $EP_NAME -y
   Oracle_Key_Vault_installation_directory
   EOF
   more script.txt
   ```

   Output similar to the following appears:

```
create_wallet -w database_name
create_endpoint -e instance_name_on_short_host_name -t ORACLE_DB -q
LINUX64 -d "full_host_name, node_IP_address"
set_default_wallet -e instance_name_on_short_host_name -w database_name
provision -v default_password -e instance_name_on_short_host_name -y
Oracle_Key_Vault_installation_directory
```

b. **Secondary nodes:** This script only creates an endpoint for the secondary nodes, associates the endpoint of the secondary nodes with the shared wallet, and downloads and installs the Oracle Key Vault client into the **existing** installation directory on each secondary node.

```
$ more secondary-run-me.sh
#!/bin/bash
export EP_NAME=${ORACLE_SID^^}_on_${HOSTNAME/.*}
cat > /home/oracle/script.txt << EOF
create_endpoint -e $EP_NAME -t ORACLE_DB -q LINUX64 -d "$HOSTNAME, $
(hostname -i)"
set_default_wallet -e $EP_NAME -w $ORACLE_UNQNAME
provision -v default_password -e $EP_NAME -y
Oracle_Key_Vault_installation_directory
EOF
more script.txt
```

Output similar to the following appears:

```
create_endpoint -e instance_name_on_short_host_name -t ORACLE_DB -q
LINUX64 -d "full_host_name, node_IP_address"
set_default_wallet -e instance_name_on_short_host_name -w database_name
provision -v default_password -e instance_name_on_short_host_name -y
Oracle_Key_Vault_installation_directory
```

5. Create the following directories on all nodes:

For example:

```
$ mkdir -pv /u01/opt/oracle/product/okv
$ mkdir -pv /u01/opt/oracle/product/tde
$ mkdir -pv /u01/opt/oracle/product/tde_seps
```

In this specification:

- The `/u01/opt/oracle/product` directory will be defined as `WALLET_ROOT` in a later step.
- `/u01/opt/oracle/product/okv` is the installation directory for the Oracle Key Vault client software. Depending on how the `TDE_CONFIGURATION` parameter is set, the Oracle Database will look for the Oracle Key Vault client software in *`wallet_root/okv`*.
- `/u01/opt/oracle/product/tde` will store an auto-login wallet, which only contains the future Oracle Key Vault password, enabling an auto-login Oracle Key Vault configuration. Depending on how `TDE_CONFIGURATION` is set, the Oracle Database will look for the TDE wallet or an auto-open wallet for Oracle Key Vault, in *`wallet_root/tde`*.
- `/u01/opt/oracle/product/tde_seps` will store an auto-login wallet, which only contains the future Oracle Key Vault password. This will hide the Oracle Key Vault password from the SQL*Plus command line and potentially from the database administrator to enforce separation of duties between Oracle database administrators and Oracle Key Vault administrators.

6. Execute the RESTful API on the lead node first, because all secondary nodes depend on the presence of the shared wallet in Oracle Key Vault that the lead node creates.

   ```
   $ java -jar okvrestservices.jar -c config.ini
   ```

   Output similar to the following should appear:

   ```
   [Line 1 OK] [CREATE WALLET] [3E48990A-82A0-48BC-ACEC-FF80CB380D38]
   [Line 2 OK] [CREATE ENDPOINT] [6FA40F80-558C-456A-84E3-25AE73B245DD]
   [Line 3 OK] [SET DEFAULT WALLET] [FINRAC1_on_rac19a:FINRAC]
   [Line 4 OK] [GET ENROLLMENT TOKEN] [FINRAC1_on_rac19a]
   The endpoint software for Oracle Key Vault installed successfully.
   [Line 4 OK] [PROVISION]
   [Line 4 OK] [CLEANUP]
   ```

   Make a note of the UUID of the wallet, which appears in Line 1 and is in **bold**.

   After you execute this command on the lead node, execute it on all secondary nodes.

7. After successful installation of the Oracle Key Vault client, execute the `root.sh` script to install the PKCS library on all nodes.

   ```
   # Oracle_Key_Vault_installation_directory/bin/root.sh
   ```

   The following output should appear:

   ```
   Creating directory: /opt/oracle/extapi/64/hsm/oracle/1.0.0/
   Copying PKCS library to /opt/oracle/extapi/64/hsm/oracle/1.0.0/
   Setting PKCS library file permissions
   ```

8. Execute the Oracle Key Vault `okvutil changepwd` command on all nodes to change the password for the Oracle Key Vault client that you installed.

   Because all database administrators downloaded the same deployment script, all databases have the same password into Oracle Key Vault. This step enables each database to have a unique password.

   ```
   $ /u01/opt/oracle/product/okv/bin/okvutil changepwd -t wallet -l /u01/opt/
   oracle/product/okv/ssl/

   Enter wallet password: default_password
   Enter new wallet password: Oracle_Key_Vault_password
   Confirm new wallet password: Oracle_Key_Vault_password
   Wallet password changed successfully
   ```

9. On all nodes, add the Oracle Key Vault password into a local auto-login wallet to hide the newly changed password from database administrators.

   ```
   sqlplus c##sec_admin as syskm
   Enter password: password

   ADMINISTER KEY MANAGEMENT ADD SECRET 'Oracle_Key_Vault_password'
   FOR CLIENT 'OKV_PASSWORD'
   TO LOCAL AUTO_LOGIN KEYSTORE '/u01/opt/oracle/product/tde_seps';
   ```

10. In the root container, execute the `ALTER SYSTEM` statement to set the static `WALLET_ROOT` parameter to configure the encryption wallet location for all instances:

    ```
    CONNECT AS SYSDBA
    ```

```
ALTER SYSTEM SET WALLET_ROOT = '/u01/opt/oracle/product/'
SCOPE = SPFILE SID = '*';
```

11. Restart the database.

    Execute this command on any node.

    ```
    $ srvctl stop database -db database_name -o immediate
    $ srvctl start database -db database_name
    ```

12. Check if the wallet has been replicated across the Oracle Key Vault cluster nodes:

    ```
    $ java -jar okvrestservices.jar -c /u01/opt/oracle/product/okv/conf/
    okvclient.ora
    -j /home/oracle -u restadmin -r check_object_status -b WALLET
    -x 3E48990A-82A0-48BC-ACEC-FF80CB380D38
    ```

    In this specification:

    - `-c /u01/opt/oracle/product/okv/conf/okvclient.ora` enables the RESTful API to use a complete configuration file (`okvclient.ora`) to leverage Oracle Key Vault's clustering capabilities.
    - `-j /home/oracle` is the corresponding entry for the `client_wallet` entry in the `config.ini` file.
    - `-u restadmin` is the corresponding entry for the `usr` entry in the `config.ini` file.
    - `-r check_object_status` checks the status of an object in Oracle Key Vault.
    - `-b WALLET` is the object type that is being checked with `-r check_object_status`.
    - `-x 3E48990A-82A0-48BC-ACEC-FF80CB380D38` is the generated UUID that appeared when the wallet was created in Oracle Key Vault in Step 6.

    Output similar to the following should appear. If the status is `ACTIVE`, then continue. If the status is `PENDING`, then wait until the status is `ACTIVE`.

    ```
    [Line 0 OK] [CHECK OBJECT STATUS] [ACTIVE]
    ```

13. In the root container, use the `ALTER SYSTEM` statement to set the dynamic `TDE_CONFIGURATION` parameter.

    For example:

    ```
    ALTER SYSTEM SET TDE_CONFIGURATION = "KEYSTORE_CONFIGURATION=OKV"
    SCOPE = BOTH SID = '*';
    ```

14. Optionally, define the database default encryption algorithm after applying Oracle patch 30398099.

    By default, Oracle Database applies the `AES128` algorithm to encryption clauses that do not specify an encryption algorithm. Patch 30398099 allows you to choose from the `AES128`, `AES192`, and `AES256` encryption algorithms. If you have applied this patch, then you can execute the following command to set the encryption clause:

    ```
    ALTER SYSTEM SET "_TABLESPACE_ENCRYPTION_DEFAULT_ALGORITHM" =
    'encryption_algorithm'
    SCOPE = BOTH SID = '*';
    ```

15. In the root container, open the keystore, which opens the connection to Oracle Key Vault for the root container and all open PDBs.

Note that the Oracle Key Vault password has been replaced in all subsequent `ADMINISTER KEY MANAGEMENT` commands with `EXTERNAL STORE`, because the database automatically retrieves the Oracle Key Vault password from the local auto-login wallet that you created in Step 9.

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
IDENTIFIED BY EXTERNAL STORE
CONTAINER = ALL;
```

16. In the root container, set the master encryption key.

```
ADMINISTER KEY MANAGEMENT SET KEY
IDENTIFIED BY EXTERNAL STORE
CONTAINER = CURRENT;
```

17. Create and activate a tagged master encryption key in all PDBs in this container.

    The benefit of adding tagged master encryption keys to PDBs is that it enables you to easily identify keys that belong to a certain PDB.

    a. Connect to each PDB and execute the following `SELECT` statement to create an `ADMINISTER KEY MANAGEMENT` command that contains the PDB name and time stamp as a tag for the PDB's master encryption key.

```
CONNECT sec_admin@pdb_name AS SYSKM
Enter password: password

SELECT ' ADMINISTER KEY MANAGEMENT SET KEY
USING TAG '''||SYS_CONTEXT('USERENV', 'CON_NAME')||' '||
TO_CHAR (SYSDATE, 'YYYY-MM-DD HH24:MI:SS')||'''
IDENTIFIED BY EXTERNAL STORE;' "SET KEY COMMAND" FROM DUAL;
```

    b. Execute the generated output of this `SELECT` statement.

       For example:

```
ADMINISTER KEY MANAGEMENT SET KEY
USING TAG 'pdb_name date time'
IDENTIFIED BY EXTERNAL STORE;
```

18. On all nodes, add the Oracle Key Vault password into an auto-login wallet to enable auto-login connection into Oracle Key Vault.

    This step is mandatory in Oracle RAC. Having an auto-login connection into Oracle Key Vault is especially important when Oracle RAC nodes are automatically restarted (for example, while applying quarterly release upgrades using the `opatchauto` patch tool).

```
ADMINISTER KEY MANAGEMENT ADD SECRET 'Oracle_Key_Vault_password'
FOR CLIENT 'HSM_PASSWORD'
TO LOCAL AUTO_LOGIN KEYSTORE '/u01/opt/oracle/product/tde';
```

19. In the root container, execute the `ALTER SYSTEM` statement to change the `TDE_CONFIGURATION` parameter.

    For example:

```
ALTER SYSTEM SET TDE_CONFIGURATION = "KEYSTORE_CONFIGURATION=OKV|FILE"
SCOPE = BOTH SID = '*';
```

20. Change the default behavior of the database to always encrypt new tablespaces with the `AES128` algorithm (or the algorithm that you specified in Step 14), even if the `CREATE TABLESPACE` command does not contain the encryption clauses.

    Execute this statement once on any node:

```
ALTER SYSTEM SET ENCRYPT_NEW_TABLESPACES = ALWAYS SCOPE = BOTH SID = '*';
```

21. From the root, encrypt sensitive credential data with AES256 for database links in the SYS.LINK$ and SYS.SCHEDULER$_CREDENTIAL system tables.

    This command requires the SYSKM administrative privilege:

    ```
    sqlplus c##sec_admin as syskm
    Enter password: password

    ALTER DATABASE DICTIONARY ENCRYPT CREDENTIALS;
    ```

22. Log in to the PDB and create a tablespace.

    For example, to create a tablespace named protected:

    ```
    CREATE TABLESPACE protected DATAFILE SIZE 50M;
    ```

23. Confirm that the tablespace is encrypted even though the encryption clauses were omitted.

    ```
    SELECT C.NAME AS pdb_name, T.NAME AS tablespace_name, E.ENCRYPTIONALG AS ALG
    FROM V$TABLESPACE T, V$ENCRYPTED_TABLESPACES E, V$CONTAINERS C
    WHERE E.TS# = T.TS# AND E.CON_ID = T.CON_ID AND E.CON_ID = C.CON_ID
    ORDER BY E.CON_ID, T.NAME;
    ```

24. Create a table in the encrypted tablespace that you just created.

    For example:

    ```
    CREATE TABLE SYSTEM.test TABLESPACE protected
    AS SELECT * FROM DBA_OBJECTS;
    ```

25. Select from this table to confirm that you can read encrypted data:

    ```
    SELECT COUNT(*), OWNER FROM SYSTEM.test
    GROUP BY OWNER
    ORDER BY 1 DESC;
    ```

26. In the PDBs, encrypt the existing tablespaces.

    Optionally, encrypt the SYSTEM, SYSAUX, and USERS tablespaces. If you omit the encryption algorithm, then the default algorithm (AES128, or the algorithm that you specified in Step 14) is applied.

    ```
    ALTER TABLESPACE tablespace_name ENCRYPTION ONLINE ENCRYPT;
    ```

27. Optionally, validate the configuration.

    a. Confirm that the auto-login for Oracle Key Vault is working.

       You can test this by restarting the database, logging into the PDB, and then selecting from the encrypted table. To restart the database:

       ```
       $ srvctl stop database -db database_name -o immediate
       $ srvctl start database -db database_name
       ```

       After logging in to the PDB, select from the SYSTEM.test table.

       ```
       SELECT COUNT(*), OWNER FROM SYSTEM.test
       GROUP BY OWNER
       ORDER BY 1 DESC;
       ```

    b. Confirm that the master encryption key re-key operations in all open PDBs are successful.

First, as a user who has the SYSKM administrative privilege, execute the following SELECT statement to create an ADMINISTER KEY MANAGEMENT command that contains the PDB name and time stamp.

```
SELECT ' ADMINISTER KEY MANAGEMENT SET KEY
USING TAG '''||SYS_CONTEXT('USERENV', 'CON_NAME')||' '||
TO_CHAR (SYSDATE, 'YYYY-MM-DD HH24:MI:SS')||'''
FORCE KEYSTORE IDENTIFIED BY EXTERNAL STORE;' "RE-KEY COMMAND" FROM DUAL;
```

Next, execute the generated output of this SELECT statement.

```
ADMINISTER KEY MANAGEMENT SET KEY
USING TAG 'pdb_name date time'
FORCE KEYSTORE IDENTIFIED BY EXTERNAL STORE;
```

c. From the root container, re-key previously encrypted sensitive credential data in the SYS.LINK$ and SYS.SCHEDULER$_CREDENTIAL system tables.

This command requires the SYSKM administrative privilege:

```
ALTER DATABASE DICTIONARY REKEY CREDENTIALS;
```

d. Drop the protected tablespace and its table, test.

```
DROP TABLESPACE protected
INCLUDING CONTENTS AND DATAFILES;
```

Related Topics

- Supported Encryption and Integrity Algorithms
  The supported Advanced Encryption Standard cipher keys, including tablespace and database encryption keys, can be either 128, 192, or 256 bits long. Tablespace and database encryption use the 128–bit length cipher key.

How Transparent Data Encryption Works with SecureFiles

SecureFiles, which stores LOBS, has three features: compression, deduplication, and encryption.

- About Transparent Data Encryption and SecureFiles
  SecureFiles encryption uses TDE to provide the encryption facility for LOBs.
- Example: Creating a SecureFiles LOB with a Specific Encryption Algorithm
  The CREATE TABLE statement can create a SecureFiles LOB with encryption specified.
- Example: Creating a SecureFiles LOB with a Column Password Specified
  The CREATE TABLE statement can create a SecureFiles LOB with a column password.

About Transparent Data Encryption and SecureFiles

SecureFiles encryption uses TDE to provide the encryption facility for LOBs.

When you create or alter tables, you can specify the SecureFiles encryption or LOB columns that must use the SecureFiles storage. You can enable the encryption for a LOB column by either using the current Transparent Data Encryption (TDE) syntax or by using the ENCRYPT clause as part of the LOB parameters for the LOB column. The DECRYPT option in the current syntax or the LOB parameters turn off encryption.

Example: Creating a SecureFiles LOB with a Specific Encryption Algorithm

The `CREATE TABLE` statement can create a SecureFiles LOB with encryption specified.

Example 8-1 shows how to create a SecureFiles LOB in a `CREATE TABLE` statement.

Example 8-1 Creating a SecureFiles LOB with a Specific Encryption Algorithm

```
CREATE TABLE table1 ( a BLOB ENCRYPT USING 'AES256')
    LOB(a) STORE AS SECUREFILE (
    CACHE
    );
```

Example: Creating a SecureFiles LOB with a Column Password Specified

The `CREATE TABLE` statement can create a SecureFiles LOB with a column password.

Example 8-2 shows an example of creating a SecureFiles LOB that uses password protections for the encrypted column.

All of the LOBS in the LOB column are encrypted with the same encryption specification.

Example 8-2 Creating a SecureFiles LOB with a Column Password Specified

```
CREATE TABLE table1 (a VARCHAR2(20), b BLOB)
    LOB(b) STORE AS SECUREFILE (
        CACHE
        ENCRYPT USING 'AES192' IDENTIFIED BY password
    );
```

How Transparent Data Encryption Works with Oracle Call Interface

Transparent Data Encryption does not have any effect on the operation of Oracle Call Interface (OCI).

For most practical purposes, TDE is transparent to OCI except for the row shipping feature. You cannot use the OCI row shipping feature with TDE because the key to make the row usable is not available at the receipt-point.

How Transparent Data Encryption Works with Editions

Transparent Data Encryption does not have any effect on the Editions feature of Oracle Database.

For most practical purposes, TDE is transparent to Editions. Tables are always noneditioned objects. TDE Column Encryption encrypts columns of the table. Editions are not affected by TDE tablespace encryption.

Configuring Transparent Data Encryption to Work in a Multidatabase Environment

Each Oracle database on the same server (such as databases sharing the same Oracle binary but using different data files) must access its own TDE keystore.

Keystores are not designed to be shared among databases. By design, there must be one keystore per database. You cannot use the same keystore for more than one database.

- To configure the use of keystores in a multidatabase environment, use one of the following options:
  - **Option 1:** Specify the keystore location by individually setting the `WALLET_ROOT` static initialization parameter and the `TDE_CONFIGURATION` dynamic initialization parameter (its `KEYSTORE_CONFIGURATION` attribute set to `FILE`) for each CDB (or standalone database). You must set the `KEYSTORE_CONFIGURATION` attribute to `FILE` in order for the `WALLET_ROOT` parameter to work.

    For example:

    ```
    WALLET_ROOT = $ORACLE_BASE/admin/db_unique_name
    TDE_CONFIGURATION="KEYSTORE_CONFIGURATION=FILE"
    ```

  - **Option 2:** If `WALLET_ROOT` and `TDE_CONFIGURATION` are not set, and if the databases share the same Oracle home, then ensure that the `WALLET_LOCATION` and `ENCRYPTION_WALLET_LOCATION` parameters in `sqlnet.ora` are **not** set. By default, `sqlnet.ora` is located in the `$ORACLE_BASE/network/admin` directory (if `$ORACLE_BASE` is set) or in the `$ORACLE_HOME/network/admin` directory.

    This enables Oracle Database to use the keystore that is located in either the `$ORACLE_BASE/admin/db_unique_name/wallet` (assuming `$ORACLE_BASE` is set) or the `$ORACLE_HOME/admin/db_unique_name/wallet` directory.

  - **Option 3:** If options 1 and 2 are not feasible, then use separate `sqlnet.ora` files, one for each database. Ensure that you correctly set the `TNS_ADMIN` environment variable to point to the correct database configuration. However, be aware that the `ENCRYPTION_WALLET_LOCATION` parameter in `sqlnet.ora` is deprecated, starting with release 19c, in favor of the `WALLET_ROOT` and `TDE_CONFIGURATION` initialization parameters.

> **Caution:**
>
> Using a keystore from another database can cause partial or complete data loss.

Related Topics

- Transparent Data Encryption Keystore Search Order
  The search order for the TDE keystore depends on how you have set either the instance initialization parameters, the `sqlnet.ora` parameters, or the environment variables.

- *SQL*Plus User's Guide and Reference*

9

Using sqlnet.ora to Configure Transparent Data Encryption Keystores

If you do not want to use the WALLET_ROOT and TDE_CONFIGURATION parameters to configure the keystore location, then you can use the sqlnet.ora file.

- About the Keystore Location in the sqlnet.ora File
  Oracle recommends that you use the WALLET_ROOT and TDE_CONFIGURATION parameters to configure the keystore location, but an alternative is to configure the sqlnet.ora file.
- Configuring the sqlnet.ora File for a Software Keystore Location
  The sqlnet.ora file can be used to configure the keystore location for a regular file system, for multiple database access, and for use with Oracle Automatic Storage Management (ASM).
- Example: Configuring a Software Keystore for a Regular File System
  You can configure a software keystore for a regular file system.
- Example: Configuring a Software Keystore When Multiple Databases Share the sqlnet.ora File
  You can configure multiple databases to share the sqlnet.ora file.
- Example: Configuring a Software Keystore for an Oracle Automatic Storage Management Disk Group
  You can configure sqlnet.ora for an Oracle Automatic Storage Management (ASM) disk group.

About the Keystore Location in the sqlnet.ora File

Oracle recommends that you use the WALLET_ROOT and TDE_CONFIGURATION parameters to configure the keystore location, but an alternative is to configure the sqlnet.ora file.

However, if you do want to configure the sqlnet.ora file instead, be aware that the instance initialization file-based approach using the WALLET_ROOT and TDE_CONFIGURATION parameters is recommended. It avoids the possibility of inconsistent configuration that can arise when using sqlnet.ora because each connection to the database re-reads the contents of the sqlnet.ora parameter file. With the instance initialization file based approach, because the WALLET_ROOT parameter is read only once when the database instance starts, there is no risk of inconsistency. In a multitenant environment, the use of the WALLET_ROOT initialization parameter initialization parameter is preferred over the use of sqlnet.ora because it enables each tenant PDB to perform independent management operations.

If you have not set the WALLET_ROOT and TDE_CONFIGURATION initialization parameters, then Oracle Database checks the sqlnet.ora file for the directory location of the keystore. This applies to whether the keystore is a software keystore, a hardware module security (HSM) keystore, or an Oracle Key Vault keystore. You must edit the

sqlnet.ora file to define a directory location for the keystore that you plan to create. Ensure that this directory exists beforehand. Preferably, this directory should be empty.

Note the following behavior when you must edit the sqlnet.ora file in Oracle Real Application Clusters (Oracle RAC) or a multitenant environment:

- **In an Oracle RAC environment:** If you are using the srvctl utility and if you want to include environment variables in the sqlnet.ora configuration file, then you must set these environment variables in both the operating system and the srvctl environment. Oracle recommends that you place the keystore on a shared file system, such as Oracle Automatic Storage Management (ASM) or NFS.
- **In a multitenant environment:** Only united mode is supported if sqlnet.ora is used for TDE configuration in the multitenant environment. Isolated mode is only supported if the WALLET_ROOT and TDE_CONFIGURATION parameters are used for TDE configuration.

In the sqlnet.ora file, you must set the ENCRYPTION_WALLET_LOCATION parameter to specify the keystore location. However, be aware that ENCRYPTION_WALLET_LOCATION is deprecated, starting with Oracle Database release 19c in favor of using the WALLET_ROOT and TDE_CONFIGURATION initialization parameters.

By default, the sqlnet.ora file is located in the $ORACLE_BASE/network/admin directory (if the ORACLE_BASE environment variable is set) or $ORACLE_HOME/network/admin location, or in the location set by the TNS_ADMIN environment variable. Ensure that you have properly set the TNS_ADMIN environment variable to point to the correct sqlnet.ora file. When the keystore location is not set in the sqlnet.ora file, then the V$ENCRYPTION_WALLET view displays the default location. You can check the location and status of the keystore in the V$ENCRYPTION_WALLET view.

Related Topics

- Transparent Data Encryption Keystore Search Order
  The search order for the TDE keystore depends on how you have set either the instance initialization parameters, the sqlnet.ora parameters, or the environment variables.
- *SQL*Plus User's Guide and Reference*
- *Oracle Database Net Services Reference*

Configuring the sqlnet.ora File for a Software Keystore Location

The sqlnet.ora file can be used to configure the keystore location for a regular file system, for multiple database access, and for use with Oracle Automatic Storage Management (ASM).

- To create a software keystore on a regular file system, use the following format when you edit the sqlnet.ora file:

```
ENCRYPTION_WALLET_LOCATION=
 (SOURCE=
  (METHOD=FILE)
   (METHOD_DATA=
    (DIRECTORY=path_to_keystore)))
```

If the ***path_to_keystore*** will contain an environment variable, then set this variable in the environment where the database instance is started and before you start the database. If you are using the `srvctl` utility to start the database, then set the environment variable in the `srvctl` environment as well, using the following command:

```
srvctl setenv database -db database_name -env
"environment_variable_name=environment_variable_value"
```

> **Note:**
>
> Starting with Oracle Database release 19c, the `ENCRYPTION_WALLET_LOCATION` parameter is deprecated. Instead, use the `WALLET_ROOT` static initialization parameter and the `TDE_CONFIGURATION` dynamic initialization parameter to configure the wallet location. `WALLET_ROOT` and `TDE_CONFIGURATION` can be used for a regular file system, multiple database access, and ASM.

Related Topics

- Step 1: Configure the Software Keystore Location and Type
  You must configure the keystore location and type by setting `WALLET_ROOT` in `init.ora` and `TDE_CONFIGURATION` in the database instance.

Example: Configuring a Software Keystore for a Regular File System

You can configure a software keystore for a regular file system.

The following example shows how to configure a software keystore location in the `sqlnet.ora` file for a regular file system in which the database name is `orcl`.

```
ENCRYPTION_WALLET_LOCATION=
  (SOURCE=
   (METHOD=FILE)
    (METHOD_DATA=
     (DIRECTORY=/etc/ORACLE/WALLETS/orcl)))
```

> **Note:**
>
> Starting with Oracle Database release 19c, the `ENCRYPTION_WALLET_LOCATION` parameter is deprecated. Instead, use the `WALLET_ROOT` static initialization parameter and the `TDE_CONFIGURATION` dynamic initialization parameter to configure the wallet location.

Related Topics

- Step 1: Configure the Software Keystore Location and Type
  You must configure the keystore location and type by setting `WALLET_ROOT` in `init.ora` and `TDE_CONFIGURATION` in the database instance.

Example: Configuring a Software Keystore When Multiple Databases Share the sqlnet.ora File

You can configure multiple databases to share the `sqlnet.ora` file.

The following example shows how to configure a software keystore location when multiple databases share the `sqlnet.ora` file.

```
ENCRYPTION_WALLET_LOCATION=
 (SOURCE=
  (METHOD=FILE)
   (METHOD_DATA=
    (DIRECTORY=/etc/ORACLE/WALLETS/$ORACLE_SID/)))
```

> **Note:**
>
> Starting with Oracle Database release 19c, the `ENCRYPTION_WALLET_LOCATION` parameter is deprecated. Instead, use the `WALLET_ROOT` static initialization parameter and the `TDE_CONFIGURATION` dynamic initialization parameter to configure the wallet location. You should set these parameters individually for each database.

Related Topics

- Step 1: Configure the Software Keystore Location and Type
  You must configure the keystore location and type by setting `WALLET_ROOT` in `init.ora` and `TDE_CONFIGURATION` in the database instance.

Example: Configuring a Software Keystore for an Oracle Automatic Storage Management Disk Group

You can configure `sqlnet.ora` for an Oracle Automatic Storage Management (ASM) disk group.

The following format shows how to configure a software keystore if you want to create a software keystore location on an ASM disk group:

```
ENCRYPTION_WALLET_LOCATION=
 (SOURCE=
  (METHOD=FILE)
   (METHOD_DATA=
    (DIRECTORY=+disk_group_name/path_within_the_ASM_diskgroup)))
```

> **Note:**
>
> Starting with Oracle Database release 19c, the `ENCRYPTION_WALLET_LOCATION` parameter is deprecated. Instead, use the `WALLET_ROOT` static initialization parameter and the `TDE_CONFIGURATION` dynamic initialization parameter to configure the wallet location.

Related Topics

- Configuring Keystores for Automatic Storage Management
  You can store a software keystore on an Automatic Storage Management (ASM) disk group.

10

Frequently Asked Questions About Transparent Data Encryption

Users frequently have questions about transparency and performance issues with Transparent Data Encryption.

- Transparency Questions About Transparent Data Encryption
  Transparent Data encryption handles transparency in data in a variety of ways.
- Performance Questions About Transparent Data Encryption
  There are several performance issues to consider when using Transparent Data Encryption.

Transparency Questions About Transparent Data Encryption

Transparent Data encryption handles transparency in data in a variety of ways.

Security auditors occasionally ask detailed questions about the encryption used by Transparent Data Encryption (TDE). They request information about TDE keys, algorithms, lengths, and keystores and then directly compare to requirements of regulations such as PCI-DSS. This topic contains important details about TDE encryption and key management. This information is current as of Oracle Database 12*c* (12.1.0.2). It is intended to help TDE customers respond to auditor questions quickly and accurately.

1. **Is Transparent Data Encryption compatible with my application software?**

   Transparent Data Encryption is compatible with applications by default because it does not alter the inbound SQL statements or the outbound SQL query results. Oracle executes internal testing and validation of certain Oracle and third-party application software to capture helpful deployment tips or scripts, and to evaluate performance profiles.

   Be aware of the difference between Transparent Data Encryption and the `DBMS_CRYPTO` PL/SQL package. This package is intended for different customer use cases. It is an API and toolkit solution and as such, it is non-transparent.

2. **Is Transparent Data Encryption compatible with other Oracle Database tools and technologies that I am using?**

   One of the chief benefits of Transparent Data Encryption is its integration with frequently used Oracle Database tools and technologies such as high-availability clusters, storage compression, backup compression, data movement, database backup and restore, and database replication. Specific Oracle technologies that are integrated directly with Transparent Data Encryption include Oracle Real Application Clusters (Oracle RAC), Oracle Recovery Manager (RMAN), Oracle Data Guard, Advanced Compression, Oracle Data Pump, and Oracle GoldenGate, among others. Transparent Data Encryption also has special points of integration with Oracle Exadata that fully use unique features of Oracle-engineered systems.

   Transparent Data Encryption also works easily with security features of the Oracle Database. With Transparent Data Encryption, privilege grants, roles,

Oracle Database Vault realms, Virtual Private Database policies, and Oracle Label Security labels remain in effect. You can use these and other security features in tandem with Transparent Data Encryption encryption.

3. **Are there any known Transparent Data Encryption limitations or incompatibilities?**

   - **TDE column encryption:** TDE column encryption encrypts and decrypts data transparently when data passes through the SQL layer. Some features of Oracle will bypass the SQL layer, and hence cannot benefit from TDE column encryption. The following are known database features that TDE column encryption does not support, and their relevant software version numbers:
     - Materialized View Logs (not supported prior to Oracle Database 11g release 2)
     - Synchronous and asynchronous change data capture for data warehousing (CDC)
     - Transportable Tablespaces
     - LOBs

     Note that Secure Files were introduced in Oracle Database 11g release 1, so it is not supported with TDE column encryption prior to that release
   - **TDE tablespace encryption:** TDE tablespace encryption encrypts all content that is stored in the tablespace at the block level in storage, and it generally does not conflict with other database features. TDE tablespace encryption does not have any of the limitations that TDE column encryption has. However, you can use full transportable tablespaces (TTS) with Oracle Data Pump compression and encryption when going from a TDE-encrypted source to a TDE-encrypted destination. You must have an Oracle Database release 12*c* or later database instance available so that you can use its key export or keystore (wallet) merge capabilities to get the correct master encryption key to the destination database host without having to overwrite the original Oracle wallet file. This process is subject to the standard TTS limitations, and you must remember to check for compatible endianness.

4. **What types of keys and algorithms does TDE use?**

   TDE relies on two distinct sets of encryption keys. The first set of encryption keys are TDE tablespace encryption keys, which are used to transparently encrypt and decrypt stored data. DEKs are generated automatically by the database, stored internally in the database in encrypted form, and managed mostly behind the scenes. One place where end-users interact with DEKs is when selecting the encryption algorithm and key length that TDE will use, which can be 3DES168, AES128, AES192, or AES256. This selection is made independently for each table containing encrypted columns and for each encrypted tablespace. You may also hear DEKs referred to as table keys (column encryption) or tablespace keys (tablespace encryption). The table keys are used in cipher block chaining (CBC) operating mode, and the tablespace keys are used in cipher feedback (CFB) operating mode.

   The second set of encryption keys consists of current and historical key encryption keys (KEK), also known as master encryption keys. The master encryption keys are generated automatically by the database, used automatically to encrypt and decrypt DEKs as needed, and stored externally in a protected keystore. Users may interact with the current master encryption key by periodically rekeying it, modifying certain key attributes, and so forth. Typically, the keystore for master encryption keys is either an Oracle wallet (out-of-the-box solution) or Oracle Key

Vault (a specialized key management product). Although the database uses only one TDE master key at a time, all rekeyed master encryption keys are retained in the keystore for long-term recovery of encrypted data backups.Master encryption keys always are AES256. They encrypt and decrypt DEKs using CBC operating mode. For both DEKs and master encryption keys, the underlying key material is not directly exposed. End-users see only attributes of keys necessary to manage TDE.

5. **How are Oracle keystores containing master encryption keys protected?**

   There are three different types of keystore to consider when you use an Oracle wallet as the keystore for master encryption keys: password-based, auto-login, and local auto-login. All of these keystore externalize master encryption keys, so they are separate from TDE-encrypted data. Oracle recommends that you place wallet files in local or network directories that are protected by tight file permissions and other security measures.

   The password-based wallet is an encrypted key storage file (`ewallet.p12`) that follows the PKCS #12 standard. It is encrypted by a password-derived key according to the PKCS #5 standard. A human user must enter a command containing the password for the database to open the wallet, decrypt its contents, and gain access to keys. The password-based wallet is the default keystore for TDE master keys. In the past, it was encrypted using the 3DES168 encryption algorithm and CBC operating mode. The `orapki` command `convert wallet` enables you to convert password-based wallets to AES256 and CBC operating mode. *Oracle Database Security Guide* provides details about using `orapki` to convert wallets.

   Auto-login wallets (`cwallet.sso`) optionally are derived from standard password-based wallets for special cases where automatic startup of the database is required with no human interaction to enter a wallet password. When using auto-login wallet, the master password-based wallet must be preserved because it is needed to rekey the master encryption key. In addition to the best practice of storing auto-login wallet in a local or network directory that is protected by tight file permissions, the file contents are scrambled by the database using a proprietary method for added security. A slight variation on the auto-login wallet called local auto-login wallet has similar behavior. One notable difference with local auto-login wallet is that its contents are scrambled using additional factors taken from the host machine where the file was created. This renders the local auto-login wallet unusable on other host machines. Details of the host factors and scrambling technique are proprietary.

6. **What is Oracle Key Vault and how does it manage TDE master keys?**

   Oracle Key Vault centrally manages TDE master keys, Oracle wallets, Java keystores, and more. It helps you to take control of proliferating keys and key storage files. It includes optimizations specifically for TDE and other components of the Oracle stack. For more information about using Oracle Key Vault with TDE, see the product pages on Oracle Technology Network and *Oracle Key Vault Administrator's Guide* .

Performance Questions About Transparent Data Encryption

There are several performance issues to consider when using Transparent Data Encryption.

1. **What is the typical performance overhead from Transparent Data Encryption?**

   There are many different variables involved in the creation of an accurate Transparent Data Encryption performance test. The results can vary depending on the test environment, test case or workload, measurement metrics or methods, and so on. Oracle cannot guarantee a specific performance overhead percentage that can apply in all possible scenarios. In practice, the performance tests by many Transparent Data Encryption customers are often in the low single digits as a percentage, but that is not universally the case. Customer examples that cite 1 percent and 2 percent overhead respectively are published on Oracle Technology Network in the following URL:

   `http://streaming.oracle.com/ebn/podcasts/media/12740910_ColumbiaU_120312.mp3`

   If possible, use Oracle Real Application Testing (Oracle RAT) to capture a real production workload and then replay it against Transparent Data Encryption to get a true indication of the performance overhead that the you can expect within your environment.

   See also:

   - Performance and Storage Overhead of Transparent Data Encryption
   - *Oracle Database Testing Guide* for more information about the Oracle Real Application Testing option

2. **How can I tune for optimal Transparent Data Encryption performance?**

   - **TDE column encryption:**
     - Limit the crypto processing by only encrypting the subset of columns that are strictly required to be protected. In addition, turn off the optional integrity checking feature.
     - After you apply column encryption, rebuild the column indexes.
   - **TDE tablespace encryption:** TDE tablespace encryption improves performance by caching unencrypted data in memory in the SGA buffer cache. This feature reduces the number of crypto operations that must be performed when users run `SELECT` queries, which draw from the SGA instead of drawing from disk. (Drawing from disk forces the database to perform decrypt operations.) Ensure that the size of the SGA buffer cache is large enough to take full advantage of this performance optimization.

     Another major performance boost comes from using hardware and software that supports CPU-based cryptographic acceleration available in Intel AES-NI and Oracle SPARC T4/T5. To take advantage of this feature, you must be running a recent version of the database, have a recent version of the operating system installed, and be using hardware that includes crypto acceleration circuitry within its CPUs/cores.

     Database compression further speeds up Transparent Data Encryption performance because the crypto processing occurs on data that already is compressed, resulting in less total data to encrypt and decrypt.
   - **In general:**
     - Ensure that you have applied the latest patches, which you can download from My Oracle Support at

       `https://support.oracle.com`

  - When you specify an encryption algorithm, remember that AES is slightly faster than 3DES. Use AES128 where possible. Be aware that the performance benefit is small.
  - Use Exadata, which includes additional performance benefits. For more information about Oracle Exadata, see *Oracle Database Testing Guide*.

3. **Are there specific issues that may slow down TDE performance, and if so, how do I avoid them?**

   TDE tablespace performance is slower if the database cannot use CPU-based hardware acceleration on the host machine due to factors such as older hardware, an older database version, or an older operating system.

   Note the following with regard to specific database workloads:

   - **Encrypting the whole data set at once (for example, while doing "Bulk Data Load" into an Oracle data warehouse):** Lower crypto performance has been observed during bulk load of new data into the database or data warehouse. New data cannot be cached in SGA, so TDE tablespace encryption performance optimizations are bypassed. Hence, Transparent Data Encryption has no bonus performance benefits in this type of operation.

     Follow these guidelines:

     - Ensure that the database is running on servers with CPU-based cryptographic acceleration. This accelerates not only decrypt operations, but also encrypt operations as well (for loading new data). Take the crypto processing out of band by pre-encrypting the data set and then using Transportable Tablespaces (TTS) to load into the database. Try to parallelize this procedure where possible. This requires the database instance to copy the required TDE key to the keystore on the destination database. The procedure may not be feasible when there is a fixed time window for encryption and loading, and these must be done serially.
     - Consider using TDE column encryption. Encrypt only the handful of sensitive regulated columns instead of encrypting an entire tablespace.

   - **Decrypting an entire data set at once (for example, while performing a full table scan by reading directly from disk, with no reading from SGA):**

     Lower crypto performance is observed when running full table scan queries where data is read directly from storage. Certain performance optimizations of TDE tablespace encryption are bypassed (no caching). Hence, Transparent Data Encryption has no bonus performance benefits in this type of operation.

     Follow these guidelines:

     - Ensure that the database is running on servers with CPU-based cryptographic acceleration.
     - Retest the full table scan queries with a larger SGA size to measure performance when data is read from cache. Try setting the Oracle event number 10949 to disable direct path read.
     - Partition the database so that less data is scanned by full table scan operations. Production databases often use partitioning for this kind of scenario (that is, to limit the total amount of data scanned).
     - Consider using TDE column encryption. Encrypt only the handful of sensitive regulated columns instead of encrypting an entire tablespace.

Part II
Using Oracle Data Redaction

Part II describes how to use Oracle Data Redaction.

- Introduction to Oracle Data Redaction
  Oracle Data Redaction is the ability to redact sensitive data in real time.
- Oracle Data Redaction Features and Capabilities
  Oracle Data Redaction provides a variety of ways to redact different types of data.
- Configuring Oracle Data Redaction Policies
  An Oracle Data Redaction policy defines how to redact data in a column based on the table column type and the type of redaction you want to use.
- Managing Oracle Data Redaction Policies in Oracle Enterprise Manager
  Oracle Enterprise Manager Cloud Control (Cloud Control) can manage Oracle Data Redaction policies and formats.
- Using Oracle Data Redaction with Oracle Database Features
  Oracle Data Redaction can be used with other Oracle features, but some Oracle features may have restrictions with regard to Oracle Data Redaction.
- Security Considerations for Oracle Data Redaction
  Oracle provides guidelines for using Oracle Data Redaction.

11

Introduction to Oracle Data Redaction

Oracle Data Redaction is the ability to redact sensitive data in real time.

- What Is Oracle Data Redaction?
  Oracle Data Redaction enables you to mask (redact) data that is returned from queries issued by applications.
- When to Use Oracle Data Redaction
  Use Oracle Data Redaction when you must disguise sensitive data that your applications and application users must access.
- Benefits of Using Oracle Data Redaction
  Oracle Data Redaction provides several benefits when you use it to protect your data.
- Target Use Cases for Oracle Data Redaction
  Oracle Data Redaction fulfils common use case scenarios.

What Is Oracle Data Redaction?

Oracle Data Redaction enables you to mask (redact) data that is returned from queries issued by applications.

You can redact column data by using one of the following methods:

- **Full redaction.** You redact all of the contents of the column data. The redacted value returned to the querying application user depends on the data type of the column. For example, columns of the `NUMBER` data type are redacted with a zero (`0`), and character data types are redacted with a single space.
- **Partial redaction.** You redact a portion of the column data. For example, you can redact a Social Security number with asterisks (*), except for the last 4 digits.
- **Regular expressions.** You can use regular expressions to look for patterns of data to redact. For example, you can use regular expressions to redact email addresses, which can have varying character lengths. It is designed for use with character data only.
- **Random redaction.** The redacted data presented to the querying application user appears as randomly generated values each time it is displayed, depending on the data type of the column.
- **No redaction.** The None redaction type option enables you to test the internal operation of your redaction policies, with no effect on the results of queries against tables with policies defined on them. You can use this option to test the redaction policy definitions before applying them to a production environment.

Oracle Database applies the redaction at runtime, when users access the data (that is, at query-execution time). This solution works well in a production system. During the time that the data is being redacted, all of the data processing is performed normally, and the back-end referential integrity constraints are preserved.

Data redaction can help you to comply with industry regulations such as Payment Card Industry Data Security Standard (PCI DSS) and the Sarbanes-Oxley Act.

Related Topics

- *Oracle Database Security Guide*

When to Use Oracle Data Redaction

Use Oracle Data Redaction when you must disguise sensitive data that your applications and application users must access.

Data Redaction enables you to easily disguise the data using several different redaction styles.

Oracle Data Redaction is ideal for situations in which you must redact specific characters out of the result set of queries of Personally Identifiable Information (PII) returned to certain application users. For example, you may want to present a U.S. Social Security number that ends with the numbers 4320 as `***-**-4320`.

Oracle Data Redaction is particularly suited for call center applications and other applications that are read-only. Take care when using Oracle Data Redaction with applications that perform updates back to the database, because redacted data can be written back to this database.

Benefits of Using Oracle Data Redaction

Oracle Data Redaction provides several benefits when you use it to protect your data.

These benefits are as follows:

- You have different styles of redaction from which to choose.
- Because the data is redacted at runtime, Data Redaction is well suited to environments in which data is constantly changing.
- You can create the Data Redaction policies in one central location and easily manage them from there.
- The Data Redaction policies enable you to create a wide variety of function conditions based on `SYS_CONTEXT` values, which can be used at runtime to decide when the Data Redaction policies will apply to the results of the application user's query.

Target Use Cases for Oracle Data Redaction

Oracle Data Redaction fulfils common use case scenarios.

- Oracle Data Redaction Use with Database Applications
  Oracle Data Redaction protects sensitive data that is displayed in database applications.
- Oracle Data Redaction with Ad Hoc Database Queries Considerations
  You may encounter situations where it is convenient to redact sensitive data for ad hoc queries that are performed by database users.

Oracle Data Redaction Use with Database Applications

Oracle Data Redaction protects sensitive data that is displayed in database applications.

Data Redaction is transparent to application users because it preserves the original data type and (optionally) the formatting. It is highly transparent to the database because the data remains the same in buffers, caches, and storage—only being changed at the last minute just before SQL query results are returned to the caller. The redaction is enforced consistently across all of the applications that use the same underlying database. You can specify which application users should see only redacted data by checking application user information that is passed into the database through the `SYS_CONTEXT` function; you can redact data based on attributes of the current database or application user; and you can implement multiple logical conditions within a given redaction policy. In addition, Data Redaction is implemented in a way that minimizes performance overhead. These characteristics make Oracle Data Redaction particularly well suited for usage by a range of applications, analytics tools, reporting tools, and monitoring tools that share common production databases. Although its primary target is redaction of production data for applications, Oracle Data Redaction also can be used in combination with Oracle Enterprise Manager Data Masking and Subsetting Pack for protecting sensitive data in testing and development environments.

See Also:

- *Oracle Data Masking and Subsetting Guide* for more information about data masking and subsetting
- Oracle Data Redaction and Data Masking and Subsetting Pack

Oracle Data Redaction with Ad Hoc Database Queries Considerations

You may encounter situations where it is convenient to redact sensitive data for ad hoc queries that are performed by database users.

For example, in the course of supporting a production application, a user may need to run ad hoc database queries to troubleshoot and fix an urgent problem with the application. This is different from the application-based scenarios described in Oracle Data Redaction Use with Database Applications, which typically generate a bounded set of SQL queries, use defined database accounts, and have fixed privileges.

Even though Oracle Data Redaction is not designed to prevent data exposure to database users who run ad hoc queries directly against the database, it can provide an additional layer to reduce the chances of accidental data exposure. Because such users may have rights to change data, alter the database schema, and circumvent the SQL query interface entirely, it is possible for a malicious user to bypass Data Redaction policies in certain circumstances.

Remember that the Oracle Database security tools are designed to be used together to improve overall security. By deploying one or more of these tools as a complement to Oracle Data Redaction, you can securely increase your overall security posture.

Related Topics

- Oracle Data Redaction General Security Guidelines
  It is important to understand general security guidelines for using Oracle Data Redaction.

12

Oracle Data Redaction Features and Capabilities

Oracle Data Redaction provides a variety of ways to redact different types of data.

- Full Data Redaction to Redact All Data
  Full data redaction redacts the entire contents of the specified table or view column.
- Partial Data Redaction to Redact Sections of Data
  In partial data redaction, you redact portions of the displayed output.
- Regular Expressions to Redact Patterns of Data
  Regular expressions redact specific data within a column data value, based on a pattern search.
- Redaction Using Null Values
  You can create an Oracle Data Redaction policy that redacts column data by replacing it with null values.
- Random Data Redaction to Generate Random Values
  In random data redaction, the entire value is redacted by replacing it with a random value.
- Comparison of Full, Partial, and Random Redaction Based on Data Types
  The full, partial, and random data redaction styles affect the Oracle built-in, ANSI, user-defined, and Oracle supplied types in different ways.
- No Redaction for Testing Purposes
  You can create a Data Redaction policy that does not perform redaction.
- Central Management of Named Data Redaction Policy Expressions
  You can create a library of named policy expressions that can be used in the columns of multiple tables and views.

Full Data Redaction to Redact All Data

Full data redaction redacts the entire contents of the specified table or view column.

By default the output is displayed as follows:

- **Character data types:** The output text is a single space.
- **Number data types:** The output text is a zero (`0`).
- **Date-time data types:** The output text is set to the first day of January, 2001, which appears as `01-JAN-01`.

Full redaction is the default and is used whenever a Data Redaction policy specifies the column but omits the `function_type` parameter setting. When you run the `DBMS_REDACT.ADD_POLICY` procedure, to set the `function_type` parameter setting for full redaction, you enter the following setting:

```
function_type    => DBMS_REDACT.FULL
```

You can use the `DBMS_REDACT.UPDATE_FULL_REDACTION_VALUES` procedure to change the full redaction output to different values.

Related Topics

- Syntax for Creating a Full Redaction Policy
  The `DBMS_REDACT.ADD_POLICY` procedure enables you to create a full redaction policy.

Partial Data Redaction to Redact Sections of Data

In partial data redaction, you redact portions of the displayed output.

You can set the position within the actual data at which to begin the redaction, the number of characters to redact starting from that position, and the redaction character to use. This type of redaction is useful for situations where you want it to be obvious to the person viewing the data that it was redacted in some way. Typically, you use this type of redaction for credit cards or ID numbers.

Be aware that partial data redaction requires that your data width remain fixed. If you want to redact columns containing string values of variable length, then you must use regular expressions.

To specify partial redaction, you must set the `DBMS_REDACT.ADD_POLICY` procedure `function_type` parameter to `DBMS_REDACT.PARTIAL` and use the `function_parameters` parameter to define the partial redaction behavior.

The displayed output for partial data redaction can be as follows:

- **Character data types:** When partially redacted, a Social Security number (represented as a hyphenated string within a character data type) with value `987-65-4320` could be redacted so that it is displayed as shown in the following examples. The code on the right specifies how to redact the character data: it specifies the expected input format of the actual data, the format to use for the display of the redacted output, the start position at which to begin the redaction, the character to use for the redaction, and how many characters to redact. The first example uses a predefined format (in previous releases called a shortcut) for character data type Social Security numbers, and the second example replaces the first five numbers with an asterisk (`*`) while preserving the hyphens (`-`) in between the numbers.

  ```
  XXX-XX-4320    function_parameters => DBMS_REDACT.REDACT_US_SSN_F5,

  ***-**-4320    function_parameters => 'VVVFVVFVVVV,VVV-VV-VVVV,*,1,5',
  ```

- **Number data types:** The partially redacted `NUMBER` data type Social Security number `987654328` could appear as follows. Both redact the first five digits. The first example uses a predefined format that is designed for Social Security numbers in the `NUMBER` data type, and the second replaces the first five numbers with the number `9`, starting from the first digit.

  ```
  XXXXX4328    function_parameters => DBMS_REDACT.REDACT_NUM_US_SSN_F5,

  999994328    function_parameters => '9,1,5',
  ```

- **Date-time data types:** Partially redacted datetime values can appear simply as different dates. For example, the date `29-AUG-11 10.20.50.000000 AM` could

appear as follows. In the first example, the day of the month is redacted to `02` (using the setting `d02`) and in the second example, the month is redacted to `DEC` (using `m12`). The uppercase values show the actual month (`M`), year (`Y`), hour (`H`), minute (`M`), and second (`S`).

```
02-AUG-11 10.20.50.000000 AM   function_parameters  =>  'Md02YHMS',

29-DEC-11 10.20.50.000000 AM   function_parameters  =>  'm12DYHMS',
```

Related Topics

- Regular Expressions to Redact Patterns of Data
  Regular expressions redact specific data within a column data value, based on a pattern search.
- Syntax for Creating a Partial Redaction Policy
  The `DBMS_REDACT.ADD_POLICY` statement enables you to create policies that redact specific parts of the data returned to the application.

Regular Expressions to Redact Patterns of Data

Regular expressions redact specific data within a column data value, based on a pattern search.

For example, you can redact the user name of email addresses, so that only the domain shows (for example, replacing `hpreston` in the email address `hpreston@example.com` with `[redacted]` so that it appears as `[redacted]@example.com`). To perform the redaction, set the `DBMS_REDACT.ADD_POLICY` procedure `function_type` parameter to either `DBMS_REDACT.REGEXP` or `DBMS_REDACT.REGEXP_WIDTH`, and then use the following parameters to build the regular expression:

- A string search pattern (that is, the values to search for), such as:

  ```
  regexp_pattern         => '(.+)@(.+\.[A-Za-z]{2,4})'
  ```

  This setting looks for a pattern of the following form:

  one_or_more_characters@one_or_more_characters.2-4_characters_in_range_A-Z_or_a-z

- A replacement string, which replaces the value matched by the `regexp_pattern` setting. The replacement string can include back references to sub-expressions of the main regular expression pattern. The following example replaces the data before the `@` symbol (from the `regexp_pattern` setting) with the text `[redacted]`. The `\2` setting refers to the second match group, which is `(.+\.[A-Za-z]{2,4})` from the `regexp_pattern` setting.

  ```
  regexp_replace_string  => '[redacted]@\2'
  ```

- The starting position for the string search string, such as the first character of the data, such as:

  ```
  regexp_position        => DBMS_REDACT.RE_BEGINNING
  ```

- The kind of search and replace operation to perform, such as the first occurrence, every fifth occurrence, or all of the occurrences, such as:

  ```
  regexp_occurrence      => DBMS_REDACT.RE_ALL
  ```

- The default matching behavior for the search and replace operation, such as whether the search is case-sensitive (`i` sets it to be not case-sensitive):

  ```
  regexp_match_parameter => 'i
  ```

In addition to the default parameters, you can use a set of predefined formats that enable you to use commonly used regular expressions for telephone numbers, email addresses, and credit card numbers.

Related Topics

- Syntax for Creating a Regular Expression-Based Redaction Policy
  The `regexp_*` parameters of the `DBMS_REDACT.ADD_POLICY` procedure can create a regular expression-based redaction policy.

Redaction Using Null Values

You can create an Oracle Data Redaction policy that redacts column data by replacing it with null values.

This feature enables you to use the `DBMS_REDACT.NULLIFY` function hide all of the sensitive data in a table or view column and replace it with null values. You can set this function by using the `function_type` parameter of the `DBMS_REDACT.ADD_POLICY` or `DBMS_REDACT.ALTER_POLICY` procedure.

For example:

```
function_type         => DBMS_REDACT.NULLIFY
```

Related Topics

- Creating a DBMS_REDACT.NULLIFY Redaction Policy
  You can create Oracle Data Redaction policies that return null values for the displayed value of the table or view column.

Random Data Redaction to Generate Random Values

In random data redaction, the entire value is redacted by replacing it with a random value.

The redacted values displayed in the result set of the query change randomly each time application users run the query.

This type of redaction is useful in cases where you do not want it to be obvious that the data was redacted. It works especially well for number and datetime data types, where it is difficult to distinguish between random and real data.

The displayed output for random values changes based on the data type of the redacted column, as follows:

- **Character data types:** The random output is a mixture of characters (for example, `HTU[G{\pjkEWcK`). It behaves differently for the `CHAR` and `VARCHAR2` data types, as follows:
  - **CHAR data type:** The redacted output is always in the same character set as the character set of the column. The byte length of the redacted output is always the same as the column definition length (that is, the column length that was provided at the time of table creation). For example, if the column is

`CHAR(20)`, then a string of 20 random characters is provided in the redacted output of the user's query.

  - **VARCHAR2 data type:** For random redaction of a `VARCHAR` data type, the redacted output is always in the same character set as the character set of the column. The length of the redacted output is limited based on the length of the actual data in the column. No characters in excess of the length of the actual data are displayed. For example, if the column is `VARCHAR2(20)` and the row being redacted contains actual data with a length of 12, then a string of 12 random characters (not 20) is provided in the redacted output of the user's query for that row.

- **Number data types:** Each actual number value is redacted by replacing it with a random, non-negative number modulo the absolute value of the actual data. This redaction results in random numbers that do not exceed the precision of the actual data. For example, the number `987654321` can be redacted by replacing it with any of the numbers `12345678`, `13579`, `0`, or `987654320`, but not by replacing it with any of the numbers `987654321`, `99987654321`, or `-1`. The number `-123` could be redacted by replacing it with the numbers `122`, `0`, or `83`, but not by replacing it with any of the numbers `123`, `1123`, or `-2`.

  The only exception to the above is when the actual value is an integer between -1 and 9. In this case, the actual data is redacted by replacing it with a random, non-negative integer modulo ten (10).

- **Date-time data types:** When values of the date data type are redacted using random Data Redaction, Oracle Database displays them with random dates that are always different from those of the actual data.

The setting for using random redaction is as follows:

```
function_type    => DBMS_REDACT.RANDOM
```

Related Topics

- Syntax for Creating a Random Redaction Policy
  A random redaction policy presents the redacted data to the querying application user as randomly generated values, based on the column data type.

Comparison of Full, Partial, and Random Redaction Based on Data Types

The full, partial, and random data redaction styles affect the Oracle built-in, ANSI, user-defined, and Oracle supplied types in different ways.

- Oracle Built-in Data Types Redaction Capabilities
  Oracle Data Redaction handles the Oracle built-in data types depending on the type of Data Redaction policies that are used.
- ANSI Data Types Redaction Capabilities
  Oracle Data Redaction converts ANSI data types in specific ways, depending on the type of redaction that the Data Redaction policy has.
- Built-in and ANSI Data Types Full Redaction Capabilities
  For full redaction, the default redacted value depends on whether the data type is Oracle built-in or ANSI.

- User-Defined Data Types or Oracle Supplied Types Redaction Capabilities
  Several data types or types are not supported by Oracle Data Redaction.

Oracle Built-in Data Types Redaction Capabilities

Oracle Data Redaction handles the Oracle built-in data types depending on the type of Data Redaction policies that are used.

Table 12-1 describes the Oracle Data Redaction support for Oracle built-in data types.

Table 12-1 Redaction Support for Oracle Built-in Data Types

Column Data Type	Full	Partial	Regexp	Random
Character[1]	Yes	Yes	Yes	Yes
Number[2]	Yes	Yes	No	Yes
Raw[3]	No	No	No	No
Date-time[4]	Yes	Yes	No	Yes
Interval[5]	No	No	No	No
BFILE	No	No	No	No
BLOB	Yes	No	No	No
CLOB	Yes	No	Yes	No
NCLOB	Yes	No	Yes	No
ROWID	No	No	No	No
UROWID	No	No	No	No

1 Includes CHAR, VARCHAR2 (including long VARCHAR2, for example, VARCHAR2(20000)), NCHAR, NVARCHAR2
2 Includes NUMBER, FLOAT, BINARY_FLOAT, BINARY_DOUBLE
3 Includes LONG RAW, RAW
4 Includes DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE, TIMESTAMP WITH LOCAL TIME ZONE
5 Includes INTERVAL YEAR TO MONTH, INTERVAL DAY TO SECOND

ANSI Data Types Redaction Capabilities

Oracle Data Redaction converts ANSI data types in specific ways, depending on the type of redaction that the Data Redaction policy has.

Table 12-2 compares how the full, partial, and random redaction styles work for ANSI data types, with regard to how they are converted and their support status.

Table 12-2 Redaction Support for the ANSI Data Types

Data Type	How Converted	Full Redaction	Partial Redaction	Regexp	NULL Redaction	Random Redaction
CHARACTER(n), CHAR(n)	Converted to CHAR(*n*)	Yes	Yes	Yes	Yes	Yes

Table 12-2 (Cont.) Redaction Support for the ANSI Data Types

Data Type	How Converted	Full Redaction	Partial Redaction	Regexp	NULL Redaction	Random Redaction
`CHARACTER VARYING(n)`, `CHAR VARYING(n)`	Converted to `VARCHAR2(n)`	Yes	Yes	Yes	Yes	Yes
`NATIONAL CHARACTER(n)`, `NATIONAL CHAR(n)`, `NCHAR(n)`	Converted to `NCHAR(n)`	Yes	Yes	Yes	Yes	Yes
`NATIONAL CHARACTER VARYING(n)`, `NATIONAL CHAR VARYING(n)`, `NCHAR VARYING(n)`	Converted to `NVARCHAR2(n)`	Yes	Yes	Yes	Yes	Yes
`NUMERIC[(p,s)]` `DECIMAL[(p,s)]`	Converted to `NUMBER(p,s)`	Yes	Yes	Yes	Yes	Yes
`INTEGER`, `INT`, `SMALLINT`	Converted to `NUMBER(38)`	Yes	Yes	Yes	Yes	Yes
`FLOAT`, `DOUBLE PRECISION`	Converted to `FLOAT(126)`	Yes	Yes	Yes	Yes	Yes
`REAL`	Converted to `FLOAT(63)`	Yes	Yes	Yes	Yes	Yes
`GRAPHIC`, `LONG VARGRAPHIC`, `VARGRAPHIC`, `TIME`	No conversion	No	No	No	No	No

Built-in and ANSI Data Types Full Redaction Capabilities

For full redaction, the default redacted value depends on whether the data type is Oracle built-in or ANSI.

ANSI Data Types Redaction Capabilities shows the default settings for both Oracle built-in and ANSI data type columns that use full redaction.

Table 12-3 Default Settings and Categories for Columns That Use Full Redaction

Data Type	Default Redacted Value	Data Type Category
CHARACTER	Single space (" ")	Oracle built-in
CHARACTER(n), CHAR(n)	Single space (" ")	ANSI
CHARACTER VARYING(n), CHAR VARYING(n)	Single space (" ")	ANSI
NATIONAL CHARACTER(n), NATIONAL CHAR(n), NCHAR(n)	Single space (" ")	ANSI
NATIONAL CHARACTER VARYING(n), NATIONAL CHAR VARYING(n), NCHAR VARYING(n)	Single space (" ")	ANSI
NUMBER	Zero (0)	Oracle built-in
NUMERIC[(*p*,*s*)] DECIMAL[(*p*,*s*)]	Zero (0)	Oracle built-in
INTEGER, INT, SMALLINT	Zero (0)	ANSI
FLOAT, DOUBLE PRECISION	Zero (0)	ANSI
REAL	Zero (0)	ANSI
DATE-TIME	01-01-01 or 01-01-01 01:00:00	Oracle built-in
BLOB	Oracle's raw representation of [redacted] [1]	Oracle built-in
CLOB	[redacted]	Oracle built-in
NCLOB	[redacted]	Oracle built-in

1 If you have changed the character set, then you may need to invoke the DBMS_REDACT.UPDATE_FULL_REDACTION_VALUES procedure to set the value to the raw representation in the new character set, as follows:

```
DECLARE
 new_red_blob BLOB;
BEGIN
 DBMS_LOB.CREATETEMPORARY(new_red_blob, TRUE);
 DBMS_LOB.WRITE(new_red_blob, 10, 1,
UTL_RAW.CAST_TO_RAW('[redacted]'));
 dbms_redact.update_full_redaction_values(
  blob_val      => new_red_blob);
DBMS_LOB.FREETEMPORARY(new_red_blob);
```

```
END;
/
```

After you run this procedure, restart the database.

See also Altering the Default Full Data Redaction Value for more information about using the `DBMS_REDACT.UPDATE_FULL_REDACTION_VALUES` procedure.

User-Defined Data Types or Oracle Supplied Types Redaction Capabilities

Several data types or types are not supported by Oracle Data Redaction.

Table 12-4 compares how the full, partial, regular expression, and random redaction styles work for user-defined and Oracle-supplied types.

Table 12-4 Redaction Support for the User-Defined Data Types or Oracle-Supplied Types

Data Type or Type	Full Redaction	Partial Redaction	Regexp	NULL Redaction	Random Redaction
User-defined data types	No	No	No	No	No
Oracle supplied types: Any types, XML types, Oracle Spatial types	No	No	No	No	No

No Redaction for Testing Purposes

You can create a Data Redaction policy that does not perform redaction.

This is useful for cases in which you have a redacted base table, yet you want a specific application user to have a view that always shows the actual data. You can create a new view of the redacted table and then define a Data Redaction policy for this view. The policy still exists on the base table, but no redaction is performed when the application queries using the view as long as the `DBMS_REDACT.NONE` `function_type` setting was used to create a policy on the view.

Central Management of Named Data Redaction Policy Expressions

You can create a library of named policy expressions that can be used in the columns of multiple tables and views.

By having named policy expressions, you can centrally manage all of the policy expressions within a database.

When you modify the policy expression, the change is reflected in all table columns that use the expression. The policy expression takes precedence over the `expression` setting in the Data Redaction policy. To create the policy expression, you must use the `DBMS_REDACT.CREATE_POLICY_EXPRESSION` procedure, and to apply the policy expression to a column, you use `DBMS_REDACT.APPLY_POLICY_EXPR_TO_COL`. This feature provides flexibility to redact different columns in a table or view, based on different runtime conditions.

For example, consider a use case that involves a customer care application. A customer calls the customer care center to request a return on a recent purchase. A level 1 support representative of the call center must first verify the order ID, customer name, and customer address before initiating the return. During the process, there is no need for the level 1 support representative to view the customer's credit card number. So, the credit card column is redacted when the support representative queries the customer details in the call center application. When the return is initiated, a sales representative from the return department may need to view the credit card number to process the return. However, there is no need for the sales representative to view the expiration date of the credit card. So, when the sales representative queries the customer details in the same application, the credit card number is visible but the expiration date is redacted.

In this use case, different columns in the customer details table must be redacted in different ways, based on who the logged in user is. Oracle Data Redaction simplifies the implementation of this use case by using named Data Redaction policy expressions. This type of policy expression enables you to define and associate different policy expressions on different columns in the same table or view. Moreover, you can centrally manage named policy expressions within a database. Any updates that you make to a named policy expression are immediately propagated to all of the associated table or view columns.

Related Topics

- Creating and Managing Multiple Named Policy Expressions
  A named, centrally managed Oracle Data Redaction policy expression can be used in multiple redaction policies and applied to multiple tables or views.

13

Configuring Oracle Data Redaction Policies

An Oracle Data Redaction policy defines how to redact data in a column based on the table column type and the type of redaction you want to use.

- About Oracle Data Redaction Policies
  An Oracle Data Redaction policy defines the conditions in which redaction must occur for a table or view.
- Who Can Create Oracle Data Redaction Policies?
  Because data redaction involves the protection of highly sensitive data, only trusted users should create Oracle Data Redaction policies.
- Planning an Oracle Data Redaction Policy
  Before you create a Oracle Data Redaction policy, you should plan the data redaction policy that best suits your site's needs.
- General Syntax of the DBMS_REDACT.ADD_POLICY Procedure
  To create a Data Redaction policy, you must use the `DBMS_REDACT.ADD_POLICY` procedure.
- Using Expressions to Define Conditions for Data Redaction Policies
  The `expression` parameter in the `DBMS_REDACT.ADD_POLICY` procedure sets the conditions to which the policy applies.
- Creating and Managing Multiple Named Policy Expressions
  A named, centrally managed Oracle Data Redaction policy expression can be used in multiple redaction policies and applied to multiple tables or views.
- Creating a Full Redaction Policy and Altering the Full Redaction Value
  You can create a full redaction policy to redact all contents in a data column, and optionally, you can alter the default full redaction value.
- Creating a DBMS_REDACT.NULLIFY Redaction Policy
  You can create Oracle Data Redaction policies that return null values for the displayed value of the table or view column.
- Creating a Partial Redaction Policy
  In partial data redaction, you can redact portions of data, and for different kinds of data types.
- Creating a Regular Expression-Based Redaction Policy
  A regular expression-based redaction policy enables you to redact data based on a search-and-replace model.
- Creating a Random Redaction Policy
  A random redaction policy presents redacted data as randomly generated values, such as `Ukjsl32[[]]]s`.
- Creating a Policy That Uses No Redaction
  You can create policies that use no redaction at all, for when you want to test the policy in a development environment.
- Exemption of Users from Oracle Data Redaction Policies
  You can exempt users from having Oracle Data Redaction policies applied to the data they access.

- Altering an Oracle Data Redaction Policy
  The `DBMS_REDACT.ALTER_POLICY` procedure enables you to modify Oracle Data Redaction policies.
- Redacting Multiple Columns
  You can redact more than one column in a Data Redaction policy.
- Disabling and Enabling an Oracle Data Redaction Policy
  You can disable and then reenable Oracle Data Redactions policies as necessary.
- Dropping an Oracle Data Redaction Policy
  The `DBMS_REDACT.DROP_POLICY` procedure drops Oracle Data Redaction policies.
- Tutorial: SQL Expressions to Build Reports with Redacted Values
  SQL expressions can be used to build reports based on columns that have Oracle Data Redaction policies defined on them.
- Oracle Data Redaction Policy Data Dictionary Views
  Oracle Database provides data dictionary views that list information about Data Redaction policies.

About Oracle Data Redaction Policies

An Oracle Data Redaction policy defines the conditions in which redaction must occur for a table or view.

A Data Redaction policy has the following characteristics:

- The Data Redaction policy defines the following: What kind of redaction to perform, how the redaction should occur, and when the redaction takes place. Oracle Database performs the redaction at execution time, just before the data is returned to the application.
- A Data Redaction policy can fully redact values, partially redact values, or randomly redact values. In addition, you can define a Data Redaction policy to not redact any data at all, for when you want to test your policies in a test environment.
- A Data Redaction policy can be defined with a policy expression which allows for different application users to be presented with either redacted data or actual data, based on whether the policy expression returns `TRUE` or `FALSE`. Redaction takes place when the boolean result of evaluating the policy expression is `TRUE`. For security reasons, the functions and operators that can be used in the policy expression are limited to `SYS_CONTEXT` and a few others. User-created functions are not allowed. Policy expressions can make use of the `SYS_SESSION_ROLES` namespace with the `SYS_CONTEXT` function to check for enabled roles.
- Different Data Redaction policy expressions can be created and then applied individually for different columns within the same table or view.

Table 13-1 lists the procedures in the `DBMS_REDACT` package.

Table 13-1 DBMS_REDACT Procedures

Procedure	Description
`DBMS_REDACT.ADD_POLICY`	Adds a Data Redaction policy to a table or view
`DBMS_REDACT.ALTER_POLICY`	Modifies a Data Redaction policy

Table 13-1 (Cont.) DBMS_REDACT Procedures

Procedure	Description
DBMS_REDACT.APPLY_POLICY_EXPR_TO_COL	Applies a Data Redaction policy expression to a table or view column
DBMS_REDACT.CREATE_POLICY_EXPRESSION	Creates a Data Redaction policy expression
DBMS_REDACT.DISABLE_POLICY	Disables a Data Redaction policy
DBMS_REDACT.DROP_POLICY	Drops a Data Redaction policy
DBMS_REDACT.DROP_POLICY_EXPRESSION	Drops a Data Redaction policy expression
DBMS_REDACT.ENABLE_POLICY	Enables a Data Redaction policy
DBMS_REDACT.UPDATE_FULL_REDACTION_VALUES	Globally updates the full redaction value for a given data type. You must restart the database instance before the updated values can be used.
DBMS_REDACT.UPDATE_POLICY_EXPRESSION	Updates a Data Redaction policy expression

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the DBMS_REDACT PL/SQL package
- Managing Oracle Data Redaction Policies in Oracle Enterprise Manager for information about using Oracle Enterprise Manager Cloud Control to create and manage Oracle Data Redaction policies and formats

Who Can Create Oracle Data Redaction Policies?

Because data redaction involves the protection of highly sensitive data, only trusted users should create Oracle Data Redaction policies.

To create redaction policies, you must have the EXECUTE privilege on the DBMS_REDACT PL/SQL package. To find the privileges that a user has been granted, you can query the DBA_SYS_PRIVS data dictionary view.

You do not need any privileges to access the underlying tables or views that will be protected by the policy.

Planning an Oracle Data Redaction Policy

Before you create a Oracle Data Redaction policy, you should plan the data redaction policy that best suits your site's needs.

1. Ensure that you have been granted the EXECUTE privilege on the DBMS_REDACT PL/SQL package.

2. Determine the data type of the table or view column that you want to redact.
3. Determine if the base object to which you want to add the Data Redaction policy has dependent objects. If it does have dependent objects, then these objects will become invalid when the Data Redaction policy is added to the base object, and these objects will be recompiled automatically when they are used.

   Alternatively, you can proactively recompile them yourself by using an `ALTER ... COMPILE` statement. Be aware that invalidating dependent objects (by adding a Data Redaction policy on their base object) and causing them to need to be recompiled can decrease performance in the overall system. Oracle recommends that you only add a Data Redaction policy to an object that has dependent objects during off-peak hours or during a scheduled downtime.
4. Ensure that this column is not used in an Oracle Virtual Private Database (VPD) row filtering condition. That is, it must not be part of the VPD predicate generated by the VPD policy function.
5. Decide on the type of redaction that you want to perform: full, random, partial, regular expressions, or none.
6. Decide which users to apply the Data Redaction policy to.
7. Based on this information, create the Data Redaction policy by using the `DBMS_REDACT.ADD_POLICY` procedure.
8. Configure the policy to have additional columns to be redacted.

After you create the Data Redaction policy, it is automatically enabled and ready to redact data.

Related Topics

- Redacting Multiple Columns
  You can redact more than one column in a Data Redaction policy.

General Syntax of the DBMS_REDACT.ADD_POLICY Procedure

To create a Data Redaction policy, you must use the `DBMS_REDACT.ADD_POLICY` procedure.

The complete syntax for the `DBMS_REDACT.ADD_POLICY` procedure is as follows:

```
DBMS_REDACT.ADD_POLICY (
 object_schema              IN VARCHAR2 := NULL,
 object_name                IN VARCHAR2 := NULL,
 policy_name                IN VARCHAR2,
 policy_description         IN VARCHAR2 := NULL,
 column_name                IN VARCHAR2 := NULL,
 column_description         IN VARCHAR2 := NULL,
 function_type              IN BINARY_INTEGER := DBMS_REDACT.FULL,
 function_parameters        IN VARCHAR2 := NULL,
 expression                 IN VARCHAR2,
 enable                     IN BOOLEAN := TRUE,
 regexp_pattern             IN VARCHAR2 := NULL,
 regexp_replace_string      IN VARCHAR2 := NULL,
 regexp_position            IN BINARY_INTEGER :=1,
 regexp_occurrence          IN BINARY_INTEGER :=0,
 regexp_match_parameter     IN VARCHAR2 := NULL);
```

In this specification:

- `object_schema`: Specifies the schema of the object on which the Data Redaction policy will be applied. If you omit this setting (or enter `NULL`), then Oracle Database uses the current user's name. Be aware that the meaning of "current user" here can change, depending on where you invoke the `DBMS_REDACT.ADD_POLICY` procedure.

  For example, suppose user `mpike` grants user `fbrown` the `EXECUTE` privilege on a definer's rights PL/SQL package called `mpike.protect_data` in `mpike`'s schema. From within this package, `mpike` has coded a procedure called `protect_cust_data`, which invokes the `DBMS_REDACT.ADD_POLICY` procedure. User `mpike` has set the `object_schema` parameter to `NULL`.

  When `fbrown` invokes the `protect_cust_data` procedure in the `mpike.protect_data` package, Oracle Database attempts to define the Data Redaction policy around the object `cust_data` in the `mpike` schema, not the `cust_data` object in the schema that belongs to `fbrown`.
- `object_name`: Specifies the name of the table or view to which the Data Redaction policy applies.
- `policy_name`: Specifies the name of the policy to be created. Ensure that this name is unique in the database instance. You can find a list of existing Data Redaction policies by querying the `POLICY_NAME` column of the `REDACTION_POLICIES` data dictionary view.
- `policy_description`: Specifies a brief description of the purpose of the policy.
- `column_name`: Specifies the column whose data you want to redact. Note the following:
  - **You can apply the Data Redaction policy to multiple columns.** If you want to apply the Data Redaction policy to multiple columns, then after you use `DBMS_REDACT.ADD_POLICY` to create the policy, run the `DBMS_REDACT.ALTER_POLICY` procedure as many times as necessary to add each of the remaining required columns to the policy. See Altering an Oracle Data Redaction Policy.
  - **Only one policy can be defined on a table or view.** You can, however, create a new view on the table, and by defining a second redaction policy on this new view, you can choose to redact the columns in a different way when a query is issued against this new view. When deciding how to redact a given column, Oracle Database uses the policy of the earliest view in a view chain.
  - **If you do not specify a column (for example, by entering NULL), then no columns are redacted by the policy.** This enables you to create your policies so that they are in place, and then later on, you can add the column specification when you are ready.
  - **Do not use a column that is currently used in an Oracle Virtual Private Database (VPD) row filtering condition.** In other words, the column should not be part of the VPD predicate generated by the VPD policy function. (See Oracle Data Redaction and Oracle Virtual Private Database for more information about using Data Redaction with VPD.)
  - **You cannot define a Data Redaction policy on a virtual column.** In addition, you cannot define a Data Redaction policy on a column that is involved in the SQL expression of any virtual column.

- `column_description`: Specifies a brief description of the column that you are redacting.
- `function_type`: Specifies a function that sets the type of redaction. See the following sections for more information:
  - Syntax for Creating a Full Redaction Policy
  - Syntax for Creating a Partial Redaction Policy
  - Syntax for Creating a Regular Expression-Based Redaction Policy
  - Syntax for Creating a Random Redaction Policy
  - Syntax for Creating a Policy with No Redaction

  If you omit the `function_type` parameter, then the default redaction `function_type` setting is `DBMS_REDACT.FULL`.
- `function_parameters`: Specifies how the column redaction should appear for partial redaction. See Syntax for Creating a Partial Redaction Policy.
- `expression`: Specifies a Boolean SQL expression to determine how the policy is applied. Redaction takes place only if this policy expression evaluates to `TRUE`. See Using Expressions to Define Conditions for Data Redaction Policies.
- `enable`: When set to `TRUE`, enables the policy upon creation. When set to `FALSE`, it creates the policy as a disabled policy. The default is `TRUE`. After you create the policy, you can disable or enable it. See the following sections:
  - Disabling an Oracle Data Redaction Policy
  - Enabling an Oracle Data Redaction Policy
- `regexp_pattern`, `regexp_replace_string`, `regexp_position`, `regexp_position`, `regexp_occurrence`, `regexp_match_parameter`: Enable you to use regular expressions to redact data, either fully or partially. If the `regexp_pattern` does not match anything in the actual data, then full redaction will take place, so be careful when specifying the `regexp_pattern`. Ensure that all of the values in the column conform to the semantics of the regular expression you are using. See Syntax for Creating a Regular Expression-Based Redaction Policy for more information.

Using Expressions to Define Conditions for Data Redaction Policies

The `expression` parameter in the `DBMS_REDACT.ADD_POLICY` procedure sets the conditions to which the policy applies.

- About Using Expressions in Data Redaction Policies
  The `DBMS_REDACT.ADD_POLICY` and `DBMS_REDACT.ALTER_POLICY` `expression` parameter defines a Boolean expression that must evaluate to `TRUE` to enable a redaction.
- Supported Functions for Data Redaction Expressions
  You can create expressions that use functions to return specific types of data, such as `SYS_CONTEXT` namespaces.
- Applying the Redaction Policy Based on User Environment
  You can apply a Data Redaction policy based on the user's environment, such as the session user name or a client identifier.

- Applying the Redaction Policy Based on Database Roles
  You can apply a Data Redaction policy based on a database role, such as the DBA role.
- Applying the Redaction Policy Based on Oracle Label Security Label Dominance
  You can set a condition on which to apply a Data Redaction policy based on the dominance of Oracle Label Security labels.
- Applying the Redaction Policy Based on Application Express Session States
  You can apply a Data Redaction policy based on an Oracle Application Express (APEX) session state.
- Applying the Redaction Policy to All Users
  You can apply the policy irrespective of the context to any user, with no filtering.

About Using Expressions in Data Redaction Policies

The DBMS_REDACT.ADD_POLICY and DBMS_REDACT.ALTER_POLICY expression parameter defines a Boolean expression that must evaluate to TRUE to enable a redaction.

The expression that is defined in the expression parameter is the default expression for the Oracle Data Redaction policy. If you apply a named policy expression for the columns that will be redacted by the Data Redaction policy, then the named policy expression takes precedence over the expression defined in the Data Redaction policy.

You can create expressions that make use of other Oracle Database features. For example, you can create expressions that are based on a user's environment (using the SYS_CONTEXT and XS_SYS_CONTEXT functions), character string functions, the Oracle Label Security label dominance functions, or Oracle Application Express functions.

Follow these guidelines when you write the expression:

- Use only the following operators: AND, OR, IN, NOT IN, =, !=, <>, <, >, >=, <=
- Because the expression must evaluate to TRUE for redaction, be careful when making comparisons with NULL. Remember that in SQL the value NULL is undefined, so comparisons with NULL tend to return FALSE.
- Do not use user-created functions in the expression parameter; this is not permitted.
- Remember that for user SYS and users who have the EXEMPT REDACTION POLICY privilege, all of the Data Redaction policies are bypassed, so the results of their queries are not redacted. See the following topics for more information about users who are exempted from Data Redaction policies:
  - Exemption of Users from Oracle Data Redaction Policies
  - Oracle Data Pump Security Model for Oracle Data Redaction

Supported Functions for Data Redaction Expressions

You can create expressions that use functions to return specific types of data, such as SYS_CONTEXT namespaces.

- Expressions Using Namespace Functions
  You can use the SYS_CONTEXT and XS_SYS_CONTEXT namespace functions in Data Redaction expressions.

- Expressions Using the SUBSTR Function
  You can use the SUBSTR function, which returns portion (such as characters 1–3) of the character string specified, in Data Redaction expressions. The first parameter must be a constant string or a call to the SYS_CONTEXT function or the XS_SYS_CONTEXT function.
- Expressions Using Length of Character String Functions
  You can use the following functions, which return the length of character strings, in Data Redaction expressions. Oracle Database also checks that the arguments to each of these operators is either a constant string or a call to the SYS_CONTEXT or XS_SYS_CONTEXT function.
- Expressions Using Oracle Application Express Functions
  You can use Oracle Application Express functions in Data Redaction expressions.
- Expressions Using Oracle Label Security Functions
  You can use Oracle Label Security functions with Data Redaction expressions.

Expressions Using Namespace Functions

You can use the SYS_CONTEXT and XS_SYS_CONTEXT namespace functions in Data Redaction expressions.

Table 13-2 Expressions Using Namespace Functions

Namespace Function	Description
SYS_CONTEXT	Returns the value associated with a namespace. The following namespace functions are valid: • USERENV (default namespace), which includes values such as SESSION_USER and CLIENT_IDENTIFIER. • SYS_SESSION_ROLES, which contains attributes for each role • XS$SESSION, which contains attributes for the user session. • User-defined namespaces, but these must exist in the DBA_CONTEXT catalog view before the policy expression is created.
XS_SYS_CONTEXT	Similar to SYS_CONTEXT but designed for an Oracle Real Application Security environment. XS_SYS_CONTEXT supports the same namespaces that SYS_CONTEXT supports.

See Also:

- *Oracle Database SQL Language Reference* for more information about SYS_CONTEXT
- *Oracle Database Real Application Security Administrator's and Developer's Guide* for more information about XS_SYS_CONTEXT

Expressions Using the SUBSTR Function

You can use the SUBSTR function, which returns portion (such as characters 1–3) of the character string specified, in Data Redaction expressions. The first parameter must be a constant string or a call to the SYS_CONTEXT function or the XS_SYS_CONTEXT function.

Table 13-3 Expressions Using SUBSTR String Functions

SUBSTR String Function	Description
SUBSTR	Returns a portion of the input char value, beginning at *character position*, *substring_length* characters long. SUBSTR calculates length using characters as defined by the input character set.
SUBSTRB	Returns the specified portion of the input value in bytes
SUBSTRC	Returns the specified portion of the input value in Unicode complete characters
SUBSTR2	Returns the specified portion of the input value in UCS2 code points
SUBSTR4	Returns the specified portion of the input value in UCS4 code points

> **See Also:**
>
> *Oracle Database SQL Language Reference* for more information about the SUBSTR functions

Expressions Using Length of Character String Functions

You can use the following functions, which return the length of character strings, in Data Redaction expressions. Oracle Database also checks that the arguments to each of these operators is either a constant string or a call to the SYS_CONTEXT or XS_SYS_CONTEXT function.

Table 13-4 Expressions Using Character String Functions

Character String Function	Description
LENGTH	Returns the length of the input char value. LENGTH calculates length using characters as defined by the input character set.
LENGTHB	Returns the length of the input value in bytes
LENGTHC	Returns the length of the input value in Unicode complete characters
LENGTH2	Returns the length of the input value in UCS2 code points
LENGTH4	Returns the length of the input value in UCS4 code points

> **See Also:**
>
> *Oracle Database SQL Language Reference* for more information about the `LENGTH` functions

Expressions Using Oracle Application Express Functions

You can use Oracle Application Express functions in Data Redaction expressions.

Table 13-5 Oracle Application Express Functions

Oracle Application Express Function	Description
`V`	Returns the session state for an item. It is a wrapper for the `APEX_UTIL.GET_SESSION_STATE` function
`NV`	Returns the numeric value for a numeric item. It is a wrapper for the `APEX_UTIL.GET_NUMERIC_SESSION_STATE` function

> **See Also:**
>
> *Oracle Application Express API Reference* for more information about the Oracle Application Express functions

Expressions Using Oracle Label Security Functions

You can use Oracle Label Security functions with Data Redaction expressions.

For the functions in the bold font, Oracle Data Redaction checks that their parameters are either constants or calls to only one of the `SA_UTL.NUMERIC_LABEL`, `CHAR_TO_LABEL`, and `SA_SESSION.LABEL` functions, and that the arguments to those functions are constant.

Table 13-6 Oracle Label Security Functions

Oracle Label Security Function	Description
`LBACSYS.OLS_LABEL_DOMINATES`	Checks if the session label of an Oracle Label Security policy dominates or is equal to another OLS label
`DOMINATES`	Checks if one OLS label is dominant to a second OLS label. Deprecated in Oracle Database 12c release 1 (12.1); use the `OLS_DOMINATES` or `OLS_DOM` function instead.
`OLS_DOMINATES`	Checks if one OLS label is dominant to a second OLS label

Table 13-6 (Cont.) Oracle Label Security Functions

Oracle Label Security Function	Description
OLS_DOM	Checks if one OLS label is dominant to a second OLS label
DOM	Checks if one OLS label is dominant to a second OLS label
OLS_STRICTLY_DOMINATES	Checks if one OLS label is dominant to a second OLS label and is not equal to it
STRICTLY_DOMINATES	Checks if one OLS label is dominant to a second OLS label and is not equal to it
S_DOM	Checks if one OLS label is dominant to a second OLS label and is not equal to it. Deprecated in Oracle Database 12*c* release 1 (12.1); use the `OLS_DOMINATES` or `OLS_DOM` function instead.
SA_UTL.DOMINATES	Checks if one OLS label dominates a second OLS label or if the session label for a given OLS policy dominates an OLS label
`SA_UTL.CHECK_READ`	Checks if a user can read a policy-protected row
`SA_UTL.NUMERIC_LABEL`	Returns the current session OLS label
`CHAR_TO_LABEL`	Converts a character string to an OLS label tag
`SA_SESSION.LABEL`	Returns the label that is associated with the specified OLS policy

Related Topics

- *Oracle Label Security Administrator's Guide*

Applying the Redaction Policy Based on User Environment

You can apply a Data Redaction policy based on the user's environment, such as the session user name or a client identifier.

- Use the `USERENV` namespace of the `SYS_CONTEXT` function in the `DBMS_REDACT.ADD_POLICY` `expression` parameter to apply the policy based on a user's environment.

For example, to apply the policy only to the session user name `psmith`:

```
expression  => 'SYS_CONTEXT(''USERENV'',''SESSION_USER'') = ''PSMITH'''
```

See Also:

Oracle Database SQL Language Reference for information about more namespaces that you can use with the `SYS_CONTEXT` function

Applying the Redaction Policy Based on Database Roles

You can apply a Data Redaction policy based on a database role, such as the DBA role.

- Use the SYS_SESSION_ROLES namespace in the SYS_CONTEXT function to apply the policy based on a user role.

  This namespace contains attributes for each role. The value of the attribute is TRUE if the specified role is enabled for the querying application user; the value is FALSE if the role is not enabled.

For example, suppose you wanted only supervisors to be allowed to see the actual data. The following example shows how to use the DBMS_REDACT.ADD_POLICY expression parameter to set the policy to show the actual data to any application user who has the supervisor role enabled, but redact the data for all of the other application users.

```
expression  => 'SYS_CONTEXT(''SYS_SESSION_ROLES'',''SUPERVISOR'') = ''FALSE'''
```

Applying the Redaction Policy Based on Oracle Label Security Label Dominance

You can set a condition on which to apply a Data Redaction policy based on the dominance of Oracle Label Security labels.

- Use the public standalone function OLS_LABEL_DOMINATES to check the dominance of a session label. This function returns 1 (TRUE) if the session label of the specified policy_name value dominates or is equal to the label that is specified by the label parameter; otherwise, it returns 0 (FALSE).

For example, to apply a Data Redaction policy only in cases where the session label for the policy hr_ols_pol does not dominate nor is equal to label hs:

```
expression  => 'OLS_LABEL_DOMINATES (''hr_ols_pol'',''hs'') = 0'
```

Applying the Redaction Policy Based on Application Express Session States

You can apply a Data Redaction policy based on an Oracle Application Express (APEX) session state.

- Use either of the following public Application Express APIs in the DBMS_REDACT.ADD_POLICY expression parameter to apply the policy on an Oracle Application Express session state:
  - V, which is a synonym for the APEX_UTIL.GET_SESSION_STATE function
  - NV, which is a synonym for the APEX_UTIL.GET_NUMERIC_SESSION_STATE function

For example, to set the DBMS_REDACT.ADD_POLICY expression parameter if you wanted redaction to take place when the application item called G_JOB has the value CLERK:

```
expression => 'V(''APP_USER'') != ''mavis@example.com'' or V(''APP_USER'') is
null'
```

You can, for example, use these functions to redact data based on a job or a privilege role that is stored in a session state in an APEX application.

If you want redaction to take place when the querying user is *not* within the context of an APEX application (when the query is issued from outside the APEX framework, for example directly through SQL*Plus), then use an `IS NULL` clause as follows. This policy expression causes actual data to be shown to user `mavis` only when her query comes from within an APEX application. Otherwise, the query result is redacted.

> **See Also:**
>
> *Oracle Application Express API Reference*

Applying the Redaction Policy to All Users

You can apply the policy irrespective of the context to any user, with no filtering.

However, be aware that user `SYS` and users who have the `EXEMPT REDACTION POLICY` privilege are always except from Oracle Data Redaction policies.

- To apply the policy to users who are not `SYS` or have been granted the `EXEMPT REDACTION POLICY` privilege, write the `DBMS_REDACT.ADD_POLICY expression` parameter to evaluate to `TRUE`.

For example:

```
expression  => '1=1'
```

> **See Also:**
>
> Exemption of Users from Oracle Data Redaction Policies

Creating and Managing Multiple Named Policy Expressions

A named, centrally managed Oracle Data Redaction policy expression can be used in multiple redaction policies and applied to multiple tables or views.

- About Data Redaction Policy Expressions to Define Conditions
  A named Oracle Data Redaction policy expression is designed to work as an alternative to the policy expression that is used in existing Data Redaction policies.
- Creating and Applying a Named Data Redaction Policy Expression
  The `DBMS_REDACT.CREATE_POLICY_EXPRESSION` and `DBMS_REDACT.APPLY_POLICY_EXPR_TO_COL` enable you to create and apply a named Data Redaction policy expression.
- Updating a Named Data Redaction Policy Expression
  You can use the `DBMS_REDACT.UPDATE_POLICY_EXPRESSION` procedure to update a Data Redaction policy expression. The update takes place immediately and is reflected in all columns that use the policy expression.

- Dropping a Named Data Redaction Expression Policy
  You can use the `DBMS_REDACT.DROP_POLICY_EXPRESSION` procedure to drop a Data Redaction expression policy.
- Tutorial: Creating and Sharing a Named Data Redaction Policy Expression
  This tutorial shows how to create an Oracle Data Redaction policy expression, apply it to multiple tables, and centrally manage the policy expression.

About Data Redaction Policy Expressions to Define Conditions

A named Oracle Data Redaction policy expression is designed to work as an alternative to the policy expression that is used in existing Data Redaction policies.

A named policy expression enables you to redact data based on runtime conditions. This type of policy can only affect whether or not redaction takes place on columns of the table or view on which the redaction policy is defined. By default, a Data Redaction policy expression applies to all the columns of a table or view. Alternatively, you can choose to create and associate a policy expression for individual columns of a table or view. These column level expressions are called as named policy expressions; in other words, a policy expression with a name. A named policy expressions has the following properties:

You can use Data Redaction policy expressions in the following ways.:

- A single Data Redaction policy expression can be shared by more than one Data Redaction policy by applying it to columns that are a part of separate Data Redaction policies.
- Each named policy expression can be associated with multiple columns of the same or different tables or views.
- Each named policy expression can be associated with columns within the same or different Data Redaction policies.
- The named policy expression overrides the default policy expression of the associated columns. The default policy expression still applies to redaction columns that have no named policy expressions applied to them.
- Any updates made to a named policy expression apply to all of the column associations of the expression.
- You cannot associate multiple named policy expressions for the same column.
- In a multitenant environment, you cannot associate named policy expressions with columns in a different pluggable database (PDB).

The column to which you apply a named policy expression must already be redacted by a Data Redaction policy. After the named policy expression is applied, the result of its evaluation takes precedence over that of the default policy expression when deciding whether or not to redact the column. When you modify a named policy expression, the changes are applied to all the tables and views that use it. In a multitenant environment, as with Data Redaction policies, a named policy expression is valid only in the PDB in which it was created, and can only be applied to columns of objects within the PDB in which it was created.

Table 13-7 describes the `DBMS_REDACT` PL/SQL procedures that you can use to create and manage named policy expressions. To find information about policy expressions, query the `REDACTION_EXPRESSIONS` data dictionary view.

Table 13-7 DBMS_REDACT Policy Expression Procedures

Procedure	Description
DBMS_REDACT.CREATE_POLICY_EXPRESSION	Creates a Data Redaction policy expression
DBMS_REDACT.UPDATE_POLICY_EXPRESSION	Updates a Data Redaction policy expression
DBMS_REDACT.APPLY_POLICY_EXPR_TO_COL	Applies a Data Redaction policy expression to a table or a view column
DBMS_REDACT.DROP_POLICY_EXPRESSION	Drops a Data Redaction policy expression

Related Topics

- Managing Named Data Redaction Policy Expressions Using Enterprise Manager
  You can manage Oracle Data Redaction policy expressions in Enterprise Manager Cloud Control.

Creating and Applying a Named Data Redaction Policy Expression

The DBMS_REDACT.CREATE_POLICY_EXPRESSION and DBMS_REDACT.APPLY_POLICY_EXPR_TO_COL enable you to create and apply a named Data Redaction policy expression.

1. Ensure that the COMPATIBLE initialization parameter is set to 12.2.0.0.

   To find the current setting, use the SHOW PARAMETER command.

2. To create the policy expression, run the DBMS_REDACT.CREATE_POLICY_EXPRESSION procedure.

   For example:

```
BEGIN
 DBMS_REDACT.CREATE_POLICY_EXPRESSION (
  policy_expression_name        => 'redact_pol',
  expression                    => '1=1',
  policy_expression_description => 'Determines whether the column will be
redacted');
END;
/
```

3. Run the DBMS_REDACT.APPLY_POLICY_EXPR_TO_COL procedure to apply the policy expression to a table or view column.

   For example, assume that you have already created a Data Redaction policy on the SALARY column of the HR.EMPLOYEES table, as follows:

```
BEGIN
  DBMS_REDACT.ADD_POLICY  (
   object_schema           => 'hr',
   object_name             => 'employees',
   policy_name             => 'overall_policy',
   expression              => '1=0');
END;
/
BEGIN
  DBMS_REDACT.ALTER_POLICY (
   object_schema           => 'hr',
   object_name             => 'employees' ,
```

```
   policy_name             => 'overall_policy',
   function_type           => DBMS_REDACT.FULL,
   action                  => DBMS_REDACT.ADD_COLUMN,
   column_name             => 'SALARY );
END;
/
```

Then you can apply the policy expression to the `SALARY` table as follows:

```
BEGIN
 DBMS_REDACT.APPLY_POLICY_EXPR_TO_COL (
   object_schema           => 'hr',
   object_name             => 'employees',
   column_name             => 'salary',
   policy_expression_name  => 'redact_pol');
END;
/
```

In this specification:

- `object_schema`: Specifies the schema of the object on which the policy expression will be used. If you omit this setting (or enter `NULL`), then Oracle Database uses the name of the current schema.
- `object_name`: Specifies the name of the table or view to be used for the policy expression.
- `column_name`: Specifies the column to which you want to apply the policy expression.
- `policy_expression_name`: Specifies the name of the policy expression.

After you create an Oracle Data Redaction policy expression, you can apply it to a column of a table or view which is part of an existing Data Redaction policy.

Updating a Named Data Redaction Policy Expression

You can use the `DBMS_REDACT.UPDATE_POLICY_EXPRESSION` procedure to update a Data Redaction policy expression. The update takes place immediately and is reflected in all columns that use the policy expression.

You can query the `REDACTION_EXPRESSIONS` data dictionary view to find existing Data Redaction policy expressions.

1. Ensure that the `COMPATIBLE` initialization parameter is set to `12.2.0.0`.

   To find the current setting, use the `SHOW PARAMETER` command.

2. Run the `DBMS_REDACT.UPDATE_POLICY_EXPRESSION` procedure to perform the update.

   For example:

```
BEGIN
 DBMS_REDACT.UPDATE_POLICY_EXPRESSION(
  policy_expression_name => 'redact_pol',
  expression             => '1=0');
END;
/
```

Dropping a Named Data Redaction Expression Policy

You can use the DBMS_REDACT.DROP_POLICY_EXPRESSION procedure to drop a Data Redaction expression policy.

You can query the REDACTION_EXPRESSIONS data dictionary view to find existing Data Redaction policy expressions.

1. Ensure that the COMPATIBLE initialization parameter is set to 12.2.0.0.

   To find the current setting, use the SHOW PARAMETER command.

2. Remove the named policy expression's association with any table or view column.

   You cannot drop a policy expression if it is associated with an existing table or view column. To remove a given column's association with a named policy expression (to revert to redacting that column based on the evaluation result of the default policy expression), you must set the policy_expression_name parameter of the DBMS_REDACT.APPLY_POLICY_EXPR_TO_COL procedure to NULL.

   For example:

```
BEGIN
 DBMS_REDACT.APPLY_POLICY_EXPR_TO_COL(
   object_schema          => 'hr',
   object_name            => 'employees',
   column_name            => 'salary',
   policy_expression_name =>  null);
END;
/
```

3. Run DBMS_REDACT.DROP_POLICY_EXPRESSION to drop the policy expression.

   For example:

```
BEGIN
 DBMS_REDACT.DROP_POLICY_EXPRESSION(
  policy_expression_name  => 'redact_pol');
END;
/
```

Tutorial: Creating and Sharing a Named Data Redaction Policy Expression

This tutorial shows how to create an Oracle Data Redaction policy expression, apply it to multiple tables, and centrally manage the policy expression.

- Step 1: Create Users for This Tutorial
  You must create two users for this tutorial: dr_admin, who will create the Oracle Data Redaction policies, and hr_clerk, who will test them.
- Step 2: Create an Oracle Data Redaction Policy
  User dr_admin is ready to create an Oracle Data Redaction policy to protect the HR.EMPLOYEES and HR.JOBS tables.
- Step 3: Test the Oracle Data Redaction Policy
  User hr_clerk is ready to query the tables that have redacted data.

- Step 4: Create and Apply a Policy Expression to the Redacted Table Columns
  Next, user dr_admin is ready to create a Data Redaction policy expression and apply it to two of the three redacted table columns.
- Step 5: Test the Data Redaction Policy Expression
  User hr_clerk is now ready to test the hr_redact_pol policy expression.
- Step 6: Modify the Data Redaction Policy Expression
  User dr_admin decides to modify the Data Redaction policy expression so that user HR will have access to the redacted data, not user hr_clerk.
- Step 7: Test the Modified Policy Expression
  Users HR and hr_clerk are ready to test the modified Data Redaction policy expression.
- Step 8: Remove the Components of This Tutorial
  If you do not need the components of this tutorial, then you can remove them.

Step 1: Create Users for This Tutorial

You must create two users for this tutorial: dr_admin, who will create the Oracle Data Redaction policies, and hr_clerk, who will test them.

Before you begin this tutorial, ensure that the COMPATIBLE initialization parameter is set to 12.2.0.0. You can check this setting by using the SHOW PARAMETER command.

1. Log in to SQL*Plus as user SYS with the SYSDBA administrative privilege.

```
sqlplus sys as sysdba
Enter password: password
```

2. In a multitenant environment, connect to the appropriate PDB.

   For example:

```
CONNECT SYS@my_pdb AS SYSDBA
Enter password: password
```

   To find the available PDBs, run the show pdbs command. To check the current PDB, run the SHOW CON_NAME command.

3. Create the dr_admin and hr_clerk user accounts.

```
GRANT CREATE SESSION TO dr_admin IDENTIFIED BY password;
GRANT CREATE SESSION TO hr_clerk IDENTIFIED BY password;
```

4. Grant the EXECUTE privilege to the dr_admin user.

```
GRANT EXECUTE ON DBMS_REDACT TO dr_admin;
```

5. Connect as user HR.

```
CONNECT HR --Or, for a PDB, CONNECT hr@my_pdb
Enter password: password
```

6. Grant hr_clerk the SELECT privilege on the EMPLOYEES and JOBS tables.

```
GRANT SELECT on EMPLOYEES to hr_clerk;
GRANT SELECT on JOBS to hr_clerk;
```

Step 2: Create an Oracle Data Redaction Policy

User dr_admin is ready to create an Oracle Data Redaction policy to protect the HR.EMPLOYEES and HR.JOBS tables.

1. Connect as user dr_admin.

```
CONNECT dr_admin --Or, for a PDB, CONNECT dr_admin@my_pdb
Enter password: password
```

2. Create the hr_emp_redact_comp_pol policy, which will perform full redaction of the HR.EMPLOYEES.SALARY column.

```
BEGIN
 DBMS_REDACT.ADD_POLICY(
   object_schema       => 'hr',
   object_name         => 'employees',
   column_name         => 'salary',
   policy_name         => 'hr_emp_redact_comp_pol',
   function_type       => DBMS_REDACT.FULL,
   expression          => '1=1');
END;
/
```

3. Alter the hr_redact_comp_pol policy to also redact the COMMISSION_PCT column of the HR.EMPLOYEES table.

```
BEGIN
 DBMS_REDACT.ALTER_POLICY(
   object_schema       => 'hr',
   object_name         => 'employees',
   policy_name         => 'hr_emp_redact_comp_pol',
   action              => DBMS_REDACT.ADD_COLUMN,
   column_name         => 'commission_pct',
   function_type       => DBMS_REDACT.FULL,
   expression          => '1=1');
END;
/
```

4. Create the hr_jobs_redact_comp_pol policy for the max_salary column of the HR.JOBS table.

```
BEGIN
 DBMS_REDACT.ADD_POLICY(
   object_schema       => 'hr',
   object_name         => 'jobs',
   column_name         => 'max_salary',
   policy_name         => 'hr_jobs_redact_comp_pol',
   function_type       => DBMS_REDACT.FULL,
   expression          => '1=1');
END;
/
```

At this stage, the data in the HR.EMPLOYEES.SALARY, HR.EMPLOYEES.COMMISSION_PCT, and HR.JOBS.MAX_SALARY columns are redacted.

Step 3: Test the Oracle Data Redaction Policy

User hr_clerk is ready to query the tables that have redacted data.

1. Connect as user hr_clerk.

```
CONNECT hr_clerk --Or, for a PDB, CONNECT hr_clerk@my_pdb
Enter password: password
```

2. Query the HR.EMPLOYEES table.

```
SELECT SALARY, COMMISSION_PCT FROM HR.EMPLOYEES WHERE SALARY > 15000;
```

The output should be as follows:

```
    SALARY COMMISSION_PCT
---------- --------------
         0
         0
         0
```

3. Query the HR.JOBS table.

```
SELECT MAX_SALARY FROM HR.JOBS WHERE MAX_SALARY > 15000;
```

The output should be as follows:

```
MAX_SALARY
----------
         0
         0
         0
         0
         0
```

Step 4: Create and Apply a Policy Expression to the Redacted Table Columns

Next, user dr_admin is ready to create a Data Redaction policy expression and apply it to two of the three redacted table columns.

This policy expression will enable user hr_clerk to view the redacted data.

1. Connect as user dr_admin.

```
CONNECT dr_admin --Or, for a PDB, CONNECT dr_admin@my_pdb
Enter password: password
```

2. Create the policy expression.

```
BEGIN
 DBMS_REDACT.CREATE_POLICY_EXPRESSION(
   policy_expression_name  => 'hr_redact_pol',
   expression              => 'SYS_CONTEXT(''USERENV'',''SESSION_USER'') !=
''HR_CLERK''');
END;
/
```

This expression returns FALSE for the hr_clerk user, which enables the hr_clerk user to view actual data in the HR.EMPLOYEES and HR.JOBS tables that are subject to the Data Redaction policies.

3. Apply the hr_redact_pol policy expression to the HR.EMPLOYEES.SALARY column.

```
BEGIN
 DBMS_REDACT.APPLY_POLICY_EXPR_TO_COL(
   object_schema          => 'hr',
   object_name            => 'employees',
   column_name            => 'salary',
   policy_expression_name => 'hr_redact_pol');
```

```
END;
/
```

4. Apply the `hr_redact_pol` policy expression to the `HR.JOBS.MAX_SALARY` column.

```
BEGIN
 DBMS_REDACT.APPLY_POLICY_EXPR_TO_COL(
   object_schema          => 'hr',
   object_name            => 'jobs',
   column_name            => 'max_salary',
   policy_expression_name => 'hr_redact_pol');
END;
/
```

User `hr_clerk` can view data in the `HR.EMPLOYEES.SALARY` and `HR.JOBS.MAX_SALARY`, but the data in the `HR.EMPLOYEES.COMMISSION_PCT` column will still be redacted for this user.

Step 5: Test the Data Redaction Policy Expression

User `hr_clerk` is now ready to test the `hr_redact_pol` policy expression.

1. Connect as user `hr_clerk`.

```
CONNECT hr_clerk --Or, for a PDB, CONNECT hr_clerk@my_pdb
Enter password: password
```

2. Query the `HR.EMPLOYEES` table.

```
SELECT SALARY, COMMISSION_PCT FROM HR.EMPLOYEES WHERE SALARY > 15000;
```

The output should be as follows:

```
    SALARY COMMISSION_PCT
---------- --------------
     24000
     17000
     17000
```

User `hr_clerk` now can view the `SALARY` column data, but still has not access to the `COMMISSION_PCT` column data.

3. Query the `HR.JOBS` table.

```
SELECT MAX_SALARY FROM HR.JOBS WHERE MAX_SALARY > 15000;
```

The output should be as follows:

```
MAX_SALARY
----------
     40000
     30000
     16000
     16000
     20080
```

User `hr_clerk` now can view the `MAX_SALARY` column data.

Step 6: Modify the Data Redaction Policy Expression

User `dr_admin` decides to modify the Data Redaction policy expression so that user `HR` will have access to the redacted data, not user `hr_clerk`.

1. Connect as user dr_admin.

```
CONNECT dr_admin --Or, for a PDB, CONNECT dr_admin@my_pdb
Enter password: password
```

2. Modify the hr_redact_pol policy as follows:

```
BEGIN
 DBMS_REDACT.UPDATE_POLICY_EXPRESSION(
  policy_expression_name => 'hr_redact_pol',
  expression             => 'SYS_CONTEXT(''USERENV'',''SESSION_USER'') !=
''HR''');
END;
/
```

Step 7: Test the Modified Policy Expression

Users HR and hr_clerk are ready to test the modified Data Redaction policy expression.

1. Connect as user HR.

```
CONNECT HR --Or, for a PDB, CONNECT HR@my_pdb
Enter password: password
```

2. Query the HR.EMPLOYEES table.

```
SELECT SALARY, COMMISSION_PCT FROM HR.EMPLOYEES WHERE SALARY > 15000;
```

The output should be as follows:

```
    SALARY COMMISSION_PCT
---------- --------------
     24000
     17000
     17000
```

User HR now has access to the redacted data. A query by HR on the HR.JOBS.MAX_SALARY column will produce similar results.

```
SELECT MAX_SALARY FROM HR.JOBS WHERE MAX_SALARY > 15000;

MAX_SALARY
----------
     40000
     30000
     16000
     16000
     20080
```

3. Connect as user hr_clerk.

```
CONNECT hr_clerk --Or, for a PDB, CONNECT hr_clerk@my_pdb
Enter password: password
```

4. Query the HR.EMPLOYEES and HR.JOBS tables and then observe the results.

```
SELECT SALARY, COMMISSION_PCT FROM HR.EMPLOYEES WHERE SALARY > 15000;

    SALARY COMMISSION_PCT
---------- --------------
         0
         0
         0
```

```
SELECT MAX_SALARY FROM HR.JOBS WHERE MAX_SALARY > 15000;

MAX_SALARY
----------
         0
         0
         0
         0
         0
```

Step 8: Remove the Components of This Tutorial

If you do not need the components of this tutorial, then you can remove them.

1. Connect as user `dr_admin`.

```
CONNECT dr_admin --Or, for a PDB, CONNECT dr_admin@my_pdb
Enter password: password
```

2. Modify the policy expression so that it is no longer associated with the table columns that are associated with the expression.

   To do so, you must set the `policy_expression_name` parameter to `NULL`.

```
BEGIN
 DBMS_REDACT.APPLY_POLICY_EXPR_TO_COL(
   object_schema          => 'hr',
   object_name            => 'employees',
   column_name            => 'salary',
   policy_expression_name =>  null);
END;
/

BEGIN
 DBMS_REDACT.APPLY_POLICY_EXPR_TO_COL(
   object_schema          => 'hr',
   object_name            => 'jobs',
   column_name            => 'max_salary',
   policy_expression_name =>  null);
END;
/
```

3. Drop the policy expressions.

```
BEGIN
 DBMS_REDACT.DROP_POLICY_EXPRESSION(
  policy_expression_name  => 'hr_redact_pol');
END;
/
```

4. Drop the `hr_emp_redact_comp_pol` and `hr_jobs_redact_comp_pol` Data Redaction policies.

```
BEGIN
  DBMS_REDACT.DROP_POLICY (
    object_schema  => 'hr',
    object_name    => 'employees',
    policy_name    => 'hr_emp_redact_comp_pol');
END;
/

BEGIN
```

```
  DBMS_REDACT.DROP_POLICY (
    object_schema  => 'hr',
    object_name    => 'jobs',
    policy_name    => 'hr_jobs_redact_comp_pol');
END;
/
```

5. Connect as the `SYSTEM` user or a user who has privileges to drop user accounts.

   For example:

   ```
   CONNECT SYSTEM
   Enter password: password
   ```

6. Drop the `dr_admin` and `hr_clerk` user accounts.

   ```
   DROP USER dr_admin;
   DROP USER hr_clerk;
   ```

Creating a Full Redaction Policy and Altering the Full Redaction Value

You can create a full redaction policy to redact all contents in a data column, and optionally, you can alter the default full redaction value.

- Creating a Full Redaction Policy
  A full data redaction policy redacts all the contents of a data column.
- Altering the Default Full Data Redaction Value
  The `DBMS_REDACT.UPDATE_FULL_REDACTION_VALUES` procedure alters the default full data redaction value.

Creating a Full Redaction Policy

A full data redaction policy redacts all the contents of a data column.

- About Creating Full Data Redaction Policies
  To set a redaction policy to redact all data in the column, you must set the `function_type` parameter to `DBMS_REDACT.FULL`.
- Syntax for Creating a Full Redaction Policy
  The `DBMS_REDACT.ADD_POLICY` procedure enables you to create a full redaction policy.
- Example: Full Redaction Policy
  You can use the `DBMS_REDACT.ADD_POLICY` PL/SQL procedure to create a full redaction policy.
- Example: Fully Redacted Character Values
  You can use the `DBMS_REDACT.ADD_POLICY` PL/SQL procedure to create a policy that fully redacts character values.

About Creating Full Data Redaction Policies

To set a redaction policy to redact all data in the column, you must set the `function_type` parameter to `DBMS_REDACT.FULL`.

By default, NUMBER data type columns are replaced with zero (0) and character data type columns are replaced with a single space ( ). You can modify this default by using the DBMS_REDACT.UPDATE_FULL_REDACTION_VALUES procedure.

Related Topics

- Altering the Default Full Data Redaction Value
  The DBMS_REDACT.UPDATE_FULL_REDACTION_VALUES procedure alters the default full data redaction value.

Syntax for Creating a Full Redaction Policy

The DBMS_REDACT.ADD_POLICY procedure enables you to create a full redaction policy.

The DBMS_REDACT.ADD_POLICY fields for creating a full data redaction policy are as follows:

```
DBMS_REDACT.ADD_POLICY (
   object_schema           IN VARCHAR2 := NULL,
   object_name             IN VARCHAR2,
   column_name             IN VARCHAR2 := NULL,
   policy_name             IN VARCHAR2,
   function_type           IN BINARY_INTEGER := NULL,
   expression              IN VARCHAR2,
   enable                  IN BOOLEAN := TRUE);
```

In this specification:

- object_schema, object_name, column_name, policy_name, expression, enable: See General Syntax of the DBMS_REDACT.ADD_POLICY Procedure.
- function_type: Specifies the function used to set the type of redaction. Enter DBMS_REDACT.FULL.

  If you omit the function_type parameter, then the default redaction function_type setting is DBMS_REDACT.FULL.

  Remember that the data type of the column determines which function_type settings that you are permitted to use. See Comparison of Full, Partial, and Random Redaction Based on Data Types.

Example: Full Redaction Policy

You can use the DBMS_REDACT.ADD_POLICY PL/SQL procedure to create a full redaction policy.

Example 13-1 shows how to use full redaction for all the values in the HR.EMPLOYEES table COMMISSION_PCT column. The expression parameter applies the policy to any user querying the table, except for users who have been granted the EXEMPT REDACTION POLICY system privilege.

Example 13-1 Full Data Redaction Policy

```
BEGIN
 DBMS_REDACT.ADD_POLICY(
   object_schema    => 'hr',
   object_name      => 'employees',
   column_name      => 'commission_pct',
   policy_name      => 'redact_com_pct',
   function_type    => DBMS_REDACT.FULL,
```

```
   expression       => '1=1');
END;
/
```

Query and redacted result:

```
SELECT COMMISSION_PCT FROM HR.EMPLOYEES;

COMMISSION_PCT
--------------
0
0
0
```

Related Topics

- Exemption of Users from Oracle Data Redaction Policies
  You can exempt users from having Oracle Data Redaction policies applied to the data they access.

Example: Fully Redacted Character Values

You can use the `DBMS_REDACT.ADD_POLICY` PL/SQL procedure to create a policy that fully redacts character values.

Example 13-2 shows how to redact fully the user IDs of the `user_id` column in the `mavis.cust_info` table. The `user_id` column is of the `VARCHAR2` data type. The output is a blank string. The `expression` setting enables users who have the `MGR` role to view the user IDs.

Example 13-2 Fully Redacted Character Values

```
BEGIN
 DBMS_REDACT.ADD_POLICY(
   object_schema   => 'mavis',
   object_name     => 'cust_info',
   column_name     => 'user_id',
   policy_name     => 'redact_cust_user_ids',
   function_type   => DBMS_REDACT.FULL,
   expression      => 'SYS_CONTEXT(''SYS_SESSION_ROLES'',''MGR'') = ''FALSE''');
END;
/
```

Query and redacted result:

```
SELECT user_id FROM mavis.cust_info;

USER_ID
------------
0
0
0
```

Altering the Default Full Data Redaction Value

The `DBMS_REDACT.UPDATE_FULL_REDACTION_VALUES` procedure alters the default full data redaction value.

- About Altering the Default Full Data Redaction Value
  You can alter the default displayed values for full Data Redaction polices.

- Syntax for the DBMS_REDACT.UPDATE_FULL_REDACTION_VALUES Procedure
  The DBMS_REDACT.UPDATE_FULL_REDACTION_VALUES procedure accommodates the standard supported Oracle Database data types.
- Modifying the Default Full Data Redaction Value
  To modify the default full data redaction value, use the DBMS_REDACT.UPDATE_FULL_REDACTION_VALUES procedure.

About Altering the Default Full Data Redaction Value

You can alter the default displayed values for full Data Redaction polices.

By default, 0 is the redacted value when Oracle Database performs full redaction (DBMS_REDACT.FULL) on a column of the NUMBER data type. If you want to change it to another value (for example, 7), then you can run the DBMS_REDACT.UPDATE_FULL_REDACTION_VALUES procedure to modify this value. The modification applies to all of the Data Redaction policies in the current database instance. After you modify a value, you must restart the database for it to take effect. You can find the current values by querying the REDACTION_VALUES_FOR_TYPE_FULL data dictionary view.

Be aware that this change affects all Data Redaction policies in the database that use full data redaction. Before you alter the default full data redaction value, examine the affect that this change would have on existing full Data Redaction policies.

Syntax for the DBMS_REDACT.UPDATE_FULL_REDACTION_VALUES Procedure

The DBMS_REDACT.UPDATE_FULL_REDACTION_VALUES procedure accommodates the standard supported Oracle Database data types.

The syntax is as follows:

```
DBMS_REDACT.UPDATE_FULL_REDACTION_VALUES (
 number_val       IN NUMBER                      NULL,
 binfloat_val     IN BINARY_FLOAT                NULL,
 bindouble_val    IN BINARY_DOUBLE               NULL,
 char_val         IN CHAR                        NULL,
 varchar_val      IN VARCHAR2                    NULL,
 nchar_val        IN NCHAR                       NULL,
 nvarchar_val     IN NVARCHAR2                   NULL,
 date_val         IN DATE                        NULL,
 ts_val           IN TIMESTAMP                   NULL,
 tswtz_val        IN TIMESTAMP WITH TIME ZONE    NULL,
 blob_val         IN BLOB                        NULL,
 clob_val         IN CLOB                        NULL,
 nclob_val        IN NCLOB                       NULL);
```

In this specification:

- number_val modifies the default value for columns of the NUMBER data type.
- binfloat_val modifies the default value for columns of the BINARY_FLOAT data type.
- bindouble_val modifies the default value for columns of the BINARY_DOUBLE data type.

- `char_val` modifies the default value for columns of the `CHAR` data type.
- `varchar_val` modifies the default value for columns of the `VARCHAR2` data type.
- `nchar_val` modifies the default value for columns of the `NCHAR` data type.
- `nvarchar_val` modifies the default value for columns of the `NVARCHAR2` data type.
- `date_val` modifies the default value for columns of the `DATE` data type.
- `ts_val` modifies the default value for columns of the `TIMESTAMP` data type.
- `tswtz_val` modifies the default value for columns of the `TIMESTAMP WITH TIME ZONE` data type.
- `blob_val` modifies the default value for columns of the `BLOB` data type.
- `clob_val` modifies the default value for columns of the `CLOB` data type.
- `nclob` modifies the default value for columns of the `NCLOB` data type.

Modifying the Default Full Data Redaction Value

To modify the default full data redaction value, use the `DBMS_REDACT.UPDATE_FULL_REDACTION_VALUES` procedure.

1. Log in to the database instance as a user who has been granted the `EXECUTE` privilege on the `DBMS_REDACT` PL/SQL package and who has administrative privileges, such as users who have been granted the `DBA` role.
2. Check the value that you want to change.

   For example, to check the current value for columns that use the `NUMBER` data type:

   ```
   SELECT NUMBER_VALUE FROM REDACTION_VALUES_FOR_TYPE_FULL;

   NUMBER_VALUE
   ------------
              0
   ```

3. Run the `DBMS_REDACT.UPDATE_FULL_REDACTION_VALUES` procedure to modify the value.

   For example:

   ```
   EXEC DBMS_REDACT.UPDATE_FULL_REDACTION_VALUES (number_val => 7);
   ```

4. Restart the database instance.

   For example:

   ```
   SHUTDOWN IMMEDIATE

   STARTUP
   ```

Creating a DBMS_REDACT.NULLIFY Redaction Policy

You can create Oracle Data Redaction policies that return null values for the displayed value of the table or view column.

- About Creating a Policy That Returns Null Values
  The `DBMS_REDACT.NULLIFY` `function_type` parameter redacts all the data in a column and replace it with null values.

About Creating a Policy That Returns Null Values

The DBMS_REDACT.NULLIFY function_type parameter redacts all the data in a column and replace it with null values.

You can use this function type on all supported column types that the DBMS_REDACT.FULL function type supports. It also supports the CLOB and NCLOB data types. To use the DBMS_REDACT.NULLIFY function, you must first ensure that the COMPATIBLE parameter is set at a minimum to 12.2.0.0.0.

Syntax for Creating a Policy That Returns Null Values

The DBMS_REDACT.ADD_POLICY procedure can create a redaction policy that performs a full redaction and displays null values for the redacted columns.

The syntax for using DBMS_REDACT.ADD_POLICY to return null values is as follows:

```
DBMS_REDACT.ADD_POLICY (
   object_schema           IN VARCHAR2 := NULL,
   object_name             IN VARCHAR2,
   column_name             IN VARCHAR2 := NULL,
   policy_name             IN VARCHAR2,
   function_type           IN BINARY_INTEGER := NULL,
   expression              IN VARCHAR2,
   enable                  IN BOOLEAN := TRUE);
```

In this specification:

- object_schema, object_name, column_name, policy_name, expression, enable: See General Syntax of the DBMS_REDACT.ADD_POLICY Procedure.
- function_type: Specifies the function used to set the type of redaction. Enter DBMS_REDACT.NULLIFY.

  If you omit the function_type parameter, then the default setting is DBMS_REDACT.FULL.

  Remember that the data type of the column determines which function_type settings that you are permitted to use. See Comparison of Full, Partial, and Random Redaction Based on Data Types.

Example: Redaction Policy That Returns Null Values

The DBMS_REDACT.ADD_POLICY procedure will return null values for the COMMISSION_PCT column of the HR.EMPLOYEES table.

The expression parameter applies the policy to any user who queries the table, except for users who have been granted the EXEMPT REDACTION POLICY system privilege.

Example 13-3 shows how to create the Oracle Data Redaction policy.

Example 13-3 Redaction Policy That Returns Null Values

```
BEGIN
 DBMS_REDACT.ADD_POLICY(
   object_schema    => 'hr',
   object_name      => 'employees',
   column_name      => 'commission_pct',
   policy_name      => 'nullify_com_pct',
   function_type    => DBMS_REDACT.NULLIFY,
   expression       => '1=1');
END;
/
```

Query and redacted result:

```
SELECT COMMISSION_PCT FROM HR.EMPLOYEES;

COMMISSION_PCT
--------------
```

Related Topics

- Exemption of Users from Oracle Data Redaction Policies
  You can exempt users from having Oracle Data Redaction policies applied to the data they access.

Creating a Partial Redaction Policy

In partial data redaction, you can redact portions of data, and for different kinds of data types.

- About Creating Partial Redaction Policies
  In partial data redaction, only a portion of the data, such as the first five digits of an identification number, are redacted.
- Syntax for Creating a Partial Redaction Policy
  The `DBMS_REDACT.ADD_POLICY` statement enables you to create policies that redact specific parts of the data returned to the application.
- Creating Partial Redaction Policies Using Fixed Character Formats
  The `DBMS_REDACT.ADD_POLICY function_parameters` parameter enables you to use fixed character formats.
- Creating Partial Redaction Policies Using Character Data Types
  The `DBMS_REDACT.ADD_POLICY function_parameters` parameter enables you to redact character data types.
- Creating Partial Redaction Policies Using Number Data Types
  The `DBMS_REDACT.ADD_POLICY function_parameters` parameter can redact number data types.
- Creating Partial Redaction Policies Using Date-Time Data Types
  The `DBMS_REDACT.ADD_POLICY function_parameters` parameter can redact date-time data types.

About Creating Partial Redaction Policies

In partial data redaction, only a portion of the data, such as the first five digits of an identification number, are redacted.

For example, you can redact most of a credit card number with asterisks (*), except for the last 4 digits. You can create policies for columns that use character, number, or date-time data types. For policies that redact character data types, you can use fixed character redaction formats. If you have the Enterprise Manager for Oracle Database 12.1.0.7 plug-in deployed on your system, then you can also create and save custom redaction formats.

> **Note:**
>
> In previous releases, the term shortcut was used for the term format.

Syntax for Creating a Partial Redaction Policy

The `DBMS_REDACT.ADD_POLICY` statement enables you to create policies that redact specific parts of the data returned to the application.

The `DBMS_REDACT.ADD_POLICY` fields for creating a partial redaction policy are as follows:

```
DBMS_REDACT.ADD_POLICY (
   object_schema           IN VARCHAR2 := NULL,
   object_name             IN VARCHAR2,
   column_name             IN VARCHAR2 := NULL,
   policy_name             IN VARCHAR2,
   function_type           IN BINARY_INTEGER := NULL,
   function_parameters     IN VARCHAR2 := NULL,
   expression              IN VARCHAR2,
   enable                  IN BOOLEAN := TRUE);
```

In this specification:

- `object_schema`, `object_name`, `column_name`, `policy_name`, `expression`, `enable`: See General Syntax of the DBMS_REDACT.ADD_POLICY Procedure
- `function_type`: Specifies the function used to set the type of redaction. Enter `DBMS_REDACT.PARTIAL`.
- `function_parameters`: The parameters that you set here depend on the data type of the column specified for the `column_name` parameter. See the following sections for details:
  - Creating Partial Redaction Policies Using Fixed Character Formats
  - Creating Partial Redaction Policies Using Character Data Types
  - Creating Partial Redaction Policies Using Number Data Types
  - Creating Partial Redaction Policies Using Date-Time Data Types

Creating Partial Redaction Policies Using Fixed Character Formats

The `DBMS_REDACT.ADD_POLICY function_parameters` parameter enables you to use fixed character formats.

- Settings for Fixed Character Formats
  Oracle Data Redaction provides special predefined formats to configure policies that use fixed characters.
- Example: Partial Redaction Policy Using a Fixed Character Format
  You can use the `DBMS_REDACT.ADD_POLICY` PL/SQL procedure to create a partial redaction policy that uses a fixed character format.

Settings for Fixed Character Formats

Oracle Data Redaction provides special predefined formats to configure policies that use fixed characters.

Table 13-8 describes `DBMS_REDACT.ADD_POLICY function_parameters` parameter formats that you can use for commonly redacted identity numbers (such as Social Security numbers or Canadian Social Insurance Numbers), postal codes, and credit cards that use either the `VARCHAR2` or `NUMBER` data types for their columns.

Table 13-8 Partial Fixed Character Redaction Formats

Format	Description
`DBMS_REDACT.REDACT_US_SSN_F5`	Redacts the first 5 numbers of Social Security numbers when the column is a `VARCHAR2` data type. For example, the number `987-65-4320` becomes `XXX-XX-4320`.
`DBMS_REDACT.REDACT_US_SSN_L4`	Redacts the last 4 numbers of Social Security numbers when the column is a `VARCHAR2` data type. For example, the number `987-65-4320` becomes `987-65-XXXX`.
`DBMS_REDACT.REDACT_US_SSN_ENTIRE`	Redacts the entire Social Security number when the column is a `VARCHAR2` data type. For example, the number `987-65-4320` becomes `XXX-XX-XXXX`.
`DBMS_REDACT.REDACT_NUM_US_SSN_F5`	Redacts the first 5 numbers of Social Security numbers when the column is a `NUMBER` data type. For example, the number `987654320` becomes `XXXXX4320`.
`DBMS_REDACT.REDACT_NUM_US_SSN_L4`	Redacts the last 4 numbers of Social Security numbers when the column is a `NUMBER` data type. For example, the number `987654320` becomes `98765XXXX`.
`DBMS_REDACT.REDACT_NUM_US_SSN_ENTIRE`	Redacts the entire Social Security number when the column is a `NUMBER` data type. For example, the number `987654320` becomes `XXXXXXXXX`.
`DBMS_REDACT.REDACT_SIN_NUMBER`	Redacts the Canadian Social Insurance number by replacing the first 6 digits by 9 (number). For example, `123456789` is redacted to `999999789`.

Table 13-8 (Cont.) Partial Fixed Character Redaction Formats

Format	Description
DBMS_REDACT.REDACT_SIN_UNFORMATTED	Redacts the Canadian Social Insurance number by replacing the first 6 digits by X (string). For example, 123456789 is redacted to XXXXXX789.
DBMS_REDACT.REDACT_SIN_FORMATTED	Redacts the Canadian Social Insurance Number by replacing the first 6 digits by X (string). For example, 123-456-789 is redacted to XXX-XXX-789.
DBMS_REDACT.REDACT_UK_NIN_FORMATTED	Redacts the UK National Insurance number by replacing the first 6 digits by X (string) but leaving the alphabetic characters as is. For example, ET 27 02 23 D is redacted to ET XX XX XX D.
DBMS_REDACT.REDACT_UK_NIN_UNFORMATTED	Redacts the UK National Insurance number by replacing the first 6 digits by X (string) and leaving the alphabetic characters as is. For example, ET270223D is redacted to ETXXXXXXD.
DBMS_REDACT.REDACT_CCN_FORMATTED	Redacts the credit card number (other than American Express) by replacing everything but the last 4 digits by *. For example, the credit card number 5105–1051–0510–5100 is redacted to ****–****–****–5100.
DBMS_REDACT.REDACT_CCN_NUMBER	Redacts the credit card number (other than American Express) by replacing everything but the last 4 digits by 0. For example, the credit card number 5105105105105100 is redacted to ************5100.
DBMS_REDACT.REDACT_CCN16_F12	Redacts a 16-digit credit card number (other than American Express), leaving the last 4 digits displayed. For example, 5105 1051 0510 5100 becomes ****-****-****-5100.
DBMS_REDACT.REDACT_AMEX_CCN_FORMATTED	Redacts the American Express credit card number by replacing the digits with * except the last 5 digits. For example, the credit card number 3782 822463 10005 is redacted to **** ****** 10005.
DBMS_REDACT.REDACT_AMEX_CCN_NUMBER	Redacts the American Express Credit Card Number by replacing the digits with 0 except the last 5 digits. For example, the credit card number 3782 822463 10005 is redacted to 0000 000000 10005.
DBMS_REDACT.REDACT_ZIP_CODE	Redacts a 5-digit postal code when the column is a VARCHAR2 data type. For example, 95476 becomes XXXXX.
DBMS_REDACT.REDACT_NUM_ZIP_CODE	Redacts a 5-digit postal code when the column is a NUMBER data type. For example, 95476 becomes XXXXX.
DBMS_REDACT.REDACT_DATE_EPOCH	Redacts all dates to 01-JAN-70.

Table 13-8 (Cont.) Partial Fixed Character Redaction Formats

Format	Description
DBMS_REDACT.REDACT_NA_PHONE_FORMATTED	Redacts the North American phone number by leaving the area code, but replacing everything else with X. For example, 650-555–0100 is redacted to 650-XXX-XXXX.
DBMS_REDACT.REDACT_NA_PHONE_NUMBER	Redacts the North American phone number by leaving the area code, but replacing everything else with 0. For example, 6505550100 gets redacted to 650000000.
DBMS_REDACT.REDACT_NA_PHONE_UNFORMATTED	Redacts the North American phone number by leaving the area code, but replacing everything else with X. For example, 6505550100 is redacted to 650XXXXXXX.
DBMS_REDACT.REDACT_DATE_MILLENNIUM	Redacts dates that are in the DD-MON-YY format to 01-JAN-00 (January 1, 2000).

See Also:

General Syntax of the DBMS_REDACT.ADD_POLICY Procedure for information about other DBMS_REDACT.ADD_POLICY parameters

Example: Partial Redaction Policy Using a Fixed Character Format

You can use the DBMS_REDACT.ADD_POLICY PL/SQL procedure to create a partial redaction policy that uses a fixed character format.

Example 13-4 shows how Social Security numbers in a VARCHAR2 data type column and can be redacted using the REDACT_US_SSN_F5 format.

Example 13-4 Partially Redacted Character Values

```
BEGIN
 DBMS_REDACT.ADD_POLICY(
   object_schema       => 'mavis',
   object_name         => 'cust_info',
   column_name         => 'ssn',
   policy_name         => 'redact_cust_ssns3',
   function_type       => DBMS_REDACT.PARTIAL,
   function_parameters => DBMS_REDACT.REDACT_US_SSN_F5,
   expression          => '1=1',
   policy_description  => 'Partially redacts 1st 5 digits in SS numbers',
   column_description  => 'ssn contains Social Security numbers');
END;
/
```

Query and redacted result:

```
SELECT ssn FROM mavis.cust_info;

SSN
```

```
-------
XXX-XX-4320
XXX-XX-4323
XXX-XX-4325
XXX-XX-4329
```

Creating Partial Redaction Policies Using Character Data Types

The `DBMS_REDACT.ADD_POLICY function_parameters` parameter enables you to redact character data types.

- Settings for Character Data Types
  Oracle Data Redaction provides special settings to configure policies that use character data types.
- Example: Partial Redaction Policy Using a Character Data Type
  The `DBMS_REDACT.ADD_POLICY` PL/SQL procedure can create a partial redaction policy that uses a character data type.

Settings for Character Data Types

Oracle Data Redaction provides special settings to configure policies that use character data types.

When you set the `DBMS_REDACT.ADD_POLICY function_parameters` parameter to define partial redaction of character data types, enter values for the following settings in the order shown. Separate each value with a comma

> **Note:**
>
> Be aware that you must use a fixed width character set for the partial redaction. In other words, each character redacted must be replaced by another of equal byte length. If you want to use a variable-length character set (for example, UTF-8), then you must use a regular expression-based redaction. See Syntax for Creating a Regular Expression-Based Redaction Policy for more information.

The settings are as follows:

1. **Input format:** Defines how the data is currently formatted. Enter `V` for each character that potentially can be redacted, such as all of the digits in a credit card number. Enter `F` for each character that you want to format using a formatting character, such as hyphens or blank spaces in the credit card number. Ensure that each character has a corresponding `V` or `F` value. (The input format values are not case-sensitive.)
2. **Output format:** Defines how the displayed data should be formatted. Enter `V` for each character to be potentially redacted. Replace each `F` character in the input format with the character that you want to use for the displayed output, such as a hyphen. (The output format values are not case-sensitive.)
3. **Mask character:** Specifies the character to be used for the redaction. Enter a single character to use for the redaction, such as an asterisk (*).
4. **Starting digit position:** Specifies the starting `V` digit position for the redaction.

5. **Ending digit position:** Specifies the ending `V` digit position for the redaction. Do not include the `F` positions when you decide on the ending position value.

For example, the following setting redacts the first 12 `V` digits of the credit card number `5105 1051 0510 5100`, and replaces the `F` positions (which are blank spaces) with hyphens to format it in a style normally used for credit card numbers, resulting in `****-****-****-4320`.

```
function_parameters  => 'VVVVFVVVVFVVVVFVVVV,VVVV-VVVV-VVVV-VVVV,*,1,12',
```

> **See Also:**
>
> General Syntax of the DBMS_REDACT.ADD_POLICY Procedure for information about other `DBMS_REDACT.ADD_POLICY` parameters

Example: Partial Redaction Policy Using a Character Data Type

The `DBMS_REDACT.ADD_POLICY` PL/SQL procedure can create a partial redaction policy that uses a character data type.

Example 13-5 shows how to redact Social Security numbers that are in a `VARCHAR2` data type column and to preserve the character hyphens in the Social Security number.

Example 13-5 Partially Redacted Character Values

```
BEGIN
 DBMS_REDACT.ADD_POLICY(
   object_schema       => 'mavis',
   object_name         => 'cust_info',
   column_name         => 'ssn',
   policy_name         => 'redact_cust_ssns2',
   function_type       => DBMS_REDACT.PARTIAL,
   function_parameters => 'VVVFVVFVVVV,VVV-VV-VVVV,*,1,5',
   expression          => '1=1',
   policy_description  => 'Partially redacts Social Security numbers',
   column_description  => 'ssn contains character Social Security numbers');
END;
/
```

Query and redacted result:

```
SELECT ssn FROM mavis.cust_info;

SSN
-----------
***-**-4320
***-**-4323
***-**-4325
***-**-4329
```

Creating Partial Redaction Policies Using Number Data Types

The `DBMS_REDACT.ADD_POLICY` `function_parameters` parameter can redact number data types.

- Settings for Number Data Types
  When you set values for the number data type, you must specify a mask character, a starting digit position, and ending digit position.
- Example: Partial Redaction Policy Using a Number Data Type
  The `DBMS_REDACT.ADD_POLICY` procedure can create a partial redaction policy that uses a number data type.

Settings for Number Data Types

When you set values for the number data type, you must specify a mask character, a starting digit position, and ending digit position.

For partial redaction of number data types, you can enter values for the following settings for the `function_parameters` parameter of the `DBMS_REDACT.ADD_POLICY` procedure, in the order shown.

1. **Mask character:** Specifies the character to display. Enter a number from 0 to 9.
2. **Starting digit position:** Specifies the starting digit position for the redaction, such as 1 for the first digit.
3. **Ending digit position:** Specifies the ending digit position for the redaction.

For example, the following setting redacts the first five digits of the Social Security number `987654321`, resulting in `999994321`.

```
function_parameters  => '9,1,5',
```

> **See Also:**
>
> General Syntax of the DBMS_REDACT.ADD_POLICY Procedure for information about other `DBMS_REDACT.ADD_POLICY` parameters

Example: Partial Redaction Policy Using a Number Data Type

The `DBMS_REDACT.ADD_POLICY` procedure can create a partial redaction policy that uses a number data type.

Example 13-6 shows how to partially redact a set of Social Security numbers in the `mavis.cust_info` table, for any application user who logs in. (Hence, the `expression` parameter evaluates to `TRUE`.)

This type of redaction is useful when the application is expecting a formatted number and not a string. In this scenario, the Social Security numbers are in a column of the data type `NUMBER`. In other words, the `ssn` column contains numbers only, not other characters such as hyphens or blank spaces.

Example 13-6 Partially Redacted Data Redaction Numeric Values

```
BEGIN
 DBMS_REDACT.ADD_POLICY(
   object_schema       => 'mavis',
   object_name         => 'cust_info',
   column_name         => 'ssn',
   policy_name         => 'redact_cust_ssns1',
```

```
   function_type       => DBMS_REDACT.PARTIAL,
   function_parameters => '7,1,5',
   expression          => '1=1',
   policy_description  => 'Partially redacts Social Security numbers',
   column_description  => 'ssn contains numeric Social Security numbers');
END;
/
```

Query and redacted result:

```
SELECT ssn FROM mavis.cust_info;

SSN
---------
777774320
777774323
777774325
777774329
```

Creating Partial Redaction Policies Using Date-Time Data Types

The `DBMS_REDACT.ADD_POLICY` `function_parameters` parameter can redact date-time data types.

- Settings for Date-Time Data Types
  Oracle Data Redaction provides special settings for configuring date-time data types.
- Example: Partial Redaction Policy Using Date-Time Data Type
  The `DBMS_REDACT.ADD_POLICY` procedure can create a partial redaction policy that uses the date-time data type.

Settings for Date-Time Data Types

Oracle Data Redaction provides special settings for configuring date-time data types.

For partial redaction of date-time data types, enter values for the following `DBMS_REDACT.ADD_POLICY` `function_parameters` parameter settings.

Enter these values in the order shown:

1. m: Redacts the month. To redact with a month name, append `1–12` to lowercase `m`. For example, `m5` displays as `MAY`. To omit redaction, enter an uppercase `M`.
2. d: Redacts the day of the month. To redact with a day of the month, append `1–31` to a lowercase `d`. For example, `d7` displays as `07`. If you enter a higher number than the days of the month (for example, `31` for the month of February), then the last day of the month is displayed (for example, `28`). To omit redaction, enter an uppercase `D`.
3. y: Redacts the year. To redact with a year, append `1–9999` to a lowercase `y`. For example, `y1984` displays as `84`. To omit redaction, enter an uppercase `Y`.
4. h: Redacts the hour. To redact with an hour, append `0–23` to a lowercase `h`. For example, `h20` displays as `20`. To omit redaction, enter an uppercase `H`.
5. m: Redacts the minute. To redact with a minute, append `0–59` to a lowercase `m`. For example, `m30` displays as `30`. To omit redaction, enter an uppercase `M`.

6. s: Redacts the second. To redact with a second, append `0`–`59` to a lowercase `s`. For example, `s45` displays as `45`. To omit redaction, enter an uppercase `S`.

> **See Also:**
>
> General Syntax of the DBMS_REDACT.ADD_POLICY Procedure for information about other `DBMS_REDACT.ADD_POLICY` parameters

Example: Partial Redaction Policy Using Date-Time Data Type

The `DBMS_REDACT.ADD_POLICY` procedure can create a partial redaction policy that uses the date-time data type.

Example 13-7 shows how to partially redact a date. This example redacts the birth year of customers; replacing it with 13, but retaining the remaining values.

Example 13-7 Partially Redacted Data Redaction Using Date-Time Values

```
BEGIN
 DBMS_REDACT.ADD_POLICY(
   object_schema       => 'mavis',
   object_name         => 'cust_info',
   column_name         => 'birth_date',
   policy_name         => 'redact_cust_bdate',
   function_type       => DBMS_REDACT.PARTIAL,
   function_parameters => 'mdy2013HMS',
   expression          => '1=1',
   policy_description  => 'Replaces birth year with 2013',
   column_description  => 'birth_date contains customer's birthdate');
END;
/
```

Query and redacted result:

```
SELECT birth_date FROM mavis.cust_info;

BIRTH_DATE
07-DEC-13 09.45.40.000000 AM
12-OCT-13 04.23.29.000000 AM
```

Creating a Regular Expression-Based Redaction Policy

A regular expression-based redaction policy enables you to redact data based on a search-and-replace model.

- About Creating Regular Expression-Based Redaction Policies
  Regular expression-based redaction enables you to search for patterns of data to redact.
- Syntax for Creating a Regular Expression-Based Redaction Policy
  The `regexp_*` parameters of the `DBMS_REDACT.ADD_POLICY` procedure can create a regular expression-based redaction policy.

- Regular Expression-Based Redaction Policies Using Formats
  The `DBMS_REDACT.ADD_POLICY` procedure `regexp_pattern` and `regexp_replace_string` parameters both support formats.
- Custom Regular Expression Redaction Policies
  You can customize regular expressions in Data Redaction policies.

About Creating Regular Expression-Based Redaction Policies

Regular expression-based redaction enables you to search for patterns of data to redact.

For example, you can use regular expressions to redact email addresses, which can have varying character lengths. It is designed for use with character data only. You can use formats for the search and replace operation, or you can create custom pattern formats.

You cannot use regular expressions to redact a subset of the values in a column. The `REGEXP_PATTERN` (regular expression pattern) must match *all* of the values in order for the `REGEXP_REPLACE_STRING` setting to take effect, and the `REGEXP_REPLACE_STRING` must change the value.

For rows where the `REGEXP_PATTERN` fails to match, Data Redaction performs `DBMS_REDACT.FULL` redaction. This mitigates the risk of a mistake in the `REGEXP_PATTERN` which causes the regular expression to fail to match all of the values in the column, from showing the actual data for those rows which it failed to match.

In addition, if no change to the value occurs as a result of the `REGEXP_REPLACE_STRING` setting during regular expression replacement operation, Data Redaction performs `DBMS_REDACT.FULL` redaction.

Syntax for Creating a Regular Expression-Based Redaction Policy

The `regexp_*` parameters of the `DBMS_REDACT.ADD_POLICY` procedure can create a regular expression-based redaction policy.

The `DBMS_REDACT.ADD_POLICY` fields for creating a regular expression-based data redaction policy are as follows:

```
DBMS_REDACT.ADD_POLICY (
   object_schema           IN VARCHAR2 := NULL,
   object_name             IN VARCHAR2,
   column_name             IN VARCHAR2 := NULL,
   policy_name             IN VARCHAR2,
   function_type           IN BINARY_INTEGER := NULL,
   expression              IN VARCHAR2,
   enable                  IN BOOLEAN := TRUE,
   regexp_pattern          IN VARCHAR2 := NULL,
   regexp_replace_string   IN VARCHAR2 := NULL,
   regexp_position         IN BINARY_INTEGER := 1,
   regexp_occurrence       IN BINARY_INTEGER := 0,
   regexp_match_parameter  IN VARCHAR2 := NULL);
```

In this specification:

- `object_schema`, `object_name`, `column_name`, `policy_name`, `expression`, `enable`:
  See General Syntax of the DBMS_REDACT.ADD_POLICY Procedure.

- `function_type`: Specifies the type of redaction. For regular expression based redaction, use either `DBMS_REDACT.REGEXP` or `DBMS_REDACT.REGEXP_WIDTH`.

  If you use the `DBMS_REDACT.REGEXP` redaction type, then no truncation occurs. This applies even if the redacted value is wider than the column width, and if the Oracle Call Interface width attribute (`OCI_ATTR_CHAR_SIZE`) of the column is *not* preserved. (It becomes 4000, just as it does when the `REGEXP_REPLACE` SQL operator is used on a column.)

  Using the `DBMS_REDACT.REGEXP_WIDTH` redaction type truncates any redacted value that exceeds the width of the column, and ensures that the OCI width attribute of the column (`OCI_ATTR_CHAR_SIZE`) remains unchanged.

  Note the following:

  - Use the `DBMS_REDACT.REGEXP_WIDTH` function type if your applications depend on the value of the `OCI_ATTR_CHAR_SIZE` attribute. For example, applications that are built using the Oracle OLE DB Provider interface are sensitive to the value of the `OCI_ATTR_CHAR_SIZE` attribute. If you use `DBMS_REDACT.REGEXP` as the redaction type, then the `OCI_ATTR_CHAR_SIZE` always becomes 4000. This setting makes it unsuitable as the redaction type of policies on tables that are used by Oracle OLE DB based applications. See *Oracle Call Interface Programmer's Guide* for more information about Oracle Call Interface parameter attributes.
  - When you set the `function_type` parameter to `DBMS_REDACT.REGEXP` or `DBMS_REDACT.REGEXP_WIDTH`, omit the `function_parameters` parameter from the `DBMS_REDACT.ADD_POLICY` procedure.
  - Specify the regular expression parameters in much the same way that you specify the `pattern`, `replace`, `position`, `occurrence`, and `match_parameter` arguments to the `REGEXP_REPLACE` SQL function. See *Oracle Database SQL Language Reference* for information about the `REGEXP_REPLACE` SQL function.

- `regexp_pattern`: Describes the search pattern for data that must be matched. If it finds a match, then Oracle Database replaces the data as specified by the `regexp_replace_string` setting. See the following sections for more information:

  - Regular Expression-Based Redaction Policies Using Formats
  - Custom Regular Expression Redaction Policies

- `regexp_replace_string`: Specifies how you want to replace the data to be redacted. See the following sections for more information:

  - Regular Expression-Based Redaction Policies Using Formats
  - Custom Regular Expression Redaction Policies

- `regexp_position`: Specifies the starting position for the string search. The value that you enter must be a positive integer indicating the character of the `column_name` data where Oracle Database should begin the search. The default is `1` or the `DBMS_REDACT.RE_BEGINNING` format, meaning that Oracle Database begins the search at the first character of the `column_name` data.

- `regexp_occurrence`: Specifies how to perform the search and replace operation. The value that you enter must be a nonnegative integer indicating the occurrence of the replace operation:

  - If you specify `0` or the `DBMS_REDACT.RE_ALL` format, then Oracle Database replaces all the occurrences of the match.

  - If you specify the `DBMS_REDACT.RE_FIRST` format, then Oracle Database replaces the first occurrence of the match.
  - If you specify a positive integer *n*, then Oracle Database replaces the *n*th occurrence of the match.

  If the occurrence is greater than 1, then the database searches for the second occurrence beginning with the first character following the first occurrence of pattern, and so forth.

- `regexp_match_parameter`: Specifies a text literal that lets you change the default matching behavior of the function. The behavior of this parameter is the same for this function as for the `REGEXP_REPLACE` SQL function. See *Oracle Database SQL Language Reference* for detailed information.

  To filter the search so that it is not case sensitive, specify the `RE_CASE_INSENSITIVE` format.

Regular Expression-Based Redaction Policies Using Formats

The `DBMS_REDACT.ADD_POLICY` procedure `regexp_pattern` and `regexp_replace_string` parameters both support formats.

- Regular Expression Formats
  The regular expression formats represent commonly used expressions, such as the replacement of digits within a credit card number.
- Example: Regular Expression Redaction Policy Using Formats
  The `DBMS_REDACT.ADD_POLICY` procedure can create a regular expression redaction policy that uses formats.

Regular Expression Formats

The regular expression formats represent commonly used expressions, such as the replacement of digits within a credit card number.

Table 13-9 describes the formats that you can use with the `regexp_pattern` parameter in the `DBMS_REDACT.ADD_POLICY` procedure.

Table 13-9 Formats for the regexp_pattern Parameter

Format	Description
DBMS_REDACT.RE_PATTERN_ANY_DIGIT	Searches for any digit. Replaces the identified pattern with the characters specified by the regexp_replace_string parameter. The DBMS_REDACT.RE_PATTERN_ANY_DIGIT is commonly used with the following values of the regexp_replace_string parameter: regexp_replace_string => DBMS_REDACT.RE_REDACT_WITH_SINGLE_X, This setting replaces any matched digit with the X character. The following setting replaces any matched digit with the 1 character. regexp_replace_string => DBMS_REDACT.RE_REDACT_WITH_SINGLE_1,
DBMS_REDACT.RE_PATTERN_CC_L6_T4	Searches for the middle digits of any credit card (other than American Express) that has 6 leading digits and 4 trailing digits. Replaces the identified pattern with the characters specified by the regexp_replace_string parameter. The appropriate regexp_replace_string setting to use with this format is DBMS_REDACT.RE_REDACT_CC_MIDDLE_DIGITS , which finds any credit card that could have 6 leading and 4 trailing digits left as actual data. It then redacts the middle digits.
DBMS_REDACT.RE_PATTERN_CCN	Matches credit card numbers other than American Express credit card numbers. The appropriate regexp_replace_string setting to use with this format is DBMS_REDACT.RE_REDACT_CCN. The end result is a redaction of all the digits except the last 4.
DBMS_REDACT.RE_PATTERN_AMEX_CCN	Matches American Express credit card numbers. The appropriate regexp_replace_string setting to use with this format is DBMS_REDACT.RE_REDACT_AMEX_CCN. The end result is a redaction of all the digits except the last 5.
DBMS_REDACT.RE_PATTERN_US_PHONE	Searches for any U.S. telephone number. Replaces the identified pattern with the characters specified by the regexp_replace_string parameter The appropriate regexp_replace_string setting to use with this format is DBMS_REDACT.RE_REDACT_US_PHONE_L7, which finds United States phone numbers and then redacts the last 7 digits.

Table 13-9 (Cont.) Formats for the regexp_pattern Parameter

Format	Description
DBMS_REDACT.RE_PATTERN_EMAIL_ADDRESS	Searches for any email address. Replaces the identified pattern with the characters specified by the regexp_replace_string parameter The appropriate regexp_replace_string settings that you can use with this format are as follows: RE_REDACT_EMAIL_NAME, which finds any email address and redacts the email user name RE_REDACT_EMAIL_DOMAIN, which finds any email address and redacts the email domain RE_REDACT_EMAIL_ENTIRE, which finds any email address and redacts the entire email address
DBMS_REDACT.RE_PATTERN_IP_ADDRESS	Searches for an IP address. Replaces the identified pattern with the characters specified by the regexp_replace_string parameter. The appropriate regexp_replace_string setting to use with this format is DBMS_REDACT.RE_REDACT_IP_L3, which replaces the last section of the dotted decimal string representation of an IP address with the characters 999 to indicate that it was redacted.

Table 13-10 describes formats that you can use with the regexp_replace_string parameter in the DBMS_REDACT.ADD_POLICY procedure.

Table 13-10 Formats for the regexp_replace_string Parameter

Format	Description
DBMS_REDACT.RE_REDACT_WITH_SINGLE_X	Replaces the data with a single X character for each of the actual data characters. For example, the credit card number 5105 1051 0510 5100 could be replaced with XXXX XXXX XXXX XXXX.
DBMS_REDACT.RE_REDACT_WITH_SINGLE_1	Replaces the data with a single 1 digit for each of the actual data digits. For example, the credit card number 5105 1051 0510 5100 could be replaced with 1111 1111 1111 1111.
DBMS_REDACT.RE_REDACT_CC_MIDDLE_DIGITS	Redacts the middle digits in credit card numbers, as specified by setting the regexp_pattern parameter with the RE_PATTERN_CC_L6_T4 format. The redaction replaces each redacted character with an X. For example, the credit card number 5105 1051 0510 5100 could be replaced with 5105 10XX XXXX 5100.
DBMS_REDACT.RE_REDACT_CCN	Redacts the first 12 digits of a credit card number other than an American Express card number. For example, 4012888888881881 is redacted to ************1881.

Table 13-10 (Cont.) Formats for the regexp_replace_string Parameter

Format	Description
`DBMS_REDACT.RE_REDACT_AMEX_CCN`	Redacts the first 10 digits of an American Express number. For example, `378282246310005` is redacted to `**********10005`.
`DBMS_REDACT.RE_REDACT_PHONE_L7`	Redacts the last 7 digits of U.S. telephone numbers, as specified by setting the `regexp_pattern` parameter with the `RE_PATTERN_US_PHONE` format. The redaction replaces each redacted character with an `X`. This setting only applies to hyphenated phone numbers, not phone numbers with spaces. For example, the telephone number `415-555-0100` could be replaced with `415-XXX-XXXX`.
`DBMS_REDACT.RE_REDACT_EMAIL_NAME`	Redacts the email name as specified by setting the `regexp_pattern` parameter with the `RE_PATTERN_EMAIL_ADDRESS` format. The redaction replaces the email user name with four `x` characters. For example, the email address `psmith@example.com` could be replaced with `xxxx@example.com`.
`DBMS_REDACT.RE_REDACT_EMAIL_DOMAIN`	Redacts the email domain name as specified by setting the `regexp_pattern` parameter with the `RE_PATTERN_EMAIL_ADDRESS` format. The redaction replaces the domain with five `x` characters. For example, the email address `psmith@example.com` could be replaced with `psmith@xxxxx.com`.
`DBMS_REDACT.RE_REDACT_IP_L3`	Redacts the last three digits of the IP address as specified by setting the `regexp_pattern` parameter with the `RE_PATTERN_IP_ADDRESS` format. For example, the IP address `192.0.2.254` could be replaced with `192.0.2.999`, which is an invalid IP address.

See Also:

General Syntax of the DBMS_REDACT.ADD_POLICY Procedure for information about other `DBMS_REDACT.ADD_POLICY` parameters

Example: Regular Expression Redaction Policy Using Formats

The `DBMS_REDACT.ADD_POLICY` procedure can create a regular expression redaction policy that uses formats.

Example 13-8 shows how to use regular expression formats to redact credit card numbers.

Example 13-8 Regular Expression Data Redaction Character Values

```
BEGIN
 DBMS_REDACT.ADD_POLICY(
   object_schema          => 'mavis',
   object_name            => 'cust_info',
   column_name            => 'cc_num',
   policy_name            => 'redact_cust_cc_nums',
   function_type          => DBMS_REDACT.REGEXP,
   function_parameters    => NULL,
   expression             => '1=1',
   regexp_pattern         => DBMS_REDACT.RE_PATTERN_CC_L6_T4,
   regexp_replace_string  => DBMS_REDACT.RE_REDACT_CC_MIDDLE_DIGITS,
   regexp_position        => DBMS_REDACT.RE_BEGINNING,
   regexp_occurrence      => DBMS_REDACT.RE_FIRST,
   regexp_match_parameter => DBMS_REDACT.RE_CASE_INSENSITIVE,
   policy_description     => 'Regular expressions to redact credit card numbers',
   column_description     => 'cc_num contains customer credit card numbers');
END;
/
```

Query and redacted result:

```
SELECT cc_num FROM mavis.cust_info;

CC_NUM
-------
401288XXXXXX1881
411111XXXXXX1111
555555XXXXXX1111
511111XXXXXX1118
```

Custom Regular Expression Redaction Policies

You can customize regular expressions in Data Redaction policies.

- Settings for Custom Regular Expressions
  Oracle Data Redaction provides special settings to configure policies that use regular expressions.
- Example: Custom Regular Expression Redaction Policy
  The `DBMS_REDACT.ADD_POLICY` procedure `regexp*` parameters can create a custom regular expression redaction policy.

Settings for Custom Regular Expressions

Oracle Data Redaction provides special settings to configure policies that use regular expressions.

To create custom regular expression redaction policies, you use the following parameters in the `DBMS_REDACT.ADD_POLICY` procedure:

- `regexp_pattern`: This pattern is usually a text literal and can be of any of the data types `CHAR`, `VARCHAR2`, `NCHAR`, or `NVARCHAR2`. The pattern can contain up to 512 bytes. For further information about writing the regular expression for the `regexp_pattern` parameter, see the description of the `pattern` argument of the `REGEXP_REPLACE` SQL function in *Oracle Database SQL Language Reference*, because the support that Data Redaction provides for regular expression matching is similar to that of the `REGEXP_REPLACE` SQL function.

- `regexp_replace_string`: This data can be of any of the data types `CHAR`, `VARCHAR2`, `NCHAR`, or `NVARCHAR2`. The `regexp_replace_string` can contain up to 500 back references to subexpressions in the form `\n`, where *n* is a number from 1 to 9. If you want to include a backslash (\) in the `regexp_replace_string` setting, then you must precede it with the escape character, which is also a backslash. For example, to literally replace the matched pattern with `\2` (rather than replace it with the second matched subexpression of the matched pattern), you enter `\\2` in the `regexp_replace_string` setting. For more information, see *Oracle Database SQL Language Reference*.

> **See Also:**
>
> General Syntax of the DBMS_REDACT.ADD_POLICY Procedure for information about other `DBMS_REDACT.ADD_POLICY` parameters

Example: Custom Regular Expression Redaction Policy

The `DBMS_REDACT.ADD_POLICY` procedure `regexp*` parameters can create a custom regular expression redaction policy.

Example 13-9 shows how to use regular expressions to redact the `emp_id` column data. In this example, taken together, the `regexp_pattern` and `regexp_replace_string` parameters do the following: first, find the pattern of 9 digits. For reference, break them into three groups that contain the first 3, the next 2, and then the last 4 digits. Then, replace all 9 digits with `XXXXX` concatenated with the third group (the last 4 digits) as found in the original pattern.

Query and redacted result:

```
SELECT emp_id FROM mavis.cust_info;

EMP_ID
------------
XXXXX1234
XXXXX5678
```

Example 13-9 Partially Redacted Data Redaction Using Regular Expressions

```
BEGIN
 DBMS_REDACT.ADD_POLICY(
   object_schema          => 'mavis',
   object_name            => 'cust_info',
   column_name            => 'emp_id',
   policy_name            => 'redact_cust_ids',
   function_type          => DBMS_REDACT.REGEXP,
   expression             => '1=1',
   regexp_pattern         => '(\d\d\d)(\d\d)(\d\d\d\d)',
   regexp_replace_string  => 'XXXXX\3',
   regexp_position        => 1,
   regexp_occurrence      => 0,
   regexp_match_parameter => 'i',
   policy_description     => 'Redacts customer IDs using regular expression',
   column_description     => 'emp_id contains employee ID numbers');
END;
/
```

Creating a Random Redaction Policy

A random redaction policy presents redacted data as randomly generated values, such as `Ukjsl32[[]]]s`.

- Syntax for Creating a Random Redaction Policy
  A random redaction policy presents the redacted data to the querying application user as randomly generated values, based on the column data type.
- Example: Random Redaction Policy
  You can use the `DBMS_REDACT.ADD_POLICY` PL/SQL procedure create a random redaction policy.

Syntax for Creating a Random Redaction Policy

A random redaction policy presents the redacted data to the querying application user as randomly generated values, based on the column data type.

Be aware that LOB columns are not supported.

The `DBMS_REDACT.ADD_POLICY` fields for creating a random redaction policy are as follows:

```
DBMS_REDACT.ADD_POLICY (
   object_schema           IN VARCHAR2 := NULL,
   object_name             IN VARCHAR2,
   column_name             IN VARCHAR2 := NULL,
   policy_name             IN VARCHAR2,
   function_type           IN BINARY_INTEGER := NULL,
   expression              IN VARCHAR2,
   enable                  IN BOOLEAN := TRUE);
```

In this specification:

- `object_schema`, `object_name`, `column_name`, `policy_name`, `expression`, `enable`: See General Syntax of the DBMS_REDACT.ADD_POLICY Procedure.
- `function_type`: Specifies the function used to set the type of redaction. Enter `DBMS_REDACT.RANDOM`.

  If you omit the `function_type` parameter, then the default redaction `function_type` setting is `DBMS_REDACT.FULL`.

  Remember that the data type of the column determines which `function_type` settings that you are permitted to use. See Comparison of Full, Partial, and Random Redaction Based on Data Types.

Example: Random Redaction Policy

You can use the `DBMS_REDACT.ADD_POLICY` PL/SQL procedure create a random redaction policy.

Example 13-10 shows how to generate random values. Each time you run the `SELECT` statement, the output will be different.

Example 13-10 Randomly Redacted Data Redaction Values

```
BEGIN
 DBMS_REDACT.ADD_POLICY(
   object_schema   => 'mavis',
   object_name     => 'cust_info',
   column_name     => 'login_username',
   policy_name     => 'redact_cust_rand_username',
   function_type   => DBMS_REDACT.RANDOM,
   expression      => 'SYS_CONTEXT(''USERENV'',''SESSION_USER'') =
''APP_USER''');
END;
/
```

Query and redacted result:

```
SELECT login_username FROM mavis.cust_info;

LOGIN_USERNAME
--------------
N[CG{\pTVcK
```

Creating a Policy That Uses No Redaction

You can create policies that use no redaction at all, for when you want to test the policy in a development environment.

- Syntax for Creating a Policy with No Redaction
  The None redaction type option can be used to test the internal operation of redaction policies.
- Example: Performing No Redaction
  The `DBMS_REDACT.ADD_POLICY` procedure can create a policy that performs no redaction.

Syntax for Creating a Policy with No Redaction

The None redaction type option can be used to test the internal operation of redaction policies.

The None redaction type has no effect on the query results against tables that have policies defined on them. You can use this option to test the redaction policy definitions before applying them to a production environment. Be aware that LOB columns are not supported.

The `DBMS_REDACT.ADD_POLICY` fields for creating a policy with no redaction are as follows:

```
DBMS_REDACT.ADD_POLICY (
   object_schema           IN VARCHAR2 := NULL,
   object_name             IN VARCHAR2,
   column_name             IN VARCHAR2 := NULL,
   policy_name             IN VARCHAR2,
   function_type           IN BINARY_INTEGER := NULL,
   expression              IN VARCHAR2,
   enable                  IN BOOLEAN := TRUE);
```

In this specification:

- `object_schema`, `object_name`, `column_name`, `policy_name`, `expression`, `enable`: See General Syntax of the DBMS_REDACT.ADD_POLICY Procedure.
- `function_type`: Specifies the functions used to set the type of data redaction. Enter `DBMS_REDACT.NONE`.

  If you omit the `function_type` parameter, then the default redaction `function_type` setting is `DBMS_REDACT.FULL`.

Example: Performing No Redaction

The `DBMS_REDACT.ADD_POLICY` procedure can create a policy that performs no redaction.

Example 13-11 shows how to create a Data Redaction policy that does not redact any of the displayed values.

Example 13-11 No Redacted Data Redaction Values

```
BEGIN
 DBMS_REDACT.ADD_POLICY(
   object_schema    => 'mavis',
   object_name      => 'cust_info',
   column_name      => 'user_name',
   policy_name      => 'redact_cust_no_vals',
   function_type    => DBMS_REDACT.NONE,
   expression       => '1=1');
END;
/
```

Query and redacted result:

```
SELECT user_name FROM mavis.cust_info;

USER_NAME
----------
IDA NEAU
```

Exemption of Users from Oracle Data Redaction Policies

You can exempt users from having Oracle Data Redaction policies applied to the data they access.

To do so, you should grant the users the `EXEMPT REDACTION POLICY` system privilege. Grant this privilege to trusted users only.

In addition to users who were granted this privilege, user `SYS` is also exempt from all Data Redaction policies. The person who creates the Data Redaction policy is by default not exempt from it, unless this person is user `SYS` or has the `EXEMPT REDACTION POLICY` system privilege.

Note the following:

- Users who have the `INSERT` privilege on a table can insert values into a redacted column, regardless of whether a Data Redaction policy exists on the table. Data Redaction only affects SQL `SELECT` statements (that is, queries) issued by a user, and has no effect on any other SQL issued by a user, including `INSERT`, `UPDATE`, or `DELETE` statements. (See the next bullet for exceptions to this rule.)

- Users cannot perform a `CREATE TABLE AS SELECT` where any of the columns being selected (source columns) is protected by a Data Redaction policy (and similarly, any DML operation where the source is a redacted column), unless the user was granted the `EXEMPT REDACTION POLICY` system privilege.
- The `EXEMPT REDACTION POLICY` system privilege is included in the `DBA` role, but this privilege must be granted explicitly to users because it is not included in the `WITH ADMIN OPTION` for `DBA` role grants. Users who were granted the `DBA` role are exempt from redaction policies because the `DBA` role contains the `EXP_FULL_DATABASE` role, which is granted the `EXEMPT REDACTION POLICY` system privilege.

Related Topics

- Restriction of Administrative Access to Oracle Data Redaction Policies
  You can restrict the list of users who can create, view and edit Data Redaction policies.
- Oracle Data Pump Security Model for Oracle Data Redaction
  The `DATAPUMP_EXP_FULL_DATABASE` role includes the powerful `EXEMPT REDACTION POLICY` system privilege.

Altering an Oracle Data Redaction Policy

The `DBMS_REDACT.ALTER_POLICY` procedure enables you to modify Oracle Data Redaction policies.

- About Altering Oracle Data Redaction Policies
  The `DBMS_REDACT.ALTER_POLICY` procedure alters a Data Redaction policy.
- Syntax for the DBMS_REDACT.ALTER_POLICY Procedure
  The `DBMS_REDACT.ALTER_POLICY` procedure syntax can be used to alter all types of the Data Redaction policies.
- Parameters Required for DBMS_REDACT.ALTER_POLICY Actions
  The `DBMS_REDACT.ALTER_POLICY` procedure provides parameters than can perform various actions, such as adding or modifying a column.
- Tutorial: Altering an Oracle Data Redaction Policy
  You can redact multiple columns in a table or view, with each column having its own redaction setting.

About Altering Oracle Data Redaction Policies

The `DBMS_REDACT.ALTER_POLICY` procedure alters a Data Redaction policy.

If the policy is already enabled, then you do not need to disable it first, and after you alter the policy, it remains enabled.

You can find the names of existing Data Redaction policies by querying the `POLICY_NAME` column of the `REDACTION_POLICIES` data dictionary view, and information about the columns, functions, and parameters specified in a policy by querying the `REDACTION_COLUMNS` view. To find the current value for policies that use full data redaction, you can query the `REDACTION_VALUES_FOR_TYPE_FULL` data dictionary view.

The `action` parameter specifies the type of modification that you want to perform. At a minimum, you must include the `object_name` and `policy_name` parameters when you run this procedure.

Related Topics

- REDACTION_COLUMNS Data Dictionary View Behavior When a View Is Invalid
  When an Oracle Data Redaction policy exists on a column of a view, and the view becomes invalid, the Data Redaction policy remains visible in the `REDACTION_COLUMNS` data dictionary view.

Syntax for the DBMS_REDACT.ALTER_POLICY Procedure

The `DBMS_REDACT.ALTER_POLICY` procedure syntax can be used to alter all types of the Data Redaction policies.

The syntax for the `DBMS_REDACT.ALTER_POLICY` procedure is as follows:

```
DBMS_REDACT.ALTER_POLICY (
   object_schema          IN VARCHAR2 := NULL,
   object_name            IN VARCHAR2 := NULL,
   policy_name            IN VARCHAR2,
   action                 IN BINARY_INTEGER := DBMS_REDACT.ADD_COLUMN,
   column_name            IN VARCHAR2 := NULL,
   function_type          IN BINARY_INTEGER := DBMS_REDACT.FULL,
   function_parameters    IN VARCHAR2 := NULL,
   expression             IN VARCHAR2 := NULL,
   regexp_pattern         IN VARCHAR2 := NULL,
   regexp_replace_string  IN VARCHAR2 := NULL,
   regexp_position        IN BINARY_INTEGER := NULL,
   regexp_occurrence      IN BINARY_INTEGER := NULL,
   regexp_match_parameter IN VARCHAR2 := NULL,
   policy_description     IN VARCHAR2 := NULL,
   column_description     IN VARCHAR2 := NULL);
```

In this specification:

- `action`: Enter one of the following values to define the kind of action to use:
  - `DBMS_REDACT.MODIFY_COLUMN` if you plan to change the `column_name` value.
  - `DBMS_REDACT.ADD_COLUMN` if you plan to add a new column (in addition to columns that are already protected by the policy) for redaction. This setting is the default for the `action` parameter.
  - `DBMS_REDACT.DROP_COLUMN` if you want to remove redaction from a column.
  - `DBMS_REDACT.MODIFY_EXPRESSION` if you plan to change the `expression` value. Each policy can have only one policy expression. In other words, when you modify the policy expression, you are replacing the existing policy expression with a new policy expression.
  - `DBMS_REDACT.SET_POLICY_DESCRIPTION` if you want to change the description of the policy.
  - `DBMS_REDACT.SET_COLUMN_DESCRIPTION` if you want to change the description of the column.

> **See Also:**
>
> - Parameters Required for DBMS_REDACT.ALTER_POLICY Actions
> - General Syntax of the DBMS_REDACT.ADD_POLICY Procedure for information about the remaining parameters

Parameters Required for DBMS_REDACT.ALTER_POLICY Actions

The `DBMS_REDACT.ALTER_POLICY` procedure provides parameters than can perform various actions, such as adding or modifying a column.

Table 13-11 shows the combinations of these parameters.

Table 13-11 Parameters Required for Various DBMS_REDACT.ALTER_POLICY Actions

Desired Alteration	Parameters to Set
Add or modify a column	• `action` (`DBMS_REDACT.MODIFY_COLUMN`) • `column_name` • `function_type` • `function_parameters` (if necessary) • `regexp*` (if necessary)
Change the policy expression	• `action` (`DBMS_REDACT.MODIFY_EXPRESSION`) • `expression`
Change the description of the policy	• `action` (`DBMS_REDACT.SET_POLICY_DESCRIPTION`) • `policy_description`
Change the description of the column	• `action` (`DBMS_REDACT.SET_COLUMN_DESCRIPTION`) • `column_description`
Drop a column	• `action` (`DBMS_REDACT.DROP_COLUMN`) • `column_name`

Tutorial: Altering an Oracle Data Redaction Policy

You can redact multiple columns in a table or view, with each column having its own redaction setting.

The exercise in this section shows how to modify a Data Redaction policy so that multiple columns are redacted. It also shows how to change the `expression` setting for the policy. To accomplish this, you must run the `DBMS_REDACT.ALTER_POLICY` procedure in stages.

1. Connect as a user who has privileges to create users and grant them privileges.

   For example:

   ```
   CONNECT sec_admin
   Enter password: password
   ```

2. Create the following users:

```
GRANT CREATE SESSION TO dr_admin IDENTIFIED BY password;
GRANT CREATE SESSION TO sales_rep IDENTIFIED BY password;
GRANT CREATE SESSION TO support_rep IDENTIFIED BY password;
```

3. Grant EXECUTE on the DBMS_REDACT PL/SQL package to user dr_admin:

```
GRANT EXECUTE ON DBMS_REDACT TO dr_admin;
```

4. Connect as user OE.

```
CONNECT OE
Enter password: password
```

5. Create and populate a table that contains customer credit card information.

```
CREATE TABLE cust_order_info(
 first_name varchar2(20),
 last_name varchar2(20),
 address varchar2(30),
 city varchar2(30),
 state varchar2(3),
 zip varchar2(5),
 cc_num varchar(19),
 cc_exp varchar2(7));


INSERT INTO cust_order_info VALUES ('Jane','Dough','39 Mockingbird Lane',
'San Francisco', 'CA', 94114, '5105 1051 0510 5100', '10/2018');
INSERT INTO cust_order_info VALUES ('Mary','Hightower','2319 Maple Street',
'Sonoma', 'CA', 95476, '5111 1111 1111 1118', '03/2019');
INSERT INTO cust_order_info VALUES ('Herbert','Donahue','292 Winsome Way',
'San Francisco', 'CA', 94117, '5454 5454 5454 5454', '08/2018');
```

6. Grant the SELECT privilege on the cust_order_info table to the sales_rep and support_rep users.

```
GRANT SELECT ON cust_order_info TO sales_rep, support_rep;
```

7. Connect as user dr_admin.

```
CONNECT dr_admin
Enter password: password
```

8. Create and enable policy to redact the credit card number.

```
BEGIN DBMS_REDACT.ADD_POLICY(
    object_schema              => 'oe',
    object_name                => 'cust_order_info',
    column_name                => 'cc_num',
    policy_name                => 'redact_cust_cc_info',
    function_type              => DBMS_REDACT.REGEXP,
    function_parameters        => NULL,
    expression                 => '1=1',
    regexp_pattern             => DBMS_REDACT.RE_PATTERN_CCN,
    regexp_replace_string      => DBMS_REDACT.RE_REDACT_CCN,
    regexp_position            => NULL,
    regexp_occurrence          => NULL,
    regexp_match_parameter     => NULL,
    policy_description         => 'Partially redacts credit card info',
    column_description         => 'cc_num_number lists credit card numbers');
END;
/
```

9. Modify the policy to include redaction of the expiration date.

```
BEGIN DBMS_REDACT.ALTER_POLICY(
    object_schema     => 'oe',
    object_name       => 'cust_order_info',
    policy_name       => 'redact_cust_cc_info',
    action            => DBMS_REDACT.ADD_COLUMN,
    column_name       => 'cc_exp',
    function_type     => DBMS_REDACT.RANDOM,
    expression        => '1-1');
END;
/
```

10. Modify the policy again, to use a condition so that the `sales_rep` user views the redacted values and the `support_rep` user views the actual data.

```
BEGIN
   DBMS_REDACT.ALTER_POLICY(
   object_schema     => 'oe',
   object_name       => 'cust_order_info',
   policy_name       => 'redact_cust_cc_info',
   action            => DBMS_REDACT.MODIFY_EXPRESSION,
   expression        => 'SYS_CONTEXT(''USERENV'',''SESSION_USER'') =
''SALES_REP''');
END;
/
```

11. To test the policy, have the two users query the `cust_order_info` table.

```
CONNECT support_rep
Enter password: password

SELECT cc_num, cc_exp FROM OE.cust_order_info;

CC_NUM               CC_EXP
-------------------  -------
5105 1051 0510 5100  10/2018
5111 1111 1111 1118  03/2019
5454 5454 5454 5454  08/2018
```

User `support_rep` can view the actual data.

```
CONNECT sales_rep
Enter password: password

SELECT cc_num, cc_exp FROM OE.cust_order_info;

CC_NUM             CC_EXP
---------------    -------
************5100   lST=033
************1119   OZA.w4C
************5454   B(9+;O1
```

The actual data is redacted using for user `sales_rep`.

12. Alter the `cust_order_info` to include a condition so that only `support_rep` can see the redacted data but `sales_rep` cannot.

```
CONNECT dr_admin
Enter password: password

BEGIN
   DBMS_REDACT.ALTER_POLICY(
   object_schema      => 'oe',
   object_name        => 'cust_order_info',
```

```
   policy_name        => 'redact_cust_cc_info',
   action             => DBMS_REDACT.MODIFY_EXPRESSION,
   expression         => 'SYS_CONTEXT(''USERENV'',''SESSION_USER'') =
''SUPPORT_REP''');
END;
/
```

13. Have the users test the policy again.

```
CONNECT support_rep
Enter password: password

SELECT cc_num, cc_exp FROM OE.cust_order_info;

CC_NUM             CC_EXP
----------------   -------
************5100   1^XMF~`
************1119   qz+9=#S
************5454   *KCaUkm
```

User `support_rep` can no longer view the actual data; it is now redacted.

```
CONNECT sales_rep
Enter password: password

SELECT cc_num, cc_exp FROM OE.cust_order_info;

CC_NUM               CC_EXP
-------------------  -------
5105 1051 0510 5100  10/2018
5111 1111 1111 1118  03/2019
5454 5454 5454 5454  08/2018
```

User `sales_rep` now can view the actual data.

14. If you do not need the components of this tutorial, then you can remove them as follows:

```
CONNECT dr_admin
Enter password: password

BEGIN
  DBMS_REDACT.DROP_POLICY (
    object_schema  => 'oe',
    object_name    => 'cust_order_info',
    policy_name    => 'redact_cust_cc_info');
END;
/

CONNECT sec_admin
Enter password: password

DROP USER dr_admin;
DROP USER sales_rep;
DROP USER support_rep;

CONNECT OE
Enter password: password

DROP TABLE cust_order_info;
```

Redacting Multiple Columns

You can redact more than one column in a Data Redaction policy.

- Adding Columns to a Data Redaction Policy for a Single Table or View
  You can redact columns of different data types, using different redaction types, for one table or view.
- Example: Redacting Multiple Columns
  The `DBMS_REDACT.ALTER_POLICY` procedure can redact multiple columns.

Adding Columns to a Data Redaction Policy for a Single Table or View

You can redact columns of different data types, using different redaction types, for one table or view.

1. Create the policy for the first column that you want to redact.
2. Use the `DBMS_REDACT.ALTER_POLICY` procedure to add the next column to the policy.

   As necessary, set the `action`, `column_name`, `function_type`, and `function_parameters` (or the parameters that begin with `regexp_`) parameters to define the redaction for the new column, but do not change the `object_schema`, `object_name`, `policy_name`, or `expression` parameters. Each redacted column continues to have the same redaction parameters that were used to create it.

Example: Redacting Multiple Columns

The `DBMS_REDACT.ALTER_POLICY` procedure can redact multiple columns.

Example 13-12 shows how to add a column to an existing Data Redaction policy. In this example, the `action` parameter specifies that a new column must be added, using `DBMS_REDACT.ADD_COLUMN`. The name of the new column, `card_num`, is set by the `column_name` parameter.

Example 13-12 Adding a Column to a Data Redaction Policy

```
BEGIN
 DBMS_REDACT.ALTER_POLICY(
  object_schema       => 'mavis',
  object_name         => 'cust_info',
  policy_name         => 'redact_cust_user_ids',
  action              => DBMS_REDACT.ADD_COLUMN,
  column_name         => 'card_num',
  function_type       => DBMS_REDACT.FULL,
  function_parameters => '',
  expression          => 'SYS_CONTEXT(''SYS_SESSION_ROLES'',''ADM'') =
''TRUE''');
END;
/
```

Disabling and Enabling an Oracle Data Redaction Policy

You can disable and then reenable Oracle Data Redactions policies as necessary.

- Disabling an Oracle Data Redaction Policy
  The DBMS_REDACT.DISABLE_POLICY procedure disables Oracle Data Redaction policies.
- Enabling an Oracle Data Redaction Policy
  The DBMS_REDACT.ENABLE_POLICY procedure enables Oracle Data Redaction policies.

Disabling an Oracle Data Redaction Policy

The DBMS_REDACT.DISABLE_POLICY procedure disables Oracle Data Redaction policies.

You can find the names of existing Data Redaction policies and whether they are enabled by querying the POLICY_NAME and ENABLE columns of the REDACTION_POLICIES view. However, as long as the policy still exists, you cannot create another policy for that table or view, even if the original policy is disabled. In other words, if you want to create a different policy on the same table column, then you must drop the first policy before you can create and use the new policy.

- To disable a Data Redaction policy, run the DBMS_REDACT.DISABLE_POLICY procedure, using the following syntax:

```
DBMS_REDACT.DISABLE_POLICY (
   object_schema       IN VARCHAR2 DEFAULT NULL,
   object_name         IN VARCHAR2,
   policy_name         IN VARCHAR2);
```

  In this specification:

  – object_schema: Specifies the schema of the object on which the Data Redaction policy will be applied. If you omit this setting (or enter NULL), then Oracle Database uses the name of the current schema.
  – object_name: Specifies the name of the table or view to be used for the Data Redaction policy.
  – policy_name: Specifies the name of the policy to be disabled.

Example 13-13 shows how to disable a Data Redaction policy.

Example 13-13 Disabling a Data Redaction Policy

```
BEGIN
  DBMS_REDACT.DISABLE_POLICY (
    object_schema  => 'mavis',
    object_name    => 'cust_info',
    policy_name    => 'redact_cust_user_ids');
END;
/
```

Enabling an Oracle Data Redaction Policy

The DBMS_REDACT.ENABLE_POLICY procedure enables Oracle Data Redaction policies.

Immediately after you create a new policy, you do not need to enable it; the creation process handles that for you. To find the names of existing Data Redaction policies and whether they are enabled, you can query the POLICY_NAME and ENABLE columns of the REDACTION_POLICIES view. After you run the procedure to enable the policy, the enablement takes effect immediately.

- To enable a Data Redaction policy, run the `DBMS_REDACT.ENABLE_POLICY` procedure, using the following syntax.

```
DBMS_REDACT.ENABLE_POLICY (
   object_schema       IN VARCHAR2 DEFAULT NULL,
   object_name         IN VARCHAR2,
   policy_name         IN VARCHAR2);
```

  In this specification:

  - `object_schema`: Specifies the schema of the object on which the Data Redaction policy will be applied. If you omit this setting (or enter `NULL`), then Oracle Database uses the name of the current schema.
  - `object_name`: Specifies the name of the table or view to be used for the Data Redaction policy.
  - `policy_name`: Specifies the name of the policy to be enabled.

Example 13-14 shows how to enable a Data Redaction policy.

Example 13-14 Enabling a Data Redaction Policy

```
BEGIN
  DBMS_REDACT.ENABLE_POLICY (
    object_schema  => 'mavis',
    object_name    => 'cust_info',
    policy_name    => 'redact_cust_user_ids');
END;
/
```

Dropping an Oracle Data Redaction Policy

The `DBMS_REDACT.DROP_POLICY` procedure drops Oracle Data Redaction policies.

You can drop an Oracle Data Redaction policy whether it is enabled or disabled. You can find the names of existing Data Redaction policies, by querying the `POLICY_NAME` column of the `REDACTION_POLICIES` view. When you drop a table or view that is associated with an Oracle Data Redaction policy, the policy is automatically dropped. As a best practice, drop the policy first, and then drop the table or view afterward. See Dropped Oracle Data Redaction Policies When the Recycle Bin Is Enabled for more information.

- To drop a Data Redaction policy, run the `DBMS_REDACT.DROP_POLICY` procedure.

  Use the following syntax:

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

  In this specification:

  - `object_schema`: Specifies the schema of the object to which the Data Redaction policy applies. If you omit this setting (or enter `NULL`), then Oracle Database uses the name of the current schema.
  - `object_name`: Specifies the name of the table or view to be used for the Data Redaction policy.
  - `policy_name`: Specifies the name of the policy to be dropped.

After you run the `DBMS_REDACT.DROP_POLICY` procedure, the drop takes effect immediately.

Example 13-15 shows how to drop a Data Redaction policy.

Example 13-15 Dropping a Data Redaction Policy

```
BEGIN
  DBMS_REDACT.DROP_POLICY (
    object_schema  => 'mavis',
    object_name    => 'cust_info',
    policy_name    => 'redact_cust_user_ids');
END;
/
```

Tutorial: SQL Expressions to Build Reports with Redacted Values

SQL expressions can be used to build reports based on columns that have Oracle Data Redaction policies defined on them.

The values used in the SQL expression will be redacted. This redaction occurs in such a way that the redaction takes place before the SQL expression is evaluated: the result value that is displayed in the report is the end result of the evaluated SQL expression over the redacted values, rather than the redacted result of the SQL expression as a whole.

1. Create the following Data Redaction policy for the `HR.EMPLOYEES` table.

   This policy will replace the first 4 digits of the value from the `SALARY` column with the number `9` and the first digit of the value from the `COMMISSION_PCT` column with a `9`.

   ```
   BEGIN
    DBMS_REDACT.ADD_POLICY(
      object_schema          => 'HR',
      object_name            => 'EMPLOYEES',
      column_name            => 'SALARY',
      column_description     => 'emp_sal_comm shows employee salary and
   commission',
      policy_name            => 'redact_emp_sal_comm',
      policy_description     => 'Partially redacts the emp_sal_comm column',
      function_type          => DBMS_REDACT.PARTIAL,
      function_parameters    => '9,1,4',
      expression             => '1=1');
   END;
   /
   BEGIN
    DBMS_REDACT.ALTER_POLICY(
      object_schema          => 'HR',
      object_name            => 'EMPLOYEES',
      policy_name            => 'redact_emp_sal_comm',
      action                 => DBMS_REDACT.ADD_COLUMN,
      column_name            => 'COMMISSION_PCT',
      function_type          => DBMS_REDACT.PARTIAL,
      function_parameters    => '9,1,1',
      expression             => '1=1');
   END;
   /
   ```

2. Log in to the HR schema and then run the following report.

   This report will use the SQL expression (SALARY + COMMISSION_PCT) to combine the employees' salaries and commissions.

```
SELECT (SALARY + COMMISSION_PCT) total_emp_compensation
FROM HR.EMPLOYEES
WHERE DEPARTMENT_ID = 80;

TOTAL_EMP_COMPENSATION
----------------------
                9999.9
               9999.95
              99990.95
...
```

3. Use SQL expressions for the report, including concatenation.

   For example:

```
SELECT 'Employee ID '          || EMPLOYEE_ID ||
       ' has a salary of '     || SALARY ||
       ' and a commission of ' || COMMISSION_PCT || '.'
detailed_emp_compensation
FROM HR.EMPLOYEES
WHERE DEPARTMENT_ID = 80
ORDER BY EMPLOYEE_ID;

DETAILED_EMP_COMPENSATION
-------------------------------------------------------------
Employee ID 150 has a salary of 99990 and a commission of .9.
Employee ID 151 has a salary of 9999 and a commission of .95.
Employee ID 152 has a salary of 9999 and a commission of .95.
...
```

4. Connect the user who created the redact_emp_sal_comm Data Redaction policy and then run the following statement to drop the policy.

```
BEGIN
  DBMS_REDACT.DROP_POLICY (
    object_schema => 'HR',
    object_name   => 'EMPLOYEES',
    policy_name   => 'redact_emp_sal_comm');
END;
/
```

Oracle Data Redaction Policy Data Dictionary Views

Oracle Database provides data dictionary views that list information about Data Redaction policies.

Before you can query these views, you must be granted the SELECT_CATALOG_ROLE role.

Table 13-12 lists the Data Redaction data dictionary views.

Table 13-12 Data Redaction Views

View	Description
REDACTION_COLUMNS	Describes all of the redacted columns in the database, providing the the owner of the table or view within which the column resides, the object name, the column name, the type of redaction function, the parameters to the redaction function (if any), and a description of the redaction policy. If a policy expression has been created, displays the default object-wide policy expression's SQL expression.
REDACTION_EXPRESSIONS	Displays the names of existing policy expressions and their SQL expressions
REDACTION_POLICIES	Describes all of the data redaction policies in the database. It includes information about the object owner, object name, policy name, policy expression, whether the policy is enabled, and a description of the Data Redaction policy.
REDACTION_VALUES_FOR_TYPE_FULL	Shows the current redaction values for Data Redaction policies that use full redaction

14

Managing Oracle Data Redaction Policies in Oracle Enterprise Manager

Oracle Enterprise Manager Cloud Control (Cloud Control) can manage Oracle Data Redaction policies and formats.

- About Using Oracle Data Redaction in Oracle Enterprise Manager
  Oracle Enterprise Manager Cloud Control provides an unified user interface for creating and managing Oracle Data Redaction policies.
- Oracle Data Redaction Workflow
  First, you should create sensitive column types and formats if necessary, and then create the Oracle Data Redaction policy afterward.
- Management of Sensitive Column Types in Enterprise Manager
  A sensitive column type categorizes table column sensitive information into a sensitive information type, such as credit card numbers.
- Managing Oracle Data Redaction Formats Using Enterprise Manager
  Oracle Data Redaction provides redaction formats to be used directly within a redaction policy to redact data.
- Managing Oracle Data Redaction Policies Using Enterprise Manager
  You can create, edit, view, and delete Oracle Data Redaction policies in Enterprise Manager Cloud Control.
- Managing Named Data Redaction Policy Expressions Using Enterprise Manager
  You can manage Oracle Data Redaction policy expressions in Enterprise Manager Cloud Control.

About Using Oracle Data Redaction in Oracle Enterprise Manager

Oracle Enterprise Manager Cloud Control provides an unified user interface for creating and managing Oracle Data Redaction policies.

You can do the following:

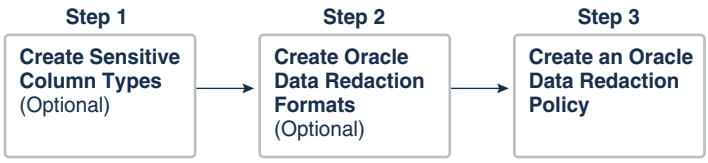
- Create and manage custom Oracle Data Redaction formats, which were previously known as Data Redaction shortcuts. (This functionality is not available from the command line.)
- Create and manage sensitive column types directly from the Oracle Data Redaction pages. While you create a Data Redaction policy, Cloud Control uses sensitive column types to obtain the Oracle Data Redaction formats that are relevant to the column that you are redacting.

> **Note:**
>
> Ensure that you have the latest plug-in for Oracle Enterprise Manager. If you have the Oracle Database plug-in release 13.1.1.0.0, then you can create named Data Redaction policy expressions in Oracle Enterprise Manager.

Oracle Data Redaction Workflow

First, you should create sensitive column types and formats if necessary, and then create the Oracle Data Redaction policy afterward.

The following figure illustrates this process:



1. (Optional) If you want to map the database columns (that contain the data that you want to redact) to new sensitive column types, then create the required sensitive column types as described in Management of Sensitive Column Types in Enterprise Manager.
2. (Optional) If you want to redact the data (present in a particular database column) using a custom redaction format, then create the required redaction format as described in Creating a Custom Oracle Data Redaction Format Using Enterprise Manager.
3. Create an Oracle Data Redaction policy for the required database, as described in Creating an Oracle Data Redaction Policy Using Enterprise Manager.

> **Note:**
>
> When you create an Oracle Data Redaction policy, it is enabled by default. For information on how to disable an enabled redaction policy, see Enabling or Disabling an Oracle Data Redaction Policy in Enterprise Manager.

Management of Sensitive Column Types in Enterprise Manager

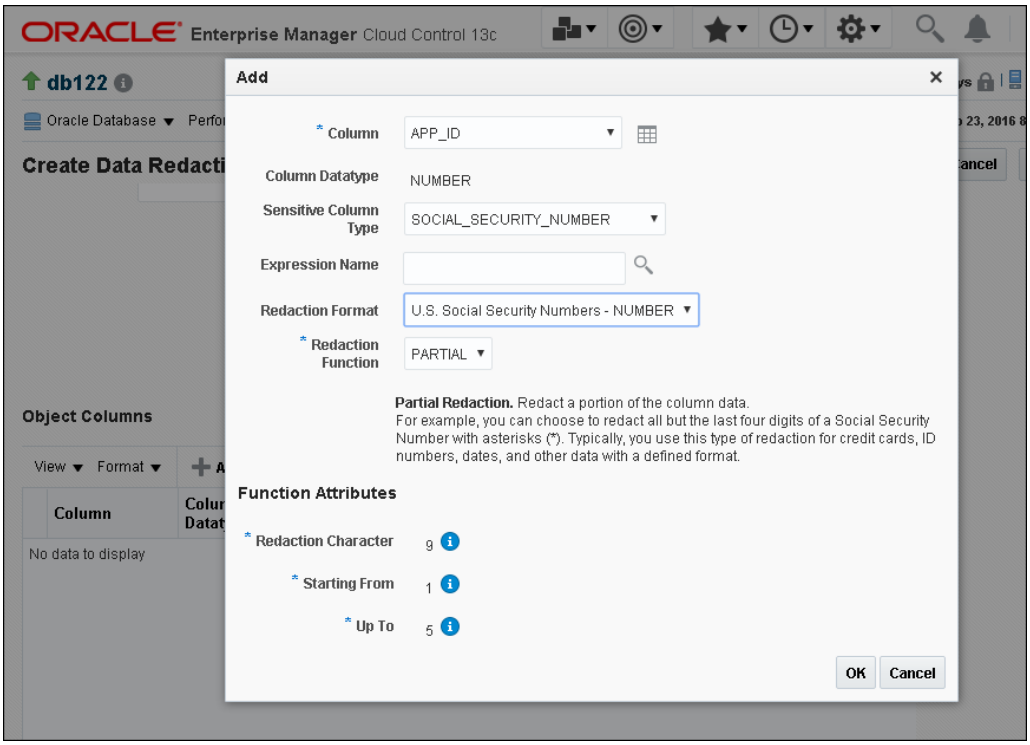
A sensitive column type categorizes table column sensitive information into a sensitive information type, such as credit card numbers.

Sensitive column types use a combination of the column name, column comments, and the data pattern defined using a regular expression to tag a column to a particular sensitive information type.

While you create Oracle Data Redaction policies, redaction formats are filtered on the basis of the chosen sensitive column type, thus saving time and effort. For example, if the database table column that you want to redact contains U.S. Social Security numbers, and you select the `SOCIAL_SECURITY_NUMBER` sensitive column type for the column while adding it to the Oracle Data Redaction policy, the default redaction formats that you can use to redact the column data are filtered, and only the relevant redaction formats are displayed.

Figure 14-1 illustrates the filtering of Oracle Data Redaction formats based on sensitive column types.

Figure 14-1 Oracle Data Redaction Formats Filtered on the Basis of Sensitive Column Types

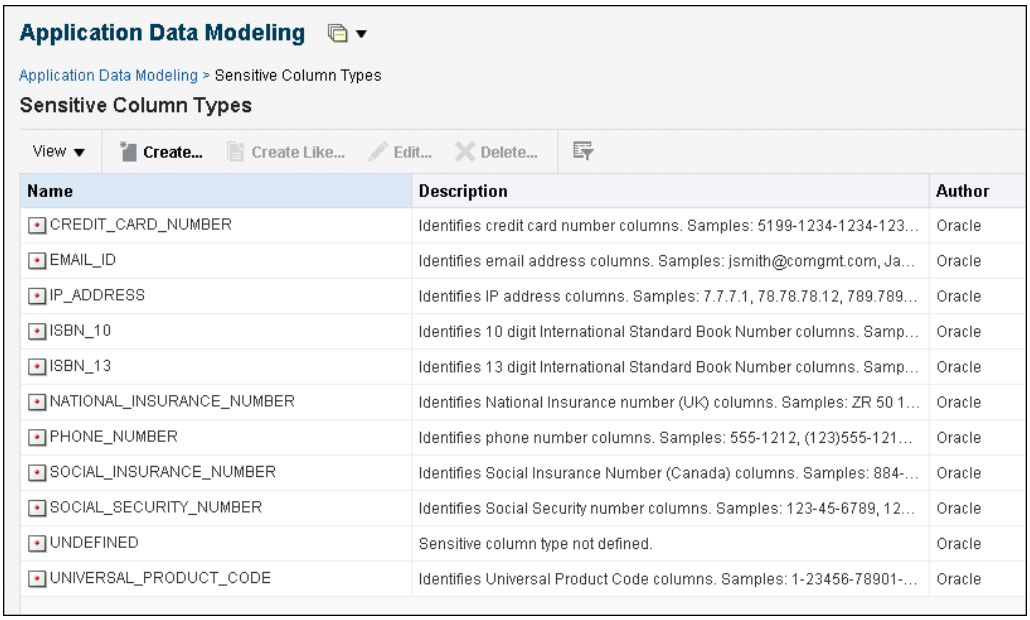


> **Note:**
>
> This functionality is available only if you have the Enterprise Manager for Oracle Database plug-in 12.1.0.7 or later deployed in your system.

As part of the Application Data Modeling feature, Oracle provides a number of default sensitive column types that a database column can be mapped to.

Figure 14-2 displays some of the default sensitive column types. To access this page, click **Manage Sensitive Column Types** on the Data Redaction Formats page.

Figure 14-2 Default Sensitive Column Types



If none of the default sensitive column types are suitable for the database column that contains the data that you want to redact, you can create a new sensitive column type, or create a sensitive column type that is based on an existing sensitive column type.

Managing Oracle Data Redaction Formats Using Enterprise Manager

Oracle Data Redaction provides redaction formats to be used directly within a redaction policy to redact data.

- About Managing Oracle Data Redaction Formats Using Enterprise Manager
  The Oracle Data Redaction formats are used for commonly redacted data, such as ID numbers, credit cards, or phone numbers.
- Creating a Custom Oracle Data Redaction Format Using Enterprise Manager
  You can create and save custom Oracle Data Redaction formats using Enterprise Manager Cloud Control.
- Editing a Custom Oracle Data Redaction Format Using Enterprise Manager
  You can edit custom Oracle Data Redaction formats using Enterprise Manager Cloud Control, but not in SQL*Plus.
- Viewing Oracle Data Redaction Formats Using Enterprise Manager
  Enterprise Manager Cloud Control displays the details of the Oracle-supplied and custom Oracle Data Redaction formats.
- Deleting a Custom Oracle Data Redaction Format Using Enterprise Manager
  You can delete custom Oracle Data Redaction formats using Enterprise Manager Cloud Control.

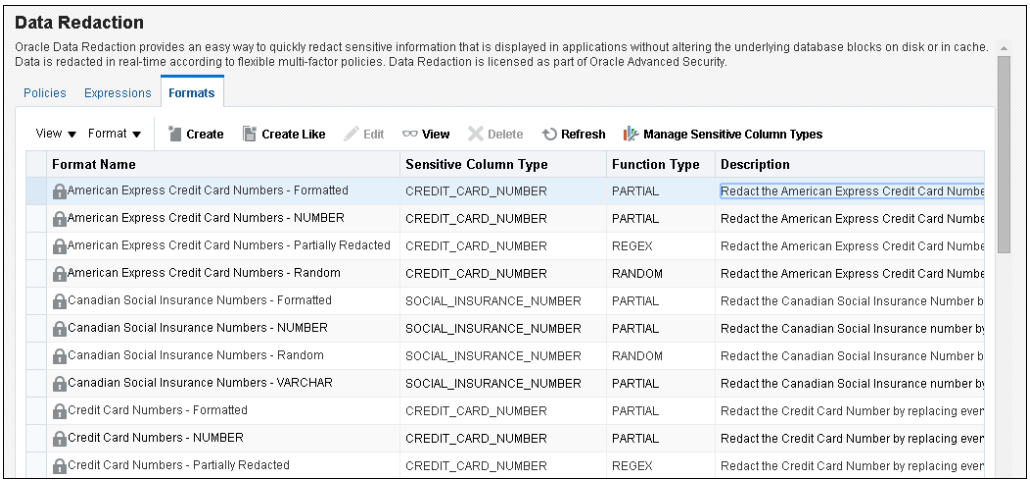
About Managing Oracle Data Redaction Formats Using Enterprise Manager

The Oracle Data Redaction formats are used for commonly redacted data, such as ID numbers, credit cards, or phone numbers.

You can use several default Oracle Data Redaction formats (previously known as Oracle Data Redaction templates). As an example of the Oracle Data Redaction formats, a set of Social Security number formats enable you to quickly designate ways to redact Social Security numbers, such as redacting the first five numbers of the Social Security number.

Figure 14-3 displays the default Oracle Data Redaction formats.

Figure 14-3 Default Oracle Data Redaction Formats



Each default Oracle Data Redaction format consists of a specific redaction function that determines the redacted output when the redaction format is used in an Oracle Data Redaction policy. For example, the `Credit Card Numbers - NUMBER` default redaction format replaces the first twelve digits of the column data with the digit 0, when it is used in an Oracle Data Redaction policy. That is, if the column data is `5555555555554444`, the redacted output will be `0000000000004444`.

If you have deployed the Enterprise Manager for Oracle Database plug-in 12.1.0.7 or higher on your system, then you can also create and save custom redaction formats, which you can then use in your redaction policies.

Creating a Custom Oracle Data Redaction Format Using Enterprise Manager

You can create and save custom Oracle Data Redaction formats using Enterprise Manager Cloud Control.

1. Log into Oracle Enterprise Manager Cloud Control as either user `SYSTEM` or `SYSMAN`.

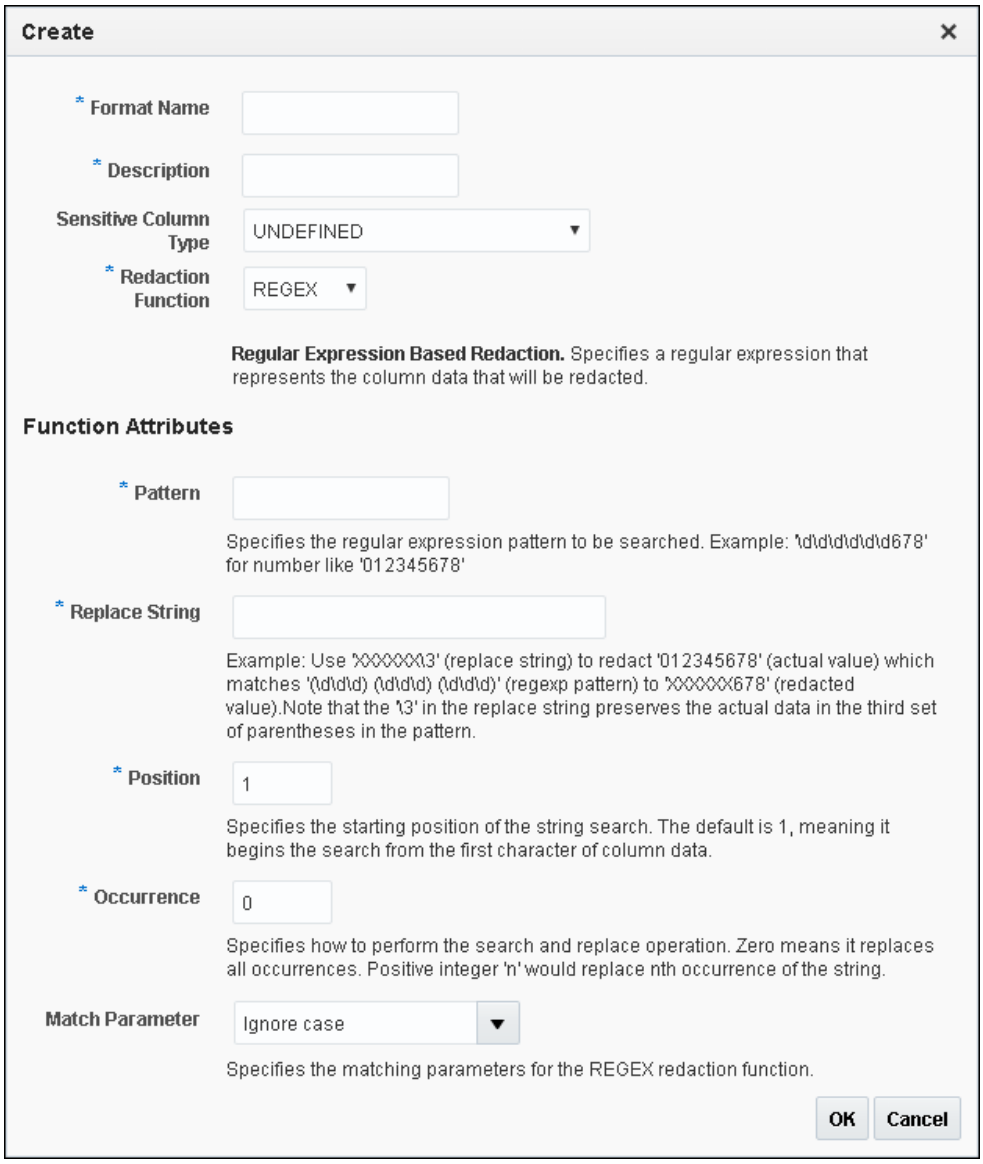
The URL is as follows:

`https://host:port/em`

2. From the **Targets** menu, select **Databases**.
3. Select **Search List**, then click the name of a database target.
4. On the home page of the database target, from the **Security** menu, select **Data Redaction**.
5. Log in to the database, if you are prompted to do so.

   Ensure that you log in to the database as a user that has the `EXECUTE` privilege on the `DBMS_REDACT` PL/SQL package.
6. Select the **Formats** tab.
7. Do one of the following:
   - To create a new redaction format, click **Create**.
   - To create a redaction format that is based on a default format, select the format and then click **Create Like**.

   If you select **Create**, then the following dialog box appears:



8. Provide a name and a description for the redaction format that you want to create.

   If you want to map the redaction format to a particular sensitive column type (such that the created redaction format can be used to redact the data of a column that is associated with the sensitive column type), then select a value for **Sensitive Column Type**.

   Select the function that the format should use to redact the column data. For **Redaction Function**, select as follows:

   - **FULL** if the format should redact the entire column data.
   - **PARTIAL** if the format should redact only a part of the column data. Ensure that you provide the function attributes, as well as the data type that you want to use the redaction format for.
   - **REGEX** if the format should redact data based on a regular expression. Ensure that you provide the function attributes.
   - **RANDOM** if the format should redact data in a random manner, using randomly generated values

- **NONE** if the format will be used to only test the definition of a redaction policy, and not redact any column data

9. Click **OK** to create and save the custom redaction format.

   This format now can be used to create a redaction policy.

Related Topics

- Oracle Data Redaction Features and Capabilities
  Oracle Data Redaction provides a variety of ways to redact different types of data.
- Creating an Oracle Data Redaction Policy Using Enterprise Manager
  You can create an Oracle Data Redaction policy using Enterprise Manager Cloud Control.

Editing a Custom Oracle Data Redaction Format Using Enterprise Manager

You can edit custom Oracle Data Redaction formats using Enterprise Manager Cloud Control, but not in SQL*Plus.

1. Log into Oracle Enterprise Manager Cloud Control as either user `SYSTEM` or `SYSMAN`.

   The URL is as follows:

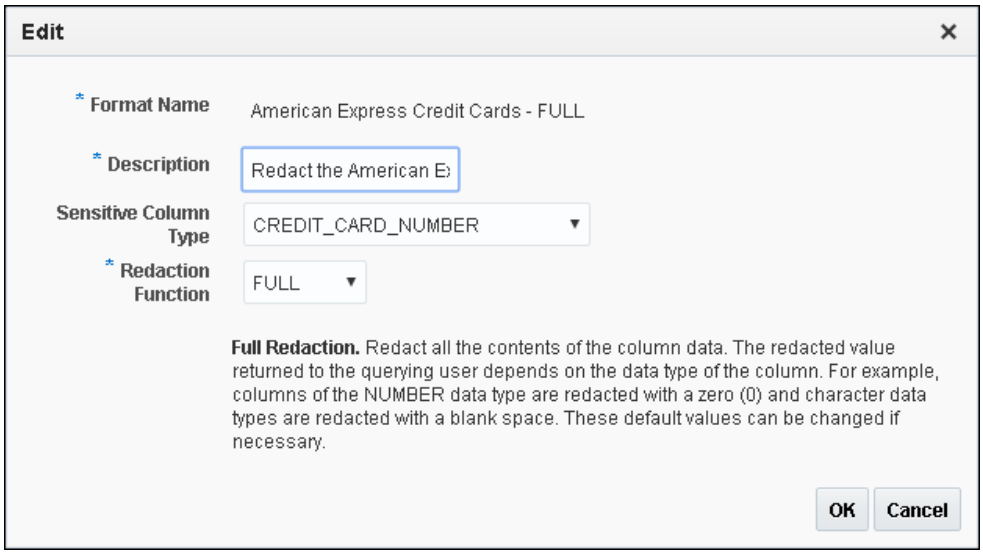
   `https://host:port/em`

2. From the **Targets** menu, select **Databases**.
3. Select **Search List**, then click the name of a database target.
4. On the home page of the database target, from the **Security** menu, select **Data Redaction**.
5. Log in to the database, if you are prompted to do so.

   Ensure that you log in to the database as a user that has the `EXECUTE` privilege on the `DBMS_REDACT` PL/SQL package.

6. Select the **Formats** tab.
7. Select the custom redaction format that you want to edit, then click **Edit.**

   A dialog box similar to the following appears:



8. (Optional) Choose to edit the format description, sensitive column type, redaction function, and the redaction function attributes.
9. Click **OK** to save the edited format.

Viewing Oracle Data Redaction Formats Using Enterprise Manager

Enterprise Manager Cloud Control displays the details of the Oracle-supplied and custom Oracle Data Redaction formats.

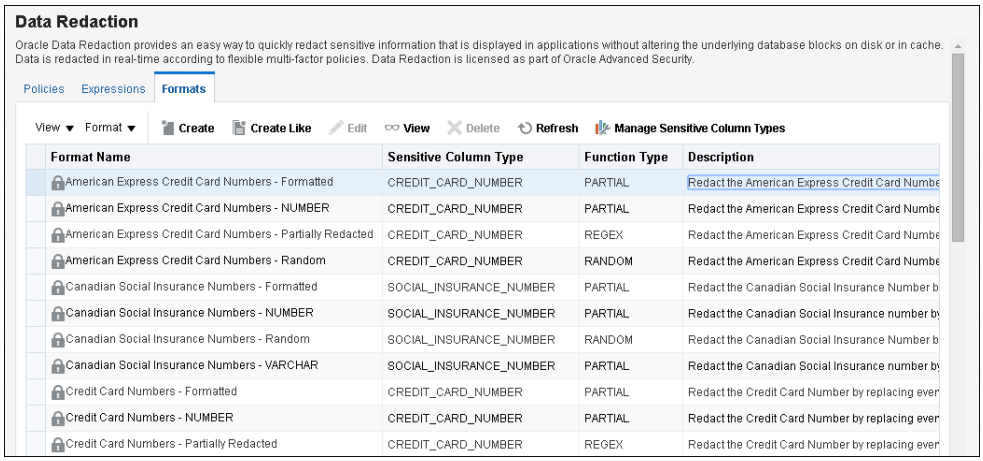
1. Log into Oracle Enterprise Manager Cloud Control as either user `SYSTEM` or `SYSMAN`.

   The URL is as follows:

   `https://host:port/em`
2. From the **Targets** menu, select **Databases**.
3. Select **Search List**, then click the name of a database target.
4. On the home page of the database target, from the **Security** menu, select **Data Redaction**.
5. Log in to the database, if you are prompted to do so.

   Ensure that you log in to the database as a user that has the `EXECUTE` privilege on the `DBMS_REDACT` PL/SQL package.
6. Select the **Formats** tab.

   The Data Redaction Formats page appears, similar to the following page.



7. Select the required redaction format, then click **View**.

Deleting a Custom Oracle Data Redaction Format Using Enterprise Manager

You can delete custom Oracle Data Redaction formats using Enterprise Manager Cloud Control.

You can only delete custom Oracle Data Redaction formats, and not the redaction formats that are provided by Oracle.

1. Log into Oracle Enterprise Manager Cloud Control as either user `SYSTEM` or `SYSMAN`.

   The URL is as follows:

   `https://host:port/em`

2. From the **Targets** menu, select **Databases**.
3. Select **Search List**, then click the name of a database target.
4. On the home page of the database target, from the **Security** menu, select **Data Redaction**.
5. Log in to the database, if you are prompted to do so.

   Ensure that you log in to the database as a user that has the `EXECUTE` privilege on the `DBMS_REDACT` PL/SQL package.

6. Select the **Formats** tab.
7. Select the custom redaction format that you want to delete, and then click **Delete**.
8. In the Confirmation dialog box, click **Yes** or **No**.

Managing Oracle Data Redaction Policies Using Enterprise Manager

You can create, edit, view, and delete Oracle Data Redaction policies in Enterprise Manager Cloud Control.

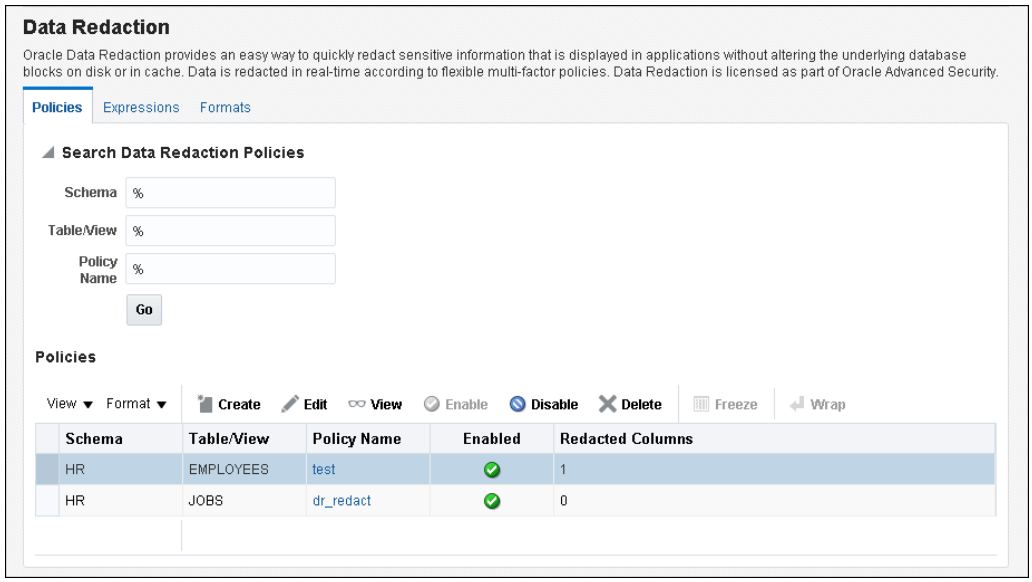
About Managing Oracle Data Redaction Policies Using Enterprise Manager

Use the Data Redaction page in Cloud Control to manage Oracle Data Redaction policies.

To redact the data present in a particular database table or view column, you must create an Oracle Data Redaction policy. Data is redacted using a redaction format that is specified by the Oracle Data Redaction policy. To redact data, you can use any of the Oracle-supplied redaction formats, or create and use a custom redaction format. If the table or view column that contains the data that you want to redact is mapped to a sensitive column type, Oracle uses the mapping to recommend suitable redaction formats for the data. Thus, Oracle Data Redaction policies encapsulate database schemas, database table and view columns, sensitive column types, and Oracle Data Redaction formats.

Figure 14-4 shows the Data Redaction page, which enables you to create and manage Oracle Data Redaction policies in Cloud Control.

Figure 14-4 Oracle Data Redaction Policies Page



Creating an Oracle Data Redaction Policy Using Enterprise Manager

You can create an Oracle Data Redaction policy using Enterprise Manager Cloud Control.

1. Log into Oracle Enterprise Manager Cloud Control as either user `SYSTEM` or `SYSMAN`.

   The URL is as follows:

   `https://host:port/em`

2. From the **Targets** menu, select **Databases**.
3. Select **Search List**, then click the name of a database target for which you want to create an Oracle Data Redaction policy.
4. On the home page of the database target, from the **Security** menu, select **Data Redaction**.
5. Log in to the database, if you are prompted to do so.

   Ensure that you log in to the database as a user that has the `EXECUTE` privilege on the `DBMS_REDACT` PL/SQL package.

6. In the Policies section of the Policies tab, select **Create**.

   If this is the first time that you are creating a Data Redaction policy, then the Data Redaction: Set up for enabling column sensitive type discovery dialog box appears. This feature enables the use of column sensitive type discovery for Data Redaction policies.

   To accomplish this, Enterprise Manager creates the `GET_COL_DATA_SENSITIVE_TYPES` procedure in the `DBSNMP` schema. To perform a sensitive type discovery for a selected column while creating an Oracle Data Redaction policy, a user must have the `EXECUTE` privilege on the `DBSNMP.GET_COL_DATA_SENSITIVE_TYPES` procedure. If the database is protected

by Oracle Database Vault, then ensure that any users who must create Data Redaction policies are participants to realms that protect the `DBSNMP` schema.

7. If the Data Redaction: Set up for enabling column sensitive type discovery dialog box appears and if the current login user does not have the correct requirements, select a credential of a user who has the `EXECUTE` privilege on `DBSNMP.GET_COL_DATA_SENSITIVE_TYPES`. Then click `OK`.
8. On the Create Data Redaction Policy page, enter the following information:
   - **Schema**: Enter (or search for) the name of the schema that contains the data you want to redact.
   - **Table/View**: Enter (or search for) the table or field that contains the column you want to redact.
   - **Policy Name**: Enter a for the policy, such as `emp_wages_pol`.
   - **Default Expression**: Enter the default expression. The default setting is 1=1, which means that the policy always will be enforced. If you are not familiar with the components of a policy expression, then click the pencil icon beside the **Policy Expression** field to use Policy Expression Builder. Select **Policy is in effect when**, select the required conditions, then click **Add**. Click **Edit** if you want to edit the policy expression manually. After building the required policy expression, click **OK**. The Policy Expression Builder appears as follows:

9. In the Object Columns section, click **Add** to add a table or view column to the redaction policy.

   A dialog box similar to the following appears:

The redaction policy is applied only on the table or view columns that are added to it.

10. From the **Column** menu, select the table or view column to which you want to apply the redaction policy.

    To the right of the **Column** menu is an icon that you can click to view the contents of the selected column.

    If the column contains sensitive data and has been mapped to a sensitive column type, then from the **Sensitive Column Type** menu, select the sensitive column type that it has been mapped to. If the search pattern in the **Sensitive Column Type** menu matches, then the sensitive column type is selected by default. For example, for a column listing credit card numbers, if there is a match, then the menu will list **Undefined** and **CREDIT_CARD_TYPE**. If there is no sensitive column type created, then the default **Sensitive Column Type** menu listing is only **Undefined**.

11. From the **Redaction Format** menu, select the redaction format that you want to use.

    The drop-down list is populated with the Oracle Database-supplied redaction formats, as well as the custom redaction formats that you have created and saved. If you do not want to use a pre-defined redaction format (that is, an Oracle-Database supplied redaction format, or a custom redaction format that you have created), and instead want to specify the redaction details while creating the redaction policy, select **CUSTOM** for **Redaction Format**.

The Add dialog box adjusts to accommodate the type of redaction format and function that you select. For example, if you select the **CUSTOM** redaction format and the **REGEX** redaction function, then the Function Attributes region appears in the dialog box.

12. From the **Redaction Function** menu, select the function that you want to use to redact the column data.

    Select **FULL** if you want to redact the entire column data, **PARTIAL** if you want to redact only a part of the column data, **REGEX** if you want to redact the column data based on a regular expression, **RANDOM** if you want to redact the column data in a random manner, using randomly generated values, or **NONE** if you only want to test the definition of the redaction policy, and not redact any column data. Note that all the redaction functions may not be applicable for a particular redaction format. The drop-down list displays only the redaction functions that are applicable for the selected redaction format.

    If you selected **CUSTOM** for **Redaction Format** in the previous step, and **PARTIAL** or **REGEX** for **Redaction Function,** ensure that you specify the function attributes.

13. Click **OK**.

14. Repeat these steps starting with Step 8 for all the columns that you want to add to the redaction policy.

15. On the Create Data Redaction Policy page, click **OK** to create the data redaction policy.

    When you create an Oracle Data Redaction policy, it is enabled by default.

Related Topics

- Creating a Custom Oracle Data Redaction Format Using Enterprise Manager
  You can create and save custom Oracle Data Redaction formats using Enterprise Manager Cloud Control.
- Oracle Data Redaction Features and Capabilities
  Oracle Data Redaction provides a variety of ways to redact different types of data.
- Enabling or Disabling an Oracle Data Redaction Policy in Enterprise Manager
  An Oracle Data Redaction policy is executed at run time only if it is enabled. When you create an Oracle Data Redaction policy, it is enabled by default.

Editing an Oracle Data Redaction Policy Using Enterprise Manager

You can edit an Oracle Data Redaction policy using Enterprise Manager Cloud Control.

1. Log into Oracle Enterprise Manager Cloud Control as either user `SYSTEM` or `SYSMAN`.

   The URL is as follows:

   `https://host:port/em`

2. From the **Targets** menu, select **Databases**.

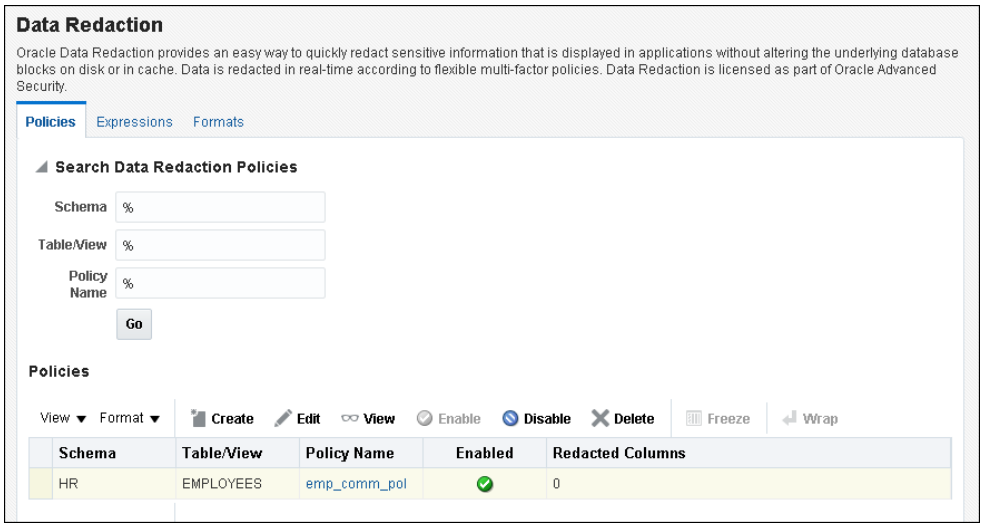
3. Select **Search List**, then search for and click the name of the database target for which the Oracle Data Redaction policy that you want to edit was created.

4. On the home page of the database target, from the **Security** menu, select **Data Redaction**.

5. Log in to the database, if you are prompted to do so.

   Ensure that you log in to the database as a user that has the `EXECUTE` privilege on the `DBMS_REDACT` PL/SQL package.

6. In the Policies section of the **Policies** tab, select the redaction policy that you want to edit, then click **Edit.**.



7. On the Edit Data Redaction Policy page, choose to edit the policy expression, add new columns to the redaction policy, modify the redaction details of a column that is a part of the policy, or delete a column from the redaction policy.

   You can do the following:

   - To add a new column to the redaction policy, in the Object Columns section, click **Add,** select the table or view column that you want to add, then specify the redaction details.
   - To modify the redaction details of a column that is a part of the policy, select the column, click **Modify,** then edit the redaction details.
   - To delete a column from the redaction policy, select the column, then click **Delete**.

8. On the Edit Data Redaction Policy page, after editing the required fields, click **OK** to save and enable the edited redaction policy.

Viewing Oracle Data Redaction Policy Details Using Enterprise Manager

You can find Oracle Data Redaction policy details such as whether the policy is enabled by using Enterprise Manager Cloud Control.

You can disable an enabled redaction policy, or enable a disabled redaction policy using Enterprise Manager Cloud Control.

1. Log into Oracle Enterprise Manager Cloud Control as either user `SYSTEM` or `SYSMAN`.

   The URL is as follows:

   `https://host:port/em`

2. From the **Targets** menu, select **Databases**.
3. Select **Search List**, then search for and click the name of the database target for which the Oracle Data Redaction policy that you want to view was created.
4. On the home page of the database target, from the **Security** menu, select **Data Redaction**.
5. Log in to the database, if you are prompted to do so.

   Ensure that you log in to the database as a user that has the `EXECUTE` privilege on the `DBMS_REDACT` PL/SQL package.
6. In the Policies section of the Policies tab, do one of the following:
   - Select the name of the policy in the table.
   - Select the required redaction policy, then click **View**.
7. To exit, click **OK**.

Enabling or Disabling an Oracle Data Redaction Policy in Enterprise Manager

An Oracle Data Redaction policy is executed at run time only if it is enabled. When you create an Oracle Data Redaction policy, it is enabled by default.

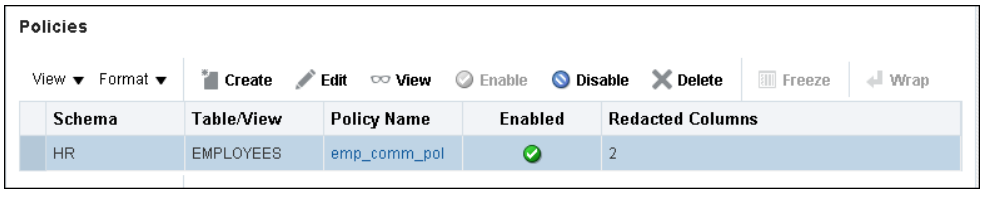
You can disable an enabled redaction policy, or enable a disabled redaction policy using Enterprise Manager Cloud Control.

1. Log into Oracle Enterprise Manager Cloud Control as either user `SYSTEM` or `SYSMAN`.

   The URL is as follows:

   `https://host:port/em`
2. From the **Targets** menu, select **Databases**.
3. Select **Search List**, then search for and click the name of the database target for which the Oracle Data Redaction policy that you want to enable or disable was created.
4. On the home page of the database target, from the **Security** menu, select **Data Redaction**.
5. Log in to the database, if you are prompted to do so.

   Ensure that you log in to the database as a user that has the `EXECUTE` privilege on the `DBMS_REDACT` PL/SQL package.
6. In the Policies section of the Policies tab, select the redaction policy that you want to enable or disable, and then click **Enable** or **Disable**.



7. In the Confirmation dialog box, click **Yes** or **No**.

Deleting an Oracle Data Redaction Policy Using Enterprise Manager

You can delete an Oracle Data Redaction policy using Enterprise Manager Cloud Control.

1. Log into Oracle Enterprise Manager Cloud Control as either user `SYSTEM` or `SYSMAN`.

   The URL is as follows:

   `https://host:port/em`

2. From the **Targets** menu, select **Databases**.
3. Select **Search List**, then search for and click the name of the database target for which the Oracle Data Redaction policy that you want to delete was created.
4. On the home page of the database target, from the **Security** menu, select **Data Redaction**.
5. Log in to the database, if you are prompted to do so.

   Ensure that you log in to the database as a user that has the `EXECUTE` privilege on the `DBMS_REDACT` PL/SQL package.

6. In the Policies section of the Policies tab, select the redaction policy that you want to delete, and then click **Delete.**
7. In the Confirmation dialog box, click **Yes** or **No**.

Managing Named Data Redaction Policy Expressions Using Enterprise Manager

You can manage Oracle Data Redaction policy expressions in Enterprise Manager Cloud Control.

- About Named Data Redaction Policy Expressions in Enterprise Manager
  You can create and apply named Oracle Data Redaction policy expression to multiple columns in tables and views in Oracle Enterprise Manager Cloud Control.
- Creating a Named Data Redaction Policy Expression in Enterprise Manager
  You can create and apply a named Oracle Data Redaction policy expression using Enterprise Manager Cloud Control.
- Editing a Named Data Redaction Policy Expression in Enterprise Manager
  You can edit a named Oracle Data Redaction policy expression using Enterprise Manager Cloud Control.
- Viewing Named Data Redaction Policy Expressions in Enterprise Manager
  You can view named Oracle Data Redaction policy expressions using Enterprise Manager Cloud Control.
- Deleting a Named Data Redaction Policy Expression in Enterprise Manager
  You can delete named Oracle Data Redaction policy expressions using Enterprise Manager Cloud Control.

About Named Data Redaction Policy Expressions in Enterprise Manager

You can create and apply named Oracle Data Redaction policy expression to multiple columns in tables and views in Oracle Enterprise Manager Cloud Control.

When you modify the policy expression, the change is reflected in all redacted columns in the database instance that use the policy expression. Cloud Control enables you to create, edit, view, apply to columns, and delete policy expressions. Before you can create and use named Data Redaction policy expressions, ensure that the `COMPATIBLE` initialization parameter is set to `12.2.0.0`.

Related Topics

- Creating and Managing Multiple Named Policy Expressions
  A named, centrally managed Oracle Data Redaction policy expression can be used in multiple redaction policies and applied to multiple tables or views.

Creating a Named Data Redaction Policy Expression in Enterprise Manager

You can create and apply a named Oracle Data Redaction policy expression using Enterprise Manager Cloud Control.

1. Log into Oracle Enterprise Manager Cloud Control as either user `SYSTEM` or `SYSMAN`.

   The URL is as follows:

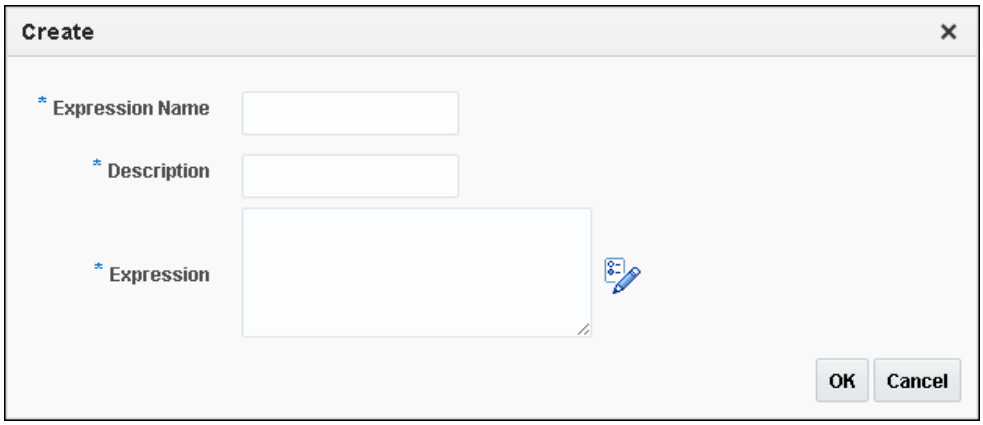
   `https://host:port/em`

2. From the **Targets** menu, select **Databases**.
3. Select **Search List**, then click the name of a database target for which you want to create an Oracle Data Redaction policy.
4. On the home page of the database target, from the **Security** menu, select **Data Redaction**.
5. Log in to the database, if you are prompted to do so.

   Ensure that you log in to the database as a user that has the `EXECUTE` privilege on the `DBMS_REDACT` PL/SQL package.

6. On the Oracle Data Redaction page, select the **Expressions** tab.
7. Click **Create**.

   The Create dialog box appears.



8. In the Create dialog box, enter the following information:
   - **Expression Name**: Enter a name for the policy expression. Existing policy expressions are listed on the Data Redaction page.
   - **Description**: Enter a brief description of the policy.
   - **Expression**: Enter the expression. For more complex expressions, such as applying or exempting the policy from specific users, click the Policy Expression Builder icon at the right of the **Expression** field. Click **OK** in the Policy Expression Builder to create the expression.
9. Click **OK** in the Create dialog box.

   After you create the policy expression, it is listed in the Data Redaction page and ready to be associated with a Data Redaction policy.
10. In the Data Redaction page, select the **Policies** tab.
11. Under Policies, select the row for the policy that redacts the column to which you want to apply the policy expression, and then click **Edit**.
12. Under Object Columns, select the column that you want and then click the **Modify** button.
13. In the Modify dialog box, select the expression from the **Expression Name** list.
14. Click **OK**, and then click **OK** again in the Edit Data Redaction Policy dialog box.

Editing a Named Data Redaction Policy Expression in Enterprise Manager

You can edit a named Oracle Data Redaction policy expression using Enterprise Manager Cloud Control.

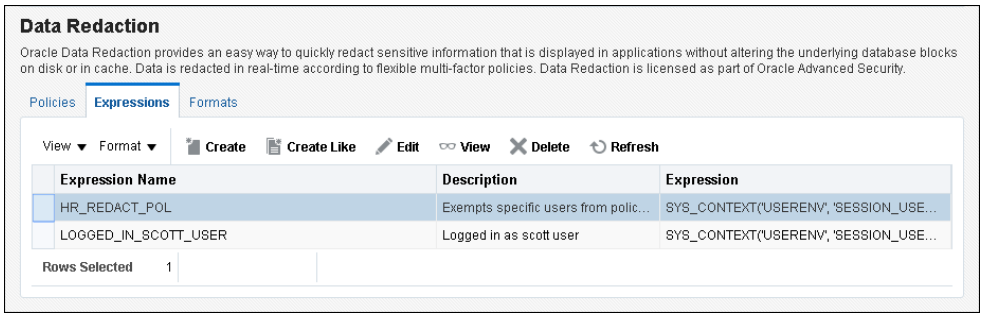
1. Log into Oracle Enterprise Manager Cloud Control as either user `SYSTEM` or `SYSMAN`.

   The URL is as follows:

   `https://host:port/em`
2. From the **Targets** menu, select **Databases**.
3. Select **Search List**, then click the name of a database target for which you want to create an Oracle Data Redaction policy.

4. On the home page of the database target, from the **Security** menu, select **Data Redaction**.
5. Log in to the database, if you are prompted to do so.

   Ensure that you log in to the database as a user that has the `EXECUTE` privilege on the `DBMS_REDACT` PL/SQL package.
6. On the Oracle Data Redaction page, select the **Expressions** tab.
7. Select the policy expression that you want to edit and then click **Edit**.



8. In the Edit dialog box, modify the **Description** and **Expression** fields as necessary. For more complex expressions, click the Policy Expression Builder icon, and then click **OK** after you have recreated the expression.
9. Click **OK** in the Edit dialog box.

Viewing Named Data Redaction Policy Expressions in Enterprise Manager

You can view named Oracle Data Redaction policy expressions using Enterprise Manager Cloud Control.

1. Log into Oracle Enterprise Manager Cloud Control as either user `SYSTEM` or `SYSMAN`.

   The URL is as follows:

   `https://host:port/em`
2. From the **Targets** menu, select **Databases**.
3. Select **Search List**, then click the name of a database target for which you want to create an Oracle Data Redaction policy.
4. On the home page of the database target, from the **Security** menu, select **Data Redaction**.
5. Log in to the database, if you are prompted to do so.

   Ensure that you log in to the database as a user that has the `EXECUTE` privilege on the `DBMS_REDACT` PL/SQL package.
6. On the Oracle Data Redaction page, select the **Expressions** tab.
7. Select the policy expression that you want to view and then click **View**.

   The View dialog box appears, showing the definition of the policy expression.
8. Click **OK** to exit the View dialog box.

Deleting a Named Data Redaction Policy Expression in Enterprise Manager

You can delete named Oracle Data Redaction policy expressions using Enterprise Manager Cloud Control.

The deletion process first dissociates the policy expression from all columns to which it is applied.

1. Log into Oracle Enterprise Manager Cloud Control as either user `SYSTEM` or `SYSMAN`.

   The URL is as follows:

   `https://host:port/em`

2. From the **Targets** menu, select **Databases**.
3. Select **Search List**, then click the name of a database target for which you want to create an Oracle Data Redaction policy.
4. On the home page of the database target, from the **Security** menu, select **Data Redaction**.
5. Log in to the database, if you are prompted to do so.

   Ensure that you log in to the database as a user that has the `EXECUTE` privilege on the `DBMS_REDACT` PL/SQL package.

6. On the Oracle Data Redaction page, select the **Expressions** tab.
7. Select the policy expression that you want to delete, and then click **Delete**.

   The Delete Expressions confirmation dialog box appears.

8. Click **OK**.

15

Using Oracle Data Redaction with Oracle Database Features

Oracle Data Redaction can be used with other Oracle features, but some Oracle features may have restrictions with regard to Oracle Data Redaction.

- Oracle Data Redaction General Usage Guidelines
  It is important to understand usage guidelines for using Oracle Data Redaction.
- Oracle Data Redaction and DML and DDL Operations
  Oracle Data Redaction affects DML and DDL operations, especially for users who issue ad-hoc SQL against tables with redacted columns.
- Oracle Data Redaction and Nested Functions, Inline Views, and the WHERE Clause
  You can use Oracle Data Redaction with nested functions, inline views, and the `WHERE` clause in `SELECT` statements.
- Oracle Data Redaction and Queries on Columns Protected by Data Redaction Policies
  Queries that include the `DISTINCT` or `ORDER BY` clause on columns that are protected by Oracle Data Redaction policies may return 0 rows.
- Oracle Data Redaction and Database Links
  Do not create Oracle Data Redaction policies on database views that reference database links.
- Oracle Data Redaction and Aggregate Functions
  Aggregate functions can affect performance overhead on Oracle Data Redaction policies.
- Oracle Data Redaction and Object Types
  You can use object types to model real-world entities such as customer accounts.
- Oracle Data Redaction and XML Generation
  You cannot use XML generation functions on columns that have Oracle Data Redaction policies defined on them.
- Oracle Data Redaction and Editions
  You cannot redact editioned views.
- Oracle Data Redaction in a Multitenant Environment
  In a multitenant environment, Oracle Data Redaction policies apply only to the objects within the current pluggable database (PDB).
- Oracle Data Redaction and Oracle Virtual Private Database
  Oracle Data Redaction does not affect Oracle Virtual Private Database policies because the VPD inline view, which contains the VPD predicate, acts on actual values.
- Oracle Data Redaction and Oracle Database Real Application Security
  Oracle Data Redaction differs from Oracle Database Real Application Security because of how security is implemented for applications.

- Oracle Data Redaction and Oracle Database Vault
  You can use Oracle Data Redaction in an Oracle Database Vault environment.
- Oracle Data Redaction and Oracle Data Pump
  Oracle Data Pump export operations can affect objects that have Oracle Data Redaction policies.
- Oracle Data Redaction and Data Masking and Subsetting Pack
  Oracle Enterprise Manager Data Masking and Subsetting Pack can be used to create a development or test copy of a production database.
- Oracle Data Redaction and JSON
  JavaScript Object Notation (JSON) can be used to create `is json` constraints on table columns.

Oracle Data Redaction General Usage Guidelines

It is important to understand usage guidelines for using Oracle Data Redaction.

- Do not include any redacted columns in a SQL expression that is used in a `GROUP BY` clause in a SQL statement. Oracle does not support this, and raises an `ORA-00979: not a GROUP BY expression` error. This happens because internally the expression in the `SELECT` list must be modified by Data Redaction, but this causes it to no longer be found when it comes time to process the `GROUP BY` clause (which is currently not updated by Data Redaction) leading to this unintended error message.
- Do not include any redacted columns in a SQL expression that is used in both the `DISTINCT` clause and `ORDER BY` clause in a SQL statement. Oracle does not support this, and raises an error: `ORA-01791: not a SELECTed expression`. This happens because internally the expression in the `SELECT` list must be modified by Data Redaction, but this causes it to no longer be found when it comes time to process the `GROUP BY` clause, leading to this unintended error message.
- An `ORA-28094: SQL construct not supported by data redaction` error is raised if a query involves a `UNION` of redacted columns and each branch of the `UNION` does not have the same redaction policy. To avoid the `ORA-28094` error, ensure that the query has the following properties:
  - When a column in the `UNION` has a redaction policy, the corresponding column in each branch of the `UNION` must use a redaction policy with the same values for all of its properties:
    * Function type
    * Function parameters or `REGEXP` parameters
    * Policy expression
    * Enable flag

    It can be a different redaction policy, but all these properties must be the same.
  - In an inline view, a `UNION` cannot have a subquery or any SQL expression that involves a redacted column.

Oracle Data Redaction and DML and DDL Operations

Oracle Data Redaction affects DML and DDL operations, especially for users who issue ad-hoc SQL against tables with redacted columns.

Note the following:

- Oracle Data Redaction treats the `RETURNING INTO` clause of a DML statement as a query, even though the result is not displayed. The result that is sent to the buffer is what would have been displayed had the `RETURNING INTO` clause been run as an ordinary SQL query, rather than as part of a DML statement. If your application performs further processing on the buffer that contains the `RETURNING INTO` value, then consider changing the application because it may not expect to find a redacted value in the buffer.
- If a redacted column appears as the source in a DML or DDL operation, then Oracle Data Redaction considers this as an attempt to circumvent the policy and prevents it with an `ORA-28081: Insufficient privileges - the command references a redacted object` error unless you have the `EXEMPT REDACTION POLICY` system privilege. Internally, Oracle Data Pump issues these kinds of operations, so you may also need to grant the `EXEMPT REDACTION POLICY` system privilege to a user if they need to perform schema-level exports of tables that have redacted columns.

Oracle Data Redaction and Nested Functions, Inline Views, and the WHERE Clause

You can use Oracle Data Redaction with nested functions, inline views, and the `WHERE` clause in `SELECT` statements.

Oracle Data Redaction policies work as follows:

- **Nested functions are redacted innermost.** For example, in `SELECT SUM(AVG(TO_NUMBER(((X))) FROM HR.EMPLOYEES WHERE ...`, the `TO_NUMBER` function is redacted first, followed by `AVG`, and then last the `SUM` function.
- **Inline views are redacted outermost.** For example, in `SELECT XYZ … AS SELECT A… AS SELECT B… AS SELECT C…`, `SELECT XYZ` is redacted first, followed by `AS SELECT A`, then `AS SELECT B`, and so on. `AS SELECT C` is redacted last.
- **The WHERE clause is never redacted.** Data Redaction redacts only data in the column `SELECT` list.

Oracle Data Redaction and Queries on Columns Protected by Data Redaction Policies

Queries that include the `DISTINCT` or `ORDER BY` clause on columns that are protected by Oracle Data Redaction policies may return 0 rows.

This happens because redaction preserves the semantics of the query. For example, a query such as `SELECT * table_name WHERE sensitive_column LIKE 'value';` would likely result in 0 rows returned, because the redacted value would not match

the '*value*' value entered in the WHERE clause. To work around this issue, rewrite the query to include an inline view so that the semantic layer can find the column. For example, instead of the following query:

```
SELECT DISTINCT sensitive_column
FROM table_name
ORDER BY sensitive_column;
```

Write the query as follows:

```
SELECT sensitive_column FROM
(SELECT DISTINCT sensitive_column
 FROM table_name
 ORDER BY sensitive_column);
```

Oracle Data Redaction and Database Links

Do not create Oracle Data Redaction policies on database views that reference database links.

You can find information about existing database links by querying the DBA_DB_LINKS data dictionary view.

> **See Also:**
>
> *Oracle Database Administrator's Guide* for detailed information about database links

Oracle Data Redaction and Aggregate Functions

Aggregate functions can affect performance overhead on Oracle Data Redaction policies.

Because Oracle Data Redaction dynamically modifies the value of each row in a column, certain SQL queries that use aggregate functions cannot take full advantage of database optimizations that presume the row values to be static.

In the case of SQL queries that call aggregate functions, it may be possible to notice performance overhead due to redaction.

Oracle Data Redaction and Object Types

You can use object types to model real-world entities such as customer accounts.

An object type is a user-defined type. You cannot redact object types. This is because Database Redaction cannot handle all of the possible ways that object types can be configured, because they are user defined. You can find the type that an object uses by querying the OBJECT_NAME and OBJECT_TYPE columns of the ALL_OBJECTS data dictionary view.

Oracle Data Redaction and XML Generation

You cannot use XML generation functions on columns that have Oracle Data Redaction policies defined on them.

Oracle XML DB Developer's Guide describes the kinds of SQL functions to which this restriction applies. This restriction applies irrespective of whether the Oracle Data Redaction policy has been enabled or disabled, or is active for the querying user.

Oracle Data Redaction and Editions

You cannot redact editioned views.

In addition to not being able to redact editioned views, you cannot use a redacted column in the definition of any editioned view. You can find information about editions by querying the `DBA_EDITIONS` data dictionary view.

Oracle Data Redaction in a Multitenant Environment

In a multitenant environment, Oracle Data Redaction policies apply only to the objects within the current pluggable database (PDB).

You cannot create a Data Redaction policy for a multitenant container database (CDB). This is because the objects for which you create Data Redaction policies typically reside in a PDB. If you have the `SYSDBA` privilege, then you can list all the PDBs in a CDB by running the `SHOW PDBS` command.

As with the CDB root, you cannot create Data Redaction policies in an application root.

Oracle Data Redaction and Oracle Virtual Private Database

Oracle Data Redaction does not affect Oracle Virtual Private Database policies because the VPD inline view, which contains the VPD predicate, acts on actual values.

Oracle Data Redaction differs from Oracle Virtual Private Database in the following ways:

- Oracle Data Redaction provides more redacting features than Oracle Virtual Private Database, which only supports `NULL` redacting. Many applications cannot use `NULL` redacting, so Data Redaction is a good solution for these applications.
- Oracle Virtual Private Database policies can be static, dynamic, and context sensitive, whereas Data Redaction policies only allow static and context-sensitive policy expressions.
- Data Redaction permits only one policy to be defined on a table or view, whereas you can define multiple Virtual Private Database policies on an object.
- Data Redaction is when application users try to access an object that is protected by a Data Redaction policy using a synonym, but (unlike Oracle Virtual Private Database) Data Redaction does not support the creation of policies directly on the synonyms themselves.

Oracle Data Redaction and Oracle Database Real Application Security

Oracle Data Redaction differs from Oracle Database Real Application Security because of how security is implemented for applications.

Oracle Data Redaction differs from Oracle Database Real Application Security in that Real Application Security provides a comprehensive authorization framework for application security.

Column security within Real Application Security is based on application privileges that are defined by applications using the Real Application Security framework.

> **See Also:**
>
> *Oracle Database Real Application Security Administrator's and Developer's Guide* for information about how you can protect table columns with custom application privileges

Oracle Data Redaction and Oracle Database Vault

You can use Oracle Data Redaction in an Oracle Database Vault environment.

For example, if there is an Oracle Database Vault realm around an object, a user who does not belong to the authorized list of realm owners or participants cannot see the object data, regardless of whether the user was granted the `EXEMPT REDACTION POLICY` privilege. If the user attempts a DML or DDL statement on the data, error messages result.

Oracle Data Redaction and Oracle Data Pump

Oracle Data Pump export operations can affect objects that have Oracle Data Redaction policies.

- Oracle Data Pump Security Model for Oracle Data Redaction
  The `DATAPUMP_EXP_FULL_DATABASE` role includes the powerful `EXEMPT REDACTION POLICY` system privilege.
- Export of Objects That Have Oracle Data Redaction Policies Defined
  You can export objects that have already had Oracle Data Redaction policies defined on them.
- Export of Data Using the EXPDP Utility access_method Parameter
  Oracle Data Pump can export data from a schema that contains an object that has a Data Redaction policy.
- Import of Data into Objects Protected by Oracle Data Redaction
  During an import operation, be careful that you do not inadvertently drop data redaction policies that protect imported data.

Oracle Data Pump Security Model for Oracle Data Redaction

The DATAPUMP_EXP_FULL_DATABASE role includes the powerful EXEMPT REDACTION POLICY system privilege.

Remember that by default the DBA role is granted the DATAPUMP_EXP_FULL_DATABASE role as well as DATAPUMP_IMP_FULL_DATABASE.

This enables users who were granted these roles to be exempt from Data Redaction policies. This means that, when you export objects with Data Redaction policies defined on them, the actual data in the protected tables is copied to the Data Pump target system without being redacted. Users with these roles, including users who were granted the DBA role, are able to see the actual data in the target system.

However, by default, all of the Data Redaction policies associated with any tables and views in the Data Pump source system are also included in the export and import operation (along with the objects themselves) and applied to the objects in the target system, so the data is still redacted when users query the objects in the target system.

Related Topics

- Exemption of Users from Oracle Data Redaction Policies
  You can exempt users from having Oracle Data Redaction policies applied to the data they access.

Export of Objects That Have Oracle Data Redaction Policies Defined

You can export objects that have already had Oracle Data Redaction policies defined on them.

- Finding Type Names Used by Oracle Data Pump
  You must find the type names Oracle Data Pump uses before exporting objects that have Oracle Data Redaction policies defined on these objects.
- Exporting Only the Data Dictionary Metadata Related to Data Redaction Policies
  You can export only the data dictionary metadata that is related to data redaction policies and full redaction settings.
- Importing Objects Using the INCLUDE Parameter in IMPDP
  You can import objects using Oracle Database Pump.

Finding Type Names Used by Oracle Data Pump

You must find the type names Oracle Data Pump uses before exporting objects that have Oracle Data Redaction policies defined on these objects.

After you find these types, you should use these types as parameters for the INCLUDE directive to the IMPDP utility, to selectively export only metadata of these specific types to the dump file.

- To find type names, query the DATABASE_EXPORT_OBJECTS view.

For example:

```
SELECT OBJECT_PATH
FROM DATABASE_EXPORT_OBJECTS
WHERE OBJECT_PATH LIKE 'RADM_%';
```

Output similar to the following appears:

```
OBJECT_PATH
------------
RADM_FPTM
RADM_POLICY
```

Exporting Only the Data Dictionary Metadata Related to Data Redaction Policies

You can export only the data dictionary metadata that is related to data redaction policies and full redaction settings.

This kind of Data Pump export could, for example, be used if you must use the same set of Data Redaction policies and settings across development, test, and production databases. Because the flag `content=metadata_only` is specified, the dump file does not contain any actual data.

- To export only the data dictionary metadata related to data redaction policies and full redaction settings, enter an `EXPDP` utility command similar to the following:

```
expdp system/password \
full=y \
COMPRESSION=NONE \
content=metadata_only \
INCLUDE=RADM_FPTM,RADM_POLICY\
directory=my_directory \
job_name=my_job_name \
dumpfile=my_data_redaction_policy_metadata.dmp
```

See Also:

- *Oracle Database Utilities* for detailed information about the `INCLUDE` parameter of the `EXPDP` utility
- *Oracle Database Utilities* for detailed information about metadata filters

Importing Objects Using the INCLUDE Parameter in IMPDP

You can import objects using Oracle Database Pump.

- To import the objects, include these names in the `INCLUDE` parameter in the `IMPDP` utility command, based on the output from querying the `OBJECT_PATH` column in the `DATABASE_EXPORT_OBJECTS` view.

Export of Data Using the EXPDP Utility access_method Parameter

Oracle Data Pump can export data from a schema that contains an object that has a Data Redaction policy.

If you are using Oracle Data Pump to perform full database export operations using the Data Pump default settings (`direct_path`), and if you receive error messages that

you do not understand, then use this section to repeat the operation in such a way as to better understand the error.

If you try to use the Oracle Data Pump Export (EXPDP) utility with the access_method parameter set to direct_path to export data from a schema that contains an object that has a Data Redaction policy defined on it, then the following error message may appear and the export operation fails:

```
ORA-31696: unable to export/import TABLE_DATA:"schema.table" using client
specified DIRECT_PATH method
```

This problem only occurs when you perform a schema-level export as a user who was not granted the EXP_FULL_DATABASE role. It does not occur during a full database export, which requires the EXP_FULL_DATABASE role. The EXP_FULL_DATABASE role includes the EXEMPT REDACTION POLICY system privilege, which bypasses Data Redaction policies.

To find the underlying problem, try the EXPDP invocation again, but do not set the access_method parameter to direct_path. Instead, use either automatic or external_table. The underlying problem could be a permissions problem, for example:

```
ORA-28081: Insufficient privileges - the command references a redacted object.
```

> **See Also:**
>
> *Oracle Database Utilities* for more information about using Data Pump Export.

Import of Data into Objects Protected by Oracle Data Redaction

During an import operation, be careful that you do not inadvertently drop data redaction policies that protect imported data.

Consider a scenario in which the source tables that were exported using the Oracle Data Pump Export (EXPDP) utility do not have Oracle Data Redaction polices. However, the destination tables to which the data is to be imported by using Oracle Data Pump Import (IMPDP) have Oracle Data Redaction policies.

During the Data Pump import operation, the status of the Data Redaction policies on the objects being imported depends on the CONTENT option of IMPDP command.

- If you use the CONTENT=ALL or CONTENT=METADATA_ONLY option in the IMPDP command, then the Data Redaction policies on the destination tables are dropped. You must recreate the Data Redaction policies.
- If you use CONTENT=DATA_ONLY in the IMPDP command, then the Data Redaction polices on the destination tables are not dropped.

> **See Also:**
>
> *Oracle Database Utilities* for more information about using Data Pump Export

Oracle Data Redaction and Data Masking and Subsetting Pack

Oracle Enterprise Manager Data Masking and Subsetting Pack can be used to create a development or test copy of a production database.

To accomplish this, you can mask this data in bulk, and then put the resulting masked data in the development or test copy.

You can still apply Data Redaction policies to the non-production database, in order to redact columns that contain data that was already masked by Oracle Enterprise Manager Data Masking and Subsetting Pack.

Remember that Oracle Enterprise Manager Data Masking and Subsetting Pack is used to mask data sets when you want to move the data to development and test environments. Data Redaction is mainly designed for redacting at runtime for production applications in a consistent fashion across multiple applications, without having to make application code changes.

> **See Also:**
>
> *Oracle Data Masking and Subsetting Guide* for more information about data masking and subsetting

Oracle Data Redaction and JSON

JavaScript Object Notation (JSON) can be used to create `is json` constraints on table columns.

However, you cannot create an Oracle Data Redaction policy on a table column that has the `is json` constraint. If you attempt to do so, an `ORA-28073 - The column column_name has an unsupported datatype` error is raised. As a workaround solution, Oracle recommends that you create a relational view that uses the `JSON_TABLE` row source operator on top of the JSON object, and then apply the Data Redaction policy to this view.

See *Oracle Database SQL Language Reference* for more information about `JASON_TABLE`.

16

Security Considerations for Oracle Data Redaction

Oracle provides guidelines for using Oracle Data Redaction.

- Oracle Data Redaction General Security Guidelines
  It is important to understand general security guidelines for using Oracle Data Redaction.
- Restriction of Administrative Access to Oracle Data Redaction Policies
  You can restrict the list of users who can create, view and edit Data Redaction policies.
- How Oracle Data Redaction Affects the SYS, SYSTEM, and Default Schemas
  Both users `SYS` and `SYSTEM` automatically have the `EXEMPT REDACTION POLICY` system privilege.
- Policy Expressions That Use SYS_CONTEXT Attributes
  Be careful when writing a policy expression that depends on a `SYS_CONTEXT` attribute that is populated by an application.
- Oracle Data Redaction Policies on Materialized Views
  You can create Oracle Data Redaction policies on materialized views and on their base tables.
- REDACTION_COLUMNS Data Dictionary View Behavior When a View Is Invalid
  When an Oracle Data Redaction policy exists on a column of a view, and the view becomes invalid, the Data Redaction policy remains visible in the `REDACTION_COLUMNS` data dictionary view.
- Dropped Oracle Data Redaction Policies When the Recycle Bin Is Enabled
  You should check if the recycle bin is enabled before you drop Oracle Data Redaction policies.

Oracle Data Redaction General Security Guidelines

It is important to understand general security guidelines for using Oracle Data Redaction.

- Oracle Data Redaction is not intended to protect against attacks by regular and privileged database users who run ad hoc queries directly against the database. If the user can issue arbitrary SQL or PL/SQL statements, then he or she will be able to access the actual value.
- Oracle Data Redaction is not intended to protect against users who run ad hoc SQL queries that attempt to determine the actual values by inference.
- Oracle Data Redaction relies on the database and application context values. For applications, it is the responsibility of the application to properly initialize the context value.
- Oracle Data Redaction is not enforced for users who are logged in using the `SYSDBA` administrative privilege.

- Certain DDL statements that attempt to copy the actual data out from under the control of a data redaction policy (that is, `CREATE TABLE AS SELECT`, `INSERT AS SELECT`) are blocked by default, but you can disable this behavior by granting the user the `EXEMPT REDACTION POLICY` system privilege.
- Oracle Data Redaction does not affect day-to-day database operations, such as backup and recovery, Oracle Data Pump exports and imports, Oracle Data Guard operations, and replication.

Restriction of Administrative Access to Oracle Data Redaction Policies

You can restrict the list of users who can create, view and edit Data Redaction policies.

To accomplish this, you can limit who has the `EXECUTE` privilege on the `DBMS_REDACT` package and by limiting who has the `SELECT` privilege on the `REDACTION_POLICIES` and `REDACTION_COLUMNS` views.

You also can restrict who is exempted from redaction by limiting the `EXEMPT REDACTION POLICY` privilege. If you use Oracle Database Vault to restrict privileged user access, then you can use realms to restrict granting of `EXEMPT REDACTION POLICY`.

Related Topics

- Oracle Data Redaction and Oracle Database Vault
  You can use Oracle Data Redaction in an Oracle Database Vault environment.
- Exemption of Users from Oracle Data Redaction Policies
  You can exempt users from having Oracle Data Redaction policies applied to the data they access.
- Introduction to Oracle Database Vault

How Oracle Data Redaction Affects the SYS, SYSTEM, and Default Schemas

Both users `SYS` and `SYSTEM` automatically have the `EXEMPT REDACTION POLICY` system privilege.

`SYSTEM` has the `EXP_FULL_DATABASE` role, which includes the `EXEMPT REDACTION POLICY` system privilege.

This means that the `SYS` and `SYSTEM` users can always bypass any existing Oracle Data Redaction policies, and will always be able to view data from tables (or views) that have Data Redaction policies defined on them.

Follow these guidelines:

- Do not create Data Redaction policies on the default Oracle Database schemas, including the `SYS` and `SYSTEM` schemas.
- Be aware that granting the `EXEMPT DATA REDACTION` system privilege to additional roles may enable users to bypass Oracle Data Redaction, because the grantee role may have been granted to additional roles.

- Do not revoke the `EXEMPT DATA REDACTION` system privilege from the roles that it was granted to by default.

Policy Expressions That Use SYS_CONTEXT Attributes

Be careful when writing a policy expression that depends on a `SYS_CONTEXT` attribute that is populated by an application.

The application might not always populate that attribute.

If the user somehow connects directly (rather than through the application), then the `SYS_CONTEXT` attribute would not have been populated. If you do not handle this `NULL` scenario in your policy expression, you could unintentionally reveal actual data to the querying user.

For example, suppose you wanted to create a policy expression that intends to redact the query results for everyone except users who have the client identifier value of `SUPERVISOR`. The following expression unintentionally enables querying users who have `NULL` as the value for their `CLIENT_IDENTIFIER` to see the real data:

```
SYS_CONTEXT('USERENV', 'CLIENT_IDENTIFIER') IS NOT 'SUPERVISOR'
```

A more rigorous policy expression redacts the result of the query if the client identifier is not set, that is, it has a `NULL` value.

```
SYS_CONTEXT('USERENV', 'CLIENT_IDENTIFIER') IS NOT 'SUPERVISOR' OR IS NULL
```

Remember that in SQL, comparisons with `NULL` are undefined, and are thus `FALSE`, but redaction only takes place when the policy expression evaluates to `TRUE`.

Oracle Data Redaction Policies on Materialized Views

You can create Oracle Data Redaction policies on materialized views and on their base tables.

However, ensure that the creator of the materialized view, or the user who performs the refresh of the materialized view, is not blocked by any Data Redaction policies. In other words, the user performing the materialized view creation or refresh operations should be exempt from the Data Redaction policy. As a best practice, when you create a new materalized view, treat it as a copy of the actual table, and then create a separate Data Redaction policy to protect it.

REDACTION_COLUMNS Data Dictionary View Behavior When a View Is Invalid

When an Oracle Data Redaction policy exists on a column of a view, and the view becomes invalid, the Data Redaction policy remains visible in the `REDACTION_COLUMNS` data dictionary view.

For example, a view can become invalid if one of its columns refers to a column that was dropped from a table upon which the view depends.

The column continues to be visible in the `REDACTION_COLUMNS` data dictionary view because the Data Redaction policy is not automatically dropped when the view becomes invalid.

Instead, the decision on whether to drop the Data Redaction policy is taken when the view is subsequently altered.

This approach was chosen in preference to automatically dropping the Data Redaction policy when the view becomes invalid because it is less error-prone and presents less risk of accidentally displaying actual data from the underlying table.

By deferring the decision to when the view is being altered, it allows the view to be recompiled after the column is restored to the table. After the column is restored to the table and the view is recompiled, then the view becomes valid and still has its Data Redaction policy in place.

On the other hand, if the invalid view definition was subsequently replaced with a valid view definition which no longer contains the column that the Data Redaction policy was previously defined on, it is at that point that the Data Redaction policy is automatically dropped. The `REDACTION_COLUMNS` data dictionary view is then updated to no longer show the column (since it is no longer part of the new view's definition).

Dropped Oracle Data Redaction Policies When the Recycle Bin Is Enabled

You should check if the recycle bin is enabled before you drop Oracle Data Redaction policies.

If you drop a table or view that has an Oracle Data Redaction policy defined on it when the recycle bin feature is enabled, and if you query the `REDACTION_COLUMNS` or `REDACTION_POLICIES` data dictionary views before you purge the recycle bin, then you will see object names such as `BIN$...` (for example, `BIN$1Xu5PSW5VaPgQxGS5AoAEA==$0`).

This is normal behavior. These policies are removed when you purge the recycle bin.

To find if the recycle bin is enabled, you can run the `SHOW PARAMETER RECYCLEBIN` command in SQL*Plus.

> **See Also:**
>
> *Oracle Database Administrator's Guide* for information about purging objects from the recycle bin

Glossary

actual data

In Oracle Data Redaction, the data in a protected table or view. An example of actual data could be the number `123456789`, and the redacted data version of this number could be `999996789`.

auto-login software keystore

A software keystore that is protected by a system-generated password and does not need to be explicitly opened by a security administrator. Auto-login software keystores are automatically opened when accessed and can be used on any computer that runs an Oracle database. For example, consider an Oracle RAC environment that has four nodes, and each node is on a different computer. This environment uses an auto-login keystore. When a REKEY operation is performed on node 1, the auto-login and password-based keystores must be copied to the computers that host nodes 2, 3, and 4. In this configuration, the auto-login keystores will be opened on all four nodes when required.

See also local auto-login software keystore.

cipher suite

A set of authentication, encryption, and data integrity algorithms used to exchange messages between network nodes using Secure Sockets Layer (SSL). During an SSL handshake, for example, the two nodes negotiate to see which cipher suite they will use when transmitting messages back and forth.

ciphertext

Message text that has been encrypted.

See also encrypted text.

data redaction

The ability to mask data with different values in real time, that is, at the moment a user tries to access the data. You can mask all of the data, a partial subset of the data, or

display random values in place of the data. It does not change the actual data in the database.

decryption

The process of converting an encrypted message (the ciphertext), back to its original message (plaintext).

encrypted text

Text that has been encrypted, using an encryption algorithm and an encryption key; the output stream of an encryption process. The text is not readable or decipherable, without decrypting it first. Also called ciphertext.

encryption

The process of converting an original message (plaintext) to an encrypted message (ciphertext).

hardware keystore

A container that stores a Transparent Data Encryption key for a hardware security module.

hardware security module

A physical device that provides secure storage for encryption keys.

inference

A query that is designed to find data by repeatedly trying queries. For example, to find the users who earn the highest salaries, an intruder could use the following query:

```
SELECT FIRST_NAME, LAST_NAME, SALARY FROM HR.EMPLOYEES WHERE SALARY > 16000
ORDER BY SALARY DESC;

FIRST_NAME           LAST_NAME                 SALARY
-------------------- ------------------------- ----------
Steven               King                       24000
Neena                Kochhar                    17000
Lex                  De Haan                    17000
```

key pair

A public key and its associated private key. See public and private key pair.

keystore

A general term for any container that stores encryption keys, such as Transparent Data Encryption keys and other encrypted data. In previous releases, this container

was referred to as a wallet, which is specific to Oracle. Starting with Oracle Database 12c release 12.1, the term changed to keystore to encompass non-Oracle Database encryption key containers, such as hardware security modules.

See also auto-login software keystore, hardware keystore, and local auto-login software keystore.

local auto-login software keystore

A software keystore that is local and restricted to the computer on which it was created.

See also auto-login software keystore.

mask

The ability to redact data from a user or an application.

Oracle-managed tablespace

An Oracle-supplied tablespace that contains information necessary for the correct functioning (confidentiality, integrity, and availability) of the database system. This information includes the system's data dictionary, the system's temporary sort area, the system's undo segment, and the system's auxiliary data. This information is only expected to be updated internally by the Oracle database server itself, and does not normally be updated directly by users.

password-based software keystore

A software keystore that must be opened with a password before it can be accessed.

See also keystore.

plaintext

Message text that has not been encrypted.

private key

In public-key cryptography, this key is the private key that is known only to its owner. It is primarily used for encrypting message digests used with digital signatures.

See public and private key pair.

public key

One of two keys that are used in public key cryptography, the other key being the private key. In typical public key cryptography usage, the public key is used to encrypt data or verify digital signatures. The the private key is used to decrypt data or generate

digital signatures. The public key, unlike the private key, can be made available to anyone whereas the private key must remain secret.

See public and private key pair.

public key encryption

The process where the sender of a message encrypts the encryption key of the recipient. Upon delivery, the message is decrypted by the recipient using its private key.

public and private key pair

A set of two related numbers used for encryption and decryption, where one is called the private key and the other is called the public key. Public keys are typically made widely available, while private keys are held by their respective owners. Data encrypted with either a public key or a private key from a key pair can be decrypted with its associated key from the key pair.

public key infrastructure (PKI)

Information security technology utilizing the principles of public key cryptography. Public key cryptography involves encrypting and decrypting information using a shared public and private key pair. Provides for secure, private communications within a public network.

redacted data

Masked data that is displayed to the querying user. For example, if the actual data is `3714-4963-5398-4321`, then the redacted data could appear, depending on the Data Redaction policy, as `XXXX-XXXX-XXXX-4321`.

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 hashed, to avoid dictionary attacks, a method that attackers use to determine sensitive passwords. The addition of salt to a password before hashing makes it more difficult for intruders to match the hash values (sometimes called verifiers) with their dictionary list of common password hash values, because they do not know the salt beforehand.

software keystore

A container that stores a Transparent Data Encryption a TDE master encryption key for use as an auto-login software keystore, a local auto-login software keystore, or a password-based software keystore.

tablespace encryption key

An encryption key for the encryption of a tablespace. The TDE tablespace encryption key encrypts the tablespace encryption key, which in turn encrypts and decrypts data in the tablespace.

TDE master encryption key

A key that is stored within a software keystore or a hardware keystore. For table encryption, this key encrypts the TDE table key, and for tablespace encryption, it encrypts the tablespace encryption key.

See also keystore.

TDE table key

An encryption key that is associated with a table whose columns are marked for encryption. The TDE master encryption key encrypts this table encryption key.

wallet

A data structure used to store and manage security credentials for an individual entity. Wallets are specific to Oracle Database only. A Wallet Resource Locator (WRL) provides all of the necessary information to locate the wallet. For Transparent Data Encryption in Oracle Database Release 12*c* and later, the term for wallet is keystore.

wallet obfuscation

The ability to store and access an Oracle wallet without querying the user for a password before access (supports single sign-on (SSO)).

Wallet Resource Locator (WRL)

A tool that provides all of the necessary information to locate a wallet. It is a path to an operating system directory that contains a wallet.

Index

Symbols

A

C

D

E

G

H

I

L

M

N

O

P

R

S

T

U

V

X