

Oracle Database 12c: Security

Student Guide - Volume II

D81599GC10

Edition 1.0

June 2014

D84090

ORACLE®

Author

Dominique Jeunot

Technical Contributors and Reviewers

Jean-François Verrier

Donna Keesling

James Spiller

Gerlinde Frenzen

Pat Huey

veerabhadra Rao Putrevu

Joel Goodman

Editors

Malavika Jinka

Anwesha Ray

Daniel Milne

Graphic Designer

Rajiv Chandrabhanu

Publishers

Jobi Varghese

Pavithran Adka

Sumesh Koshy

Srividya Rameshkumar

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Disclaimer

This document contains proprietary information and is protected by copyright and other intellectual property laws. You may copy and print this document solely for your own use in an Oracle training course. The document may not be modified or altered in any way. Except where your use constitutes "fair use" under copyright law, you may not use, share, download, upload, copy, print, display, perform, reproduce, publish, license, post, transmit, or distribute this document in whole or in part without the express authorization of Oracle.

The information contained in this document is subject to change without notice. If you find any problems in the document, please report them in writing to: Oracle University, 500 Oracle Parkway, Redwood Shores, California 94065 USA. This document is not warranted to be error-free.

Restricted Rights Notice

If this documentation is delivered to the United States Government or anyone using the documentation on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS

The U.S. Government's rights to use, modify, reproduce, release, perform, display, or disclose these training materials are restricted by the terms of the applicable Oracle license agreement and/or the applicable U.S. Government contract.

Trademark Notice

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Contents

1 Database Security: Introduction

Course Objectives 1-2

Appendices 1-5

Prerequisites 1-6

Practice: Overview 1-7

2 Understanding Security Requirements

Objectives 2-3

Fundamental Data Security Requirements 2-4

Data Security Concerns 2-6

Compliance Mandates 2-7

Security Risks 2-9

Security Standards 2-11

Developing Your Security Policy 2-12

Defining a Security Policy 2-13

Implementing a Security Policy 2-15

Quiz 2-16

Enforcing Security at Different Levels 2-17

Principle of Least Privilege 2-18

Defense in Depth 2-19

Common Exploits 2-20

Preventing Exploits 2-22

Summary 2-24

Practice: Overview 2-25

3 Choosing Security Solutions

Objectives 3-3

Database Security Solutions 3-4

Encrypting Network Traffic 3-5

Authenticating Users to Database 3-7

Encrypting Application Data with DBMS_CRYPTO 3-9

Encrypting Application Data with TDE 3-10

Redacting Sensitive Data Displayed 3-12

Masking Data for Non-Production Use 3-13

Protecting Multiple Sensitive Columns 3-14

Controlling Data Access Using Authorization	3-15
Controlling Data Access Based on Views	3-16
Controlling Data Access by Privileged Users	3-17
Controlling Data Access Based on Function	3-19
Controlling Backup Access Based on Privilege	3-20
Controlling Data Access Based on Label	3-21
Monitoring Database Activity	3-23
Oracle Audit Vault and Database Firewall	3-24
Auditing and Alerting in Real-Time	3-26
Unified Auditing	3-27
Fine-Grained Auditing	3-28
Discovering Use of Privileges and Roles	3-29
Discovering Sensitive Data	3-30
Managing Configuration, Compliance, and Change Management	3-31
Enforcing Security at Different Levels	3-33
Quiz	3-34
Summary	3-35
Practice: Overview	3-36

4 Implementing Basic Database Security

Objectives	4-3
Database Security: Checklist	4-4
Reducing Administration Effort	4-5
Installing Only What Is Required	4-6
Applying Security Patches	4-7
Secure Password Support	4-8
Automatic Secure Configuration	4-9
Locking, Expiring, or Changing Passwords of Default Accounts	4-10
Privileged Accounts	4-12
Administrative Privileges	4-13
Limiting Users with Administrative Privileges	4-14
Separation of Responsibilities	4-16
Protecting the Data Dictionary	4-18
Limiting Privileges	4-19
Limiting External Procedures Privileges	4-21
Managing Scheduler Security	4-22
External Jobs	4-23
Summary	4-24
Practice: Overview	4-25

5 Securing Network Services

- Objectives 5-3
- Network Security: Checklist 5-4
- Restricting Network IP Addresses: Valid Node Checking 5-5
- Restricting Network IP Addresses: Guidelines 5-6
- Configuring IP Restrictions with Net Manager 5-7
- Quiz 5-8
- Listener Security: Checklist 5-9
- Restricting Nodes Registration: Valid Node Checking Registration (VNCR) 5-10
- Moving the Listener to a Nondefault Port 5-11
- Administering the Listener 5-12
- Administering the Listener Securely 5-13
- Preventing Online Administration of the Listener 5-14
- INBOUND_CONNECT_TIMEOUT 5-15
- Setting Listener-Logging Parameters 5-17
- Analyzing Listener Log Files 5-19
- Listener Log Connect: Examples 5-21
- Listener Log Command: Examples 5-23
- Managing Fine-Grained Access to External Network Services 5-25
- Summary 5-26
- Practice: Overview 5-27

6 Using Basic and Strong User Authentication

- Objectives 6-3
- User Authentication 6-4
- Basic User Authentication by Password 6-6
- Protecting Passwords 6-8
- Secure External Password Store 6-9
- Basic User Authentication by Operating System 6-10
- Quiz 6-11
- Strong User Authentication 6-12
- Public Key Infrastructure (PKI) Tools 6-15
- Certificates 6-16
- How to Use Certificates for Authentication 6-17
- Quiz 6-19
- How to Use Kerberos for Authentication 6-20
- How to Use KDC with Windows 2000 for Authentication 6-22
- RADIUS Authentication: Overview 6-24
- Fixed User Database Links 6-25
- Database Links Without Credentials 6-26
- Database Links and Changing Passwords 6-28

Auditing with Database Links 6-29
Restricting a Database Link with Views 6-30
Summary 6-32
Practice: Overview 6-33

7 Using Global User Authentication

Objectives 7-3
Enterprise User Security 7-4
Oracle Identity Management Software 7-5
Directory Structure: Overview 7-7
Oracle Database: Enterprise User Security Architecture 7-8
Authenticating Enterprise Users 7-9
Enterprise Users 7-11
Setting Up Enterprise User Security 7-12
Identifying the Enterprise User 7-13
Enabling Current User Database Links 7-14
Using Enterprise Roles 7-15
User Migration Utility 7-16
Enterprise-User Auditing 7-18
Quiz 7-19
Summary 7-20
Practice or Demo: Overview 7-21

8 Using Proxy Authentication

Objectives 8-3
User Authentication 8-4
Security Challenges of Three-Tier Computing 8-5
Common Implementations of Authentication 8-6
Quiz 8-8
Using Proxy Authentication for Database Users 8-9
Using Proxy Authentication for Enterprise Users 8-11
Revoking Proxy Authentication 8-12
Application-User Model 8-13
Using Proxy Authentication with Roles 8-15
Data Dictionary Views for Proxy Authentication 8-16
Data Dictionary Views: DBA_PROXIES and USER_PROXIES 8-17
Data Dictionary Views: V\$SESSION_CONNECT_INFO 8-18
Auditing Actions Taken on Behalf of the Real User 8-19
Summary 8-20
Practice: Overview 8-21

9 Using Privileges and Roles

- Objectives 9-3
- Authorization 9-4
- Administrative Privileges 9-5
- New Administrative Privileges 9-6
 - New Administrative Privilege: SYSBACKUP 9-7
 - New Administrative Privilege: SYSDG 9-8
 - New Administrative Privilege: SYSKM 9-9
- OS Authentication and OS Groups 9-10
- Password Authentication for SYSBACKUP 9-12
- Password Authentication for SYSDG 9-14
- Roles 9-15
 - Benefits of Roles 9-16
 - Creating Common and Local Roles 9-17
 - Granting Common and Local Privileges 9-18
 - Granting Common or Local Privileges to Roles, Common or Local Roles to Roles 9-19
- Granting Common and Local Roles to Users 9-20
- Predefined Roles 9-21
- Quiz 9-22
 - Secure Application Role 9-23
 - Implementing a Secure Application Role 9-24
 - Securing Objects with Procedures 9-25
 - Using Code-Based Access Control 9-26
 - Privilege Checking During PL/SQL Calls 9-27
 - INHERIT (ANY) PRIVILEGES Privileges 9-28
 - Privilege Checking with New BEQUEATH Views 9-29
 - Quiz 9-30
 - Creating and Using Virtual Private Catalogs 9-31
 - Summary 9-33
 - Practice: Overview 9-34

10 Using Privilege Analysis

- Objectives 10-3
- Privilege Analysis 10-4
- Privilege Analysis Flow 10-5
- Creating Policies: Database and Role Analysis 10-6
- Creating Policies: Context Analysis 10-7
- Creating Policies: Combined Analysis Types 10-8
- Analyzing and Reporting 10-9

SYSTEM and OBJECT Used Privileges 10-10
Compare Used and Unused Privileges 10-12
Views 10-13
Dropping an Analysis Policy 10-14
Quiz 10-15
Summary 10-16
Practice: Overview 10-17

11 Using Application Contexts

Objectives 11-3
Application Context: Description 11-4
Using the Application Context 11-5
Setting and Retrieving Application Context Values 11-6
Application Context Data Sources 11-7
Quiz 11-9
Application Context Accessed Globally 11-10
Application Context Accessed Globally in Action 11-12
Implementing the Application Context Accessed Globally 11-14
Viewing Application Context Information 11-15
Application Context Usage Guidelines 11-16
Summary 11-18
Practice: Overview 11-19

12 Implementing Virtual Private Database

Objectives 12-3
Fine-Grained Access Control: Overview 12-4
Understanding FGAC Policy Execution 12-6
Benefits of Using Fine-Grained Access Control 12-8
Virtual Private Database 12-9
Examples of VPD 12-10
Using DBMS_RLS to Manage Policies 12-11
Column-Level VPD 12-12
Policy Types: Overview 12-13
Designing and Implementing a VPD Solution 12-14
Implementing a VPD Policy 12-15
Writing a Function That Returns Different Predicates 12-16
Exceptions to VPD Policies 12-17
Quiz 12-18
Guidelines for Policies and Context 12-19
Policy Performance 12-21
Export and Import 12-23

Policy Views 12-24
Summary 12-25
Practice: Overview 12-26

13 Implementing Oracle Label Security

Objectives 13-3
Access Control: Overview 13-4
OLS 13-5
Enabling and Managing OLS 13-6
Quiz 13-7
OLS: Features 13-8
OLS and VPD Comparison 13-10
Analyzing Application Requirements 13-11
Implementing an OLS Solution 13-12
Create Policies With Enforcement Options 13-14
Define Labels 13-16
Assign User Authorization Labels 13-18
Apply the Policy to a Table 13-20
Adding Labels to Data 13-21
Access Mediation 13-22
Quiz 13-23
OLS Special User Privileges 13-24
Example: READ Privilege 13-25
Example: FULL Privilege 13-26
Example: COMPACCESS Privilege 13-27
Using the PROFILE_ACCESS Privilege 13-28
Trusted Stored Package Units 13-29
Exporting and Importing with OLS 13-30
Performance Tips 13-31
Summary 13-32
Practice: Overview 13-33

14 Oracle Data Redaction

Objectives 14-3
Oracle Data Redaction: Overview 14-4
Oracle Data Redaction and Operational Activities 14-6
Available Redaction Methods 14-7
Oracle Data Redaction: Examples 14-8
What Is a Redaction Policy? 14-9
Managing Redaction Policies 14-10
Defining a Redaction Policy 14-11

Adding a Redaction Policy to a Table or View	14-12
Full Redaction: Examples	14-13
Partial Redaction: Examples	14-14
Regular Expression	14-16
Modifying the Redaction Policy	14-17
Exempting Users from Redaction Policies	14-18
Defining Data Redaction Policies by Using Cloud Control 12c	14-19
Creating a Data Redaction Policy	14-20
Using Oracle Data Redaction with Other Oracle Database Security Solutions	14-22
Oracle Database Security Features	14-23
Best Practices: Preventing Unauthorized Policy Modifications and Exemptions	14-25
Best Practices: Considerations	14-26
Summary	14-27
Practice: Overview	14-28

15 Application Data Model and Oracle Data Masking

Objectives	15-3
Application Data Model: Overview	15-4
ADM and Data Masking Process	15-5
Creating an ADM	15-6
Viewing ADM Content	15-7
Discovering Sensitive Columns	15-8
Data Masking: Overview	15-9
Using the Data Masking Pack	15-10
Data Masking Pack: Features	15-11
Implementing Data Masking	15-12
Identifying Sensitive Data for Masking	15-14
Creating or Using Masking Formats	15-15
Using Oracle-Supplied Mask Formats and Built-in Masking Routines	15-16
Example: Data Masking of the EMPLOYEES Table	15-18
Creating a Masking Format Using a User-Defined Function	15-19
Creating Data Masking Definitions	15-20
Importing Formats and Modifying Properties	15-21
Using Condition-Based Masking	15-22
Using Compound Masking	15-23
Using a User-Defined Masking Function and Post-Processing Masking Function	15-24
Generating the Data Masking Script	15-25
Understanding the Data Masking Process	15-26

Creating an Application Masking Template 15-27
Controlling Data Masking Operations 15-28
Summary 15-29
Practice: Overview 15-30

16 Implementing Transparent Sensitive Data Protection

Objectives 16-3
Benefits of TSDP 16-4
TSDP: Overview 16-5
Using a TSDP Policy with VPD 16-7
Using a TSDP Policy with Data Redaction 16-9
Using the Predefined REDACT_AUDIT TSDP Policy 16-11
Disabling the REDACT_AUDIT TSDP Policy 16-13
TSDP Policies in a CDB and PDBs 16-14
Exempting Users from TSDP Policies 16-15
Maintaining TSDP Types and Sensitive Columns 16-16
Maintaining TSDP Policies 16-17
Data Dictionary Views for TDSP 16-18
Summary 16-19
Practice: Overview 16-20

17 Encryption Concepts

Objectives 17-3
Understanding Encryption 17-4
Encryption Is Not Access Control 17-5
Access by Privileged Users 17-6
What Problems Does Encryption Solve? 17-8
Cost of Encryption 17-9
What to Encrypt 17-10
Quiz 17-11
Data Encryption: Challenges 17-12
Encryption Key Management: Key Generation 17-13
Encryption Key Management: Key Modification and Transmission 17-14
Encryption Key Management: Storage 17-15
Storing the Key in the Database 17-16
Storing the Key in the File System 17-18
Letting the User Manage the Key 17-19
Solutions 17-20
Summary 17-21

18 Using Application-Based Encryption

- Objectives 18-3
- Overview 18-4
- DBMS_CRYPTO Package 18-5
- Generating Keys Using RANDOMBYTES 18-7
- Quiz 18-10
- Using ENCRYPT and DECRYPT 18-11
- Enhanced Security Using Cipher Block Modes 18-14
- Hash and Message Authentication Code 18-15
- Summary 18-18
- Practice: Overview 18-19

19 Applying Transparent Data Encryption

- Objectives 19-3
- Transparent Data Encryption 19-4
- Components of TDE 19-5
- Using TDE 19-6
- Using Hardware Security Modules 19-7
- Defining the Keystore Location 19-8
- Creating and Opening the Keystore 19-10
- Generating the Master Key 19-11
- Backing Up the Keystore 19-12
- Managing the Keystore in CDB and PDBs 19-13
- Creating and Opening the Keystore 19-14
- Setting Master Encryption Keys 19-15
- Quiz 19-16
- Re-Keying Table Keys 19-17
- Creating an Encrypted Column 19-18
- Creating an Index on an Encrypted Column 19-19
- TDE Column Encryption Support 19-20
- TDE Column-Level Storage Requirements 19-22
- TDE Column Encryption: Restrictions 19-23
- Tablespace Encryption: Advantages 19-24
- Creating an Encrypted Tablespace 19-25
- SECUREFILE LOB Encryption 19-26
- Summary 19-27
- Practice: Overview 19-28

20 Database Storage Encryption

- Objectives 20-3
- RMAN-Encrypted Backups 20-4

Oracle Secure Backup Encryption	20-5
Encrypted Backups to Tape	20-7
Creating RMAN-Encrypted Backups	20-8
Using Transparent-Mode Encryption	20-9
Using Password-Mode Encryption	20-11
Using Dual-Mode Encryption	20-12
Quiz	20-13
Restoring Encrypted Backups	20-14
RMAN-Encrypted Backups: Considerations	20-15
Data Pump Encryption	20-16
ENCRYPTION Parameter	20-17
ENCRYPTION_PASSWORD Parameter	20-18
ENCRYPTION_MODE Parameter	20-19
Encrypting Dump Files	20-20
Summary	20-21
Practice: Overview	20-22

21 Using Unified Auditing

Objectives	21-3
Types of Auditing	21-4
Oracle Database 12c Auditing	21-5
Security and Performance: Audit Architecture	21-6
Tolerance Level for Loss of Audit Records	21-7
Consolidation: Unique Audit Trail	21-8
Basic Audit Versus Extended Audit Information	21-9
Extended Audit Information	21-10
Data Pump Audit Policy	21-11
Oracle RMAN Audit Information	21-12
Unified Audit Implementation	21-13
Quiz	21-15
Security: Roles	21-17
Security: SYS Auditing	21-18
Simplicity: Audit Policy	21-19
Step 1: Creating the Audit Policy	21-20
Creating the Audit Policy: Object-Specific Actions	21-21
Creating the Audit Policy: Condition	21-22
Step 2: Enabling / Disabling the Audit Policy	21-23
Auditing Actions in a CDB and PDBs	21-24
Viewing the Audit Policy	21-25
Viewing the Audit Records CDB_UNIFIED_AUDIT_TRAIL	21-26
Using Predefined Audit Policies	21-27

Including Application Context Data 21-28
Dropping the Audit Policy 21-29
Audit Cleanup 21-30
Quiz 21-31
Summary 21-32
Practice: Overview 21-33

22 Using Fine-Grained Audit

Objectives 22-3
Fine-Grained Auditing (FGA) 22-4
FGA Policy 22-5
Triggering Audit Events 22-7
Data Dictionary Views 22-8
DBA_FGA_AUDIT_TRAIL / UNIFIED_AUDIT_TRAIL 22-9
Quiz 22-10
DBMS_FGA Package 22-11
Enabling / Disabling Dropping an FGA Policy 22-12
FGA Policy Guidelines 22-13
FGA Policy Errors 22-14
Maintaining the Audit Trail 22-15
Summary 22-16
Practice 22 Overview: Using Fine-grained Audit 22-17

A Implementing Virtual Private Database Policy Groups

Objectives A-2
Implementing Policy Groups A-3
Grouping Policies A-5
Default Policy Group A-6
Creating a Driving Context A-8
Making the Context a Driving Context A-10
Creating a Policy Group A-12
Adding a Policy to a Group A-13
Summary A-15

B Encrypting Network Traffic

Encrypting Network Traffic B-2
End-to-End Encryption B-4
Configuring Network Encryption B-5
Checksumming B-6
Configuring Checksumming B-7

C General Security Reports

- Using Database Vault Reports C-2
- General Security Reports C-3
- Database Account Password Reports C-4
- Using Powerful Database Accounts and Roles Reports C-5
- Privileged Database Accounts and Roles Reports C-6
- Initialization Parameter and Operating System Directory Permission Reports C-7
- Review Privilege Reports C-8
- General Database Privilege and Resource Profile Reports C-9
- Database Audit and Privilege Reports C-10
- Object Privilege Reports C-11
- Sensitive Objects Reports C-12
- Unified Audit Trail C-14
- Other Security Vulnerability Reports C-15

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Error : You are not a Valid Partner use only

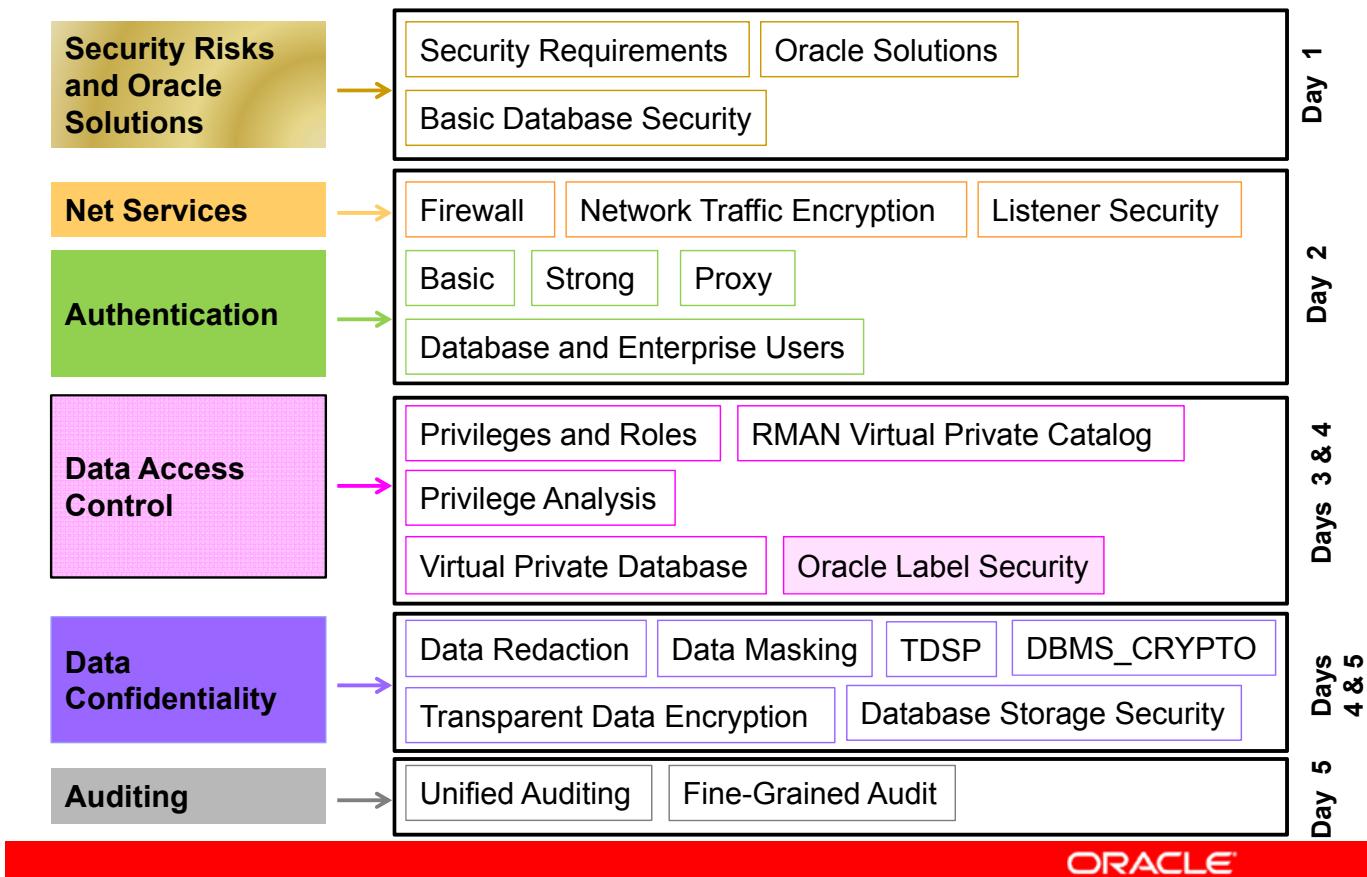
13

Implementing Oracle Label Security



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Suggested Schedule



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Enable Oracle Label Security (OLS)
- Describe OLS
 - Label concepts
 - Access mediation
- Determine when to use OLS
- Implement a simple OLS policy by:
 - Creating policies
 - Defining data labels
 - Setting up user authorizations
 - Applying policies to tables



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

This lesson is an introduction to OLS. Implementation of OLS is also presented in the lesson. Additional information can be found in the *Oracle Database Label Security Administrator's Guide*.

Access Control: Overview

Oracle provides two complementary access control models:

- Discretionary access control (DAC)
 - Allows only grant and revoke
 - Controls access on an entire object
 - Controls access by privilege
- Row-level security
 - Allows sophisticated access rules
 - Supplements DAC
 - Is provided by Virtual Private Database and OLS
- DAC and row-level security dictate row access.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

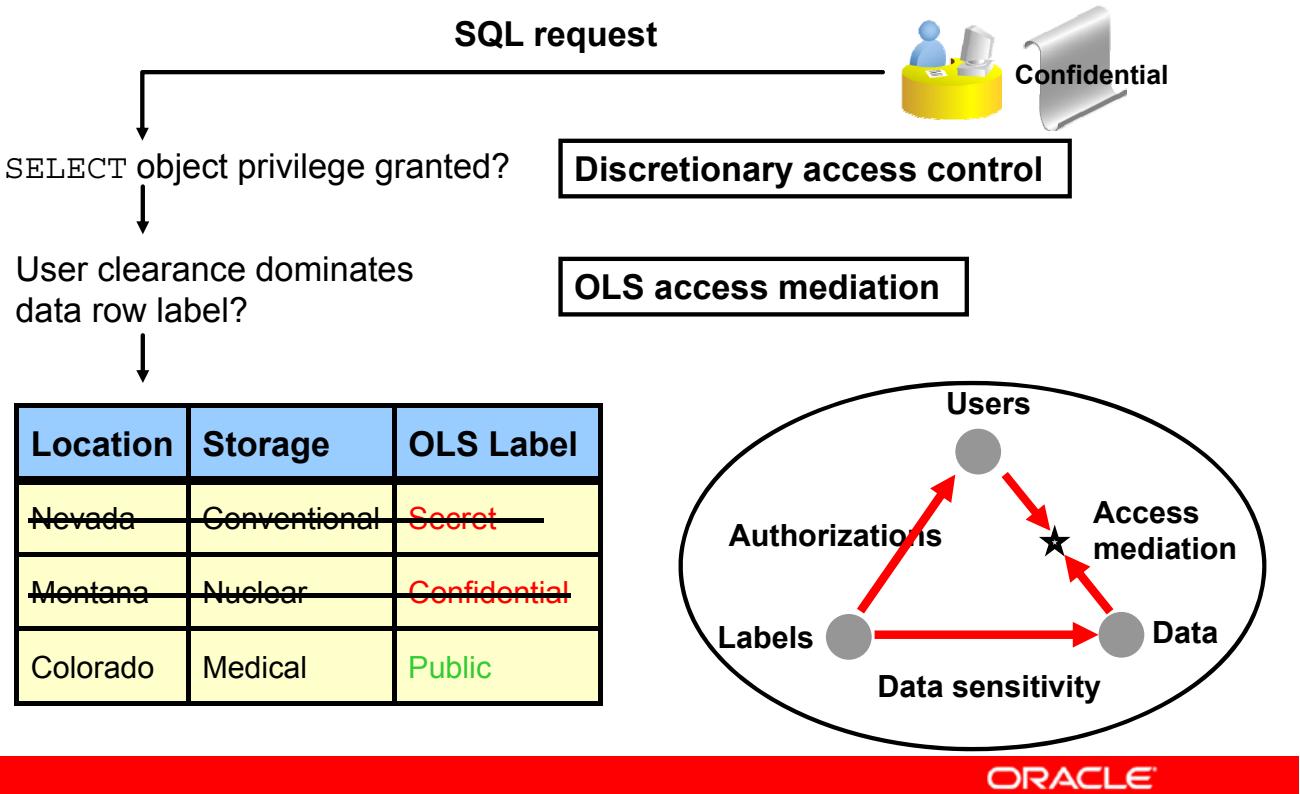
Access control defines the user's right to read, insert, update, and delete information. Every object in the Oracle database is protected by discretionary access control (DAC).

- **Discretionary access control:** DAC provides SELECT, INSERT, UPDATE, and DELETE privileges on an object to a user. The user may be granted one or more of these privileges. The limitation of DAC is that it gives access to all the data in a table or allows access to none of the data in the table.
- **Note:** There are more object privileges than are mentioned here, and some privileges can be used to restrict access on a column basis.
- **Row-level security:** This enables sophisticated access control rules beyond those of DAC by using data in the row. Virtual Private Database (VPD) and the OLS option are available to control row-level security. OLS provides sophisticated and flexible security based on row labels for fine-grained access control. OLS employs labeling concepts used by government, defense, and commercial organizations to protect sensitive information and provide data separation.

Row-level access control depends first on the basic DAC policy. DAC and row-level security together dictate the criteria, controlling whether access to a row is permitted or denied.

Oracle Database Vault without using row-level security extends DAC in ways that row-level security cannot. Oracle Database Vault enforces separation of duties, and restricts users including those with the powerful ANY privileges from seeing or modifying data in protected realms.

OLS



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

OLS does not bypass DAC but supplements it. For all users making a SQL request, DAC is applied first, then Oracle database server checks whether an OLS policy is applied. Additional predicates can easily be added to the policies to further refine access.

- **Data labels:** The objects in the database can have rows that are labeled. Data labels are used to specify the sensitivity of data. Each row has a data label. Access to rows is restricted on the basis of OLS authorizations.
- **Authorizations:** Labels are used to specify a user's "security clearance" or label authorization. Each user is assigned a set of labels that indicate the range of data labels that the user is allowed to read and write.
- **Access mediation:** The user label and the data labels are compared in a process known as access mediation that uses a set of algorithms supplied by OLS. The user is allowed to view the row when his or her label dominates the data label of that row; otherwise, he or she is not able to see the row. Whether a particular label dominates another is determined by the security administrator when the labels are created.

OLS is not enforced during the DIRECT path export and cannot be applied to objects in the SYS schema. The SYS user and users with the EXEMPT ACCESS POLICY privilege are exempt from both OLS and VPD enforcement. If your site requires that the SYS user and users with DBA-type privileges are not allowed to view application data, Oracle Database Vault has the facilities to meet this requirement. Oracle Database Vault and OLS security are designed to work together.

Enabling and Managing OLS

- Register and enable OLS:
 - Using DBCA or
 - Executing the `LBACSYS.CONFIGURE_OLS` and `LBACSYS.OLS_ENFORCEMENT.ENABLE_OLS` procedures
- The `LBACSYS` schema owns all OLS objects.
- Use Enterprise Manager Cloud Control (EMCC) or SQL*Plus to manage the Label Security policies.
- In PDBs, which can be plugged in and out of CDBs:
 - Manage OLS components on a subset of PDBs in a CDB:
 - Policies (no policies in the `root`)
 - Data labels
 - User authorizations
 - The `LBACSYS` schema is a common user schema.
 - `LBACSYS` objects are automatically available to any PDB.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

To register and enable OLS, you can use one of the following tools:

- Use the Database Configuration Assistant (DBCA)
- Connect as `SYSDBA` and execute the `LBACSYS.CONFIGURE_OLS` procedure to register the database with OLS and the `LBACSYS.OLS_ENFORCEMENT.ENABLE_OLS` procedure to activate OLS. The `LBACSYS` user is the owner of more than 200 OLS objects.

To check whether OLS has been registered, run the following statement:

```
SELECT status FROM dba_ols_status  
WHERE NAME = 'OLS_CONFIGURE_STATUS' ;
```

To check whether OLS has been enabled, run the following statement:

```
SELECT value FROM v$option WHERE parameter='OLS' ;
```

To configure OLS policies, you can use Enterprise Manager Cloud Control or SQL*Plus.

- Use Enterprise Manager Cloud Control: Enterprise Manager has pages that enable you to create policies and manage labels and policies. These pages are available from the Security section under the Administration tab on the target database home page of Cloud Control.
- Use SQL*Plus to manually execute the appropriate `LBACSYS` procedures to create policies, data labels, user authorizations, apply policies, and grant OLS privileges.

Quiz

OLS is used as an alternative to discretionary access control.

- a. True
- b. False



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: b

OLS: Features

OLS provides:

- Row-level security based on Virtual Private Database (VPD) technology
 - All required packages for access mediation
 - Complete data dictionary for managing OLS components
- A complete infrastructure for managing label security policies, sensitivity labels, and user security clearances
- Enterprise Manager pages containing a graphical user interface for managing OLS
- Integration with Oracle Identity Management starting in Oracle Database 10g Release 1



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

OLS is a packaged system that provides an easy-to-implement row-level security solution, where access control is based on data sensitivity. Security requirements are complicated by data restrictions generated by regulatory compliance. Medical data can be viewed only by attending medical professionals and by the person whose data it is. Nuclear weapon data is restricted to persons who have the proper security clearance and compartment endorsement. Different users need to see different data depending on their jobs, positions, or roles. When this data can be separated into different tables, DAC can provide the required granularity. When the data restrictions are based on row and column values, the need for fine-grained access becomes obvious.

Row-Level Security

OLS is built on the fine-grained access control technology of VPD. The major advantage of using OLS is that OLS is a complete system. It is a ready-to-use VPD. OLS provides sophisticated functions and procedures for evaluating and comparing sensitivity labels. It provides a sophisticated infrastructure for storing and managing sensitivity labels and user security clearances.

Integration with Oracle Identity Management

The previous releases of OLS have relied on the Oracle database as the central repository for policy and user label authorizations. This architecture took advantage of the scalability and high availability of the Oracle database, but did not make use of the Oracle Identity Management infrastructure, which includes Oracle Internet Directory. This directory is part of the Oracle Identity Management platform. Integrating your installation of OLS with Oracle Internet Directory allows label authorizations to be part of your standard provisioning process.

For sites that use Oracle Internet Directory, database servers retrieve OLS policy information from the directory. Administrators use the olsadmintool policy administration tool to operate directly on the directory to insert, alter, or remove metadata as needed. Because enterprise users can log in to multiple databases by using the credentials stored in Oracle Internet Directory, it is logical to store their OLS policy authorizations and privileges there as well. An administrator can then modify these authorizations and privileges simply by updating these metadata in the directory. (Other aspects of managing enterprise users are performed through the Oracle Identity Management Provisioning console.)

For distributed databases, centralized policy management removes the need for replicating policies because the appropriate policy information is available in the directory. Policy changes in the directory are synchronized with policy information in the databases by means of the Directory Integration Platform and are effective without further effort.

The following OLS information is stored in the directory:

- Policy information, namely, policy name, column name, policy enforcement options, and audit options
- User profiles identifying their labels and privileges
- Policy label components: levels, compartments, and groups
- Policy data labels

The database-specific metadata is not stored in the directory. The examples include:

- Lists of schemas or tables, with associated policy information
- Program units, with associated policy privileges

OLS and VPD Comparison

	VPD	OLS
Access Control	Based on the WHERE clause	Compare data labels with user clearances.
Implemented	With user-programmed policies	With provided code; no coding needed
Table changes	No columns added	Data label column added (hidden)
New Data Classification	None	Automatic
In addition to DAC	Policies automatically applied	User clearances automatically applied
Column-level control	Uses column-level VPD	May be used with column-level VPD
Oracle Database Vault		User clearance may be used as a factor to control access.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

OLS is built on the same technology as that used for the VPD. OLS does not depend on preexisting data attributes as the basis for access control, but depends on assigned data labels and user clearances. Label Security adds a column to every table protected. This column can be a hidden column.

Every application of VPD is custom built. VPD provides row-level access control by using application context and a WHERE clause that is added to every SQL statement. OLS provides the packages required to implement the customer requirements.

When new data is added to the protected table, OLS assigns data labels based on the user clearance automatically. If new values are placed in the columns used by VPD, the WHERE clause in the policy may need to be changed.

OLS is designed to work with column-level VPD and Oracle Database Vault. A column-level policy may be applied to further restrict column access, and user clearances may be used as factors in Oracle Database Vault to limit access to schemas and commands.

Analyzing Application Requirements

- Identify application tables that need OLS:
 - The majority of the tables DO NOT require OLS.
 - Use existing tools when possible.
 - Do not apply OLS to everything.
 - Identify important application queries where possible.
- Discretionary access control (DAC) is sufficient for most tables:
 - Database roles
 - Secure application roles
 - Stored procedures and functions
 - Oracle Database Vault



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

- **Identify the application tables that need OLS:** Usually, only a very few tables hold data that requires the protection provided by OLS. Do not apply OLS where it is not needed. Use the appropriate security technology for your situation. The built-in technologies described below frequently meet the business-security requirements with minimal overhead.
OLS has a performance cost. Identify the most resource-intensive application queries and tune them for use in the OLS environment.
- **Discretionary access control (DAC) is sufficient for most tables:** DAC is always applied before the OLS policies. DAC specifies access control privileges at the object level.
 - **Database roles and secure application roles:** These have been covered in the previous lessons.
 - **Stored procedures and functions:** These can be used to encapsulate objects, allowing the owner to expose only certain methods of accessing the object. This technique can provide very tight control over data integrity.
 - **Oracle Database Vault:** This can be used to extend DAC in ways that OLS cannot. Oracle Database Vault is covered in detail in the course titled *Implementing Oracle Database Vault*.

Implementing an OLS Solution

The steps to implement an OLS solution are:

1. Develop a strategy to understand the security problem.
2. Analyze the data levels in the application.
3. Create policies: SA_SYSDBA.CREATE_POLICY
4. Define data labels: The level is required.

Level: Secret	Compartment: Finance	Group: Europe
----------------------	-----------------------------	----------------------

5. Assign user authorizations for each policy.
6. Apply the policies on a table.
7. Assign a data label to each row of the table in the OLS column.
8. Test thoroughly, review and document your decisions.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

1. **Develop a strategy:** Talk to the right people. Identify those individuals in your organization who really understand the business-security problem. Make sure that you understand the problem before adding additional security to your application.
2. **Analyze the data to be protected:** Ask the following questions:
 - Where does the sensitive data reside in the application (which tables)?
 - Who needs access to this data?
 - Who owns the data?
 - Who should be able to read the data?
 - Who should be able to make updates?

This analysis includes a grouping of the user community by access needs:

- Does that grouping follow organizational lines?
- Does it depend on the job function?

This process is repeated for each set of data that is to be protected.

3. **Create policies:** Name the policies and define the default policy enforcement options.

4. **Define labels:** When the OLS policy is applied to a table, a label column is added to the table. The value in this label column is compared to the access permissions assigned to the user to determine which users have access to that row. The labels have three parts: level, compartment, and group. Each of these can be defined for your site. After the labels have been defined, you must assign the data labels to the data rows. This is not automatic.
The Data labels must be applied to the table. That means update the rows of the table by assigning the data label to every row, before the policy is applied. If the data label column is null, that row will not be accessible.
Note: Label definitions are global. Therefore, two policies in the same database cannot define the same security label with different values. This means that if you try to run two OLS-enabled applications against the same database, you must have considered this, or label collisions occur.
5. **Assign user authorizations:** A user authorization is the range of labels that a user can access. They are created and assigned to the user on the basis of access requirements. Special privileges are included in this set of authorizations. After the policy has been applied, no user can access the data without a set of authorizations. This step is independent of applying the policy; user authorizations may be assigned before or after the policy is applied.
6. **Apply policies:** A policy must be applied to the table to be enforced. Policies can be administered by users with appropriate privileges, such as a security officer. Policies can be enforced, disabled, applied, or removed. In a single database, several OLS policies can be protecting data. Each of these policies can be applied to different tables, and multiple policies may be applied to a single table. To access data, a user must have label authorizations (or clearances) for all policies protecting that data. To access any particular row, you must be authorized by *all* policies protecting the table containing your desired rows. The security officer must be given specific permissions to create and administer policies and labels. These actions are performed by using the Oracle Policy Manager interface or PL/SQL packaged procedures. See the *Oracle Database Label Security Administrator's Guide* for a full description of the packages, procedures, and required privileges.
7. **Update the OLS column of each row** in the table where an OLS policy is applied with an available data label. Choose a relevant label according to the sensitivity of each row.
8. **Test, review, and document your policy decisions:** Test to verify that the results are as expected when SELECT, INSERT, UPDATE and DELETE statements are executed on a table with an OLS policy. This documentation provides a reference point for future changes and audits. Implementing label security can be complex. Many seemingly small decisions are made for ease of use or performance. When these policies are called into question, the documentation saves many hours of reanalyzing the application.

Create Policies With Enforcement Options

Create the policy to contain the label information:

- Set the policy name.
- Set the policy label column.
- Set the default access options:
 - Access-control enforcement:
 - READ_CONTROL
 - WRITE_CONTROL
 - Label-management enforcement:
 - LABEL_DEFAULT
 - LABEL_UPDATE
 - CHECK_CONTROL
 - Options to override enforcement:
 - ALL_CONTROL / NO_CONTROL



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The first step in setting up OLS is to create policies. The named policy is a container for all the information that is associated with a policy: labels, tables, views, privileges, and procedures.

Creating the policy requires a policy-specific column name and specifies a set of default policy options. The column is added to every table associated with the policy.

```
EXEC SA_SYSDBA.CREATE_POLICY ( policy_name => 'facility',
                                column_name => 'fac_col',
                                default_options => 'read_control,
                                update_control')
```

Policy enforcement options can be set at the policy level as shown or at the schema or table level when the policy is applied.

This is a brief introduction to the policy-enforcement options. For more details, see the *Oracle Database Label Security Administrator Guide*.

A basic policy with full enforcement would have the READ_CONTROL, WRITE_CONTROL, and LABEL_DEFAULT options enforced.

The interaction of the various enforcement options should be well understood for effectively designing an OLS system.

Access-control enforcement controls read and write access to the data. The policies can be set for each type of data manipulation language (DML):

- **READ_CONTROL** enforces the policy for all queries, controlling which data rows are accessible for SELECT, UPDATE, and DELETE. If READ_CONTROL is OFF on a policy, for any table protected by the policy, all rows are accessible to all users.
- **WRITE_CONTROL** determines the ability to insert, update, and delete data in a row. If this option is active, it enforces INSERT_CONTROL, UPDATE_CONTROL, and DELETE_CONTROL. You can apply INSERT_CONTROL, UPDATE_CONTROL, and DELETE_CONTROL separately.

Label-management enforcement ensures that data labels written for inserted or updated rows do not violate policies set for such labels:

- **LABEL_DEFAULT** uses the session's default row label value unless the user explicitly specifies a label on INSERT.
- **LABEL_UPDATE** applies policy enforcement to the UPDATE operations that set or change the value of a label attached to a row. The WRITEUP, WRITEDOWN, and WRITEACROSS privileges are enforced only if the LABEL_UPDATE option is active. Note that any label function that is in force overrides the LABEL_UPDATE option.
- **CHECK_CONTROL** applies the READ_CONTROL policy enforcement to the INSERT and UPDATE statements to ensure that the new row label is read-accessible by the user that is changing the label.

Options to override enforcement can suspend or apply all other enforcement options:

- **ALL_CONTROL** applies all enforcement options.
- **NO_CONTROL** applies no enforcement options. A labeling function or a SQL predicate can still be applied. In the absence of a label function, data rows that are inserted do not have a label. If enforcement options are turned on later, the unlabeled rows are not visible unless the user has the READ or FULL access authorization.

Set Policy Enforcement Options at the policy level as shown in the previous slide with the DEFAULT_OPTIONS parameter. These options will be used unless other options are specified when the policy is applied to the table or schema.

Note: Table-level options take precedence over the schema-level options.

Define Labels

Labels have three parts.

S : FIN : As, Eu

Level	:	Compartment	:	Group
-------	---	-------------	---	-------

- Every label must have a **level**. **P::**
- The label is defined on the combinations of the parts. **S : : WR_S**

```
SA_COMPONENTS.CREATE_LEVEL ( POLICY_NAME =>'FACILITY',
    LEVEL_NUMBER => '400', SHORT_NAME => 'S', LONG_NAME => 'SECRET')
```

```
SA_COMPONENTS.CREATE_COMPARTMENT( POLICY_NAME =>'FACILITY',
    COMP_NUMBER => '85', SHORT_NAME => 'FIN', LONG_NAME => 'Financial')
```

```
SA_COMPONENTS.CREATE_GROUP( POLICY_NAME =>'FACILITY',
    GROUP_NUMBER => '1000', SHORT_NAME => 'WR_S', LONG_NAME => 'WR_SALES',
    PARENT_NAME => 'WR')
```

```
SA_LABEL_ADMIN.CREATE_LABEL(POLICY_NAME=>'FACILITY', LABEL_VALUE=>'S::WR')
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Each data label can have three parts: a level, one or more compartments, and one or more groups. Every label must have a level, but the compartment and group portions of the label are optional. Each level, compartment, and group that will be used in a label must be created before it can be used in a label.

Defining the required levels, groups, and compartments follows the analysis of the data-security needs. Each part of the label is defined. The following pages discuss:

- Defining **levels**:
 - A level is an arbitrary name, such as SECRET or CONFIDENTIAL. Higher and lower levels are determined by the “tag” (the numeric form of the level). The numeric form can range from 0 through 9999. A user with a higher level can access lower levels. Levels have ranking determined by the numeric tag.
 - Assume that only levels are used. A user with an authorization on SECRET level may access data with a SECRET level or below.
 - Each policy has its own set of levels.

- Defining **groups**:
 - A data label can have zero or more groups assigned to it. Groups are a means of making a hierarchical assignment. Data assigned to a group can be accessed by a user belonging to that group or a parent of the group.
 - In the example, WESTERN_REGION is the parent group to WR_FINANCE and WR_SALES. A user belonging to the WESTERN_REGION group can access data belonging to either the WR_FINANCE group or the WR_SALES group. But a user belonging to the WR_SALES group cannot access data belonging to WR_FINANCE or WESTERN_REGION.
 - A label can include multiple groups. A user having permission on ANY of the groups in the data label is allowed access. The numeric form does not indicate any type of ranking. It does not indicate a parent-child relationship, or greater or less sensitivity. It controls only the display order of the short form of the group name in the label character string.
- Note:** The group access is evaluated after the level is evaluated. All labels have a level. So access is further restricted by standard groups. Group access does not give access when access would be denied by the level.
- Defining **compartments**: Compartments provide a finer granularity to the level classification. To access rows with compartments included in the data label, the user label must include the level and the compartment. The following are the characteristics of compartments:
 - Compartments are independent classifications. A data label can have zero or more compartments assigned to it. For a user to access that row, the user authorization must first have the correct level and include ALL the compartments. Compartments can be used to separate the work of various projects.
 - In the example, a data row has a label with two compartments: OP and CH. To access the row, the user must have a label with the same or higher level, and at least the two compartments (OP and CH).
 - The numeric tags for compartments do not indicate a higher or lower classification. This tag only provides a way to order the compartment-name strings in a label. For example, FIN is not higher than OP, but in a label, OP appears before FIN:
S:OP, FIN.
- Creating **data labels** by combining particular sets of levels, compartments, and groups. The short form is used in the LABEL_VALUE parameter of the CREATE_LABEL procedure. When a valid data label is created:
 - The label is automatically designated as a valid data label. This functionality limits the labels that can be assigned to data. OLS can also create valid data labels dynamically at run time from those components that are predefined in Oracle Internet Directory. Most users, however, prefer to create the labels manually to limit data-label proliferation.
 - A numeric label tag is associated with the text string representing the label. It is this numeric label tag (not the text string) that is stored in the policy label column of the protected table. This tag must be unique across all policies in the database. It is a good practice for performance reasons to set tags for labels of higher sensitivity to a higher numeric value than tags for labels of lower sensitivity.

Assign User Authorization Labels

A user is assigned:

- Maximum and minimum labels
- A default session label
- A row label for inserts

```
EXEC SA_USER_ADMIN.SET_USER_LABELS
  ( POLICY_NAME =>'FACILITY',
    USER_NAME => 'MYCO_MGR',
    MAX_READ_LABEL =>'S::US,EU,ASIA')
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

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

Each user has a session label and a row label. The session label is the particular combination of levels, compartments, and groups on which a user works at any given time. The user can change the session label to any combination of components for which he or she is authorized.

When a user writes data without specifying its label, a row label is assigned automatically, using the user's session label. However, the user can set the label for the written row within certain restrictions on the components of the label that he or she specifies with the `SA_SESSION.SET_ROW_LABEL` procedure.

Find on the next page all configurable label authorizations.

- **MAX_READ_LABEL:** Specifies the label string to be used to initialize the user's maximum authorized read label. It is composed of the user's maximum level. Compartments are authorized for read access, and groups are authorized for read access.
- **MAX_WRITE_LABEL:** Specifies the label string to be used to initialize the user's maximum authorized write label. It is composed of the user's maximum level. Compartments are authorized for write access, and groups are authorized for write access. If **MAX_WRITE_LABEL** is not specified, it is set to **MAX_READ_LABEL**.
- **MIN_WRITE_LABEL:** Specifies the label string to be used to initialize the user's minimum authorized write label. It contains only the level, with no compartments or groups. If **MIN_WRITE_LABEL** is not specified, it is set to the lowest defined level for the policy, with no compartments or groups.
- **DEF_LABEL:** It is the label string including level, compartments, and groups that determines the initial session setting for what a user can see. It can be increased by the user up to **MAX_READ_LABEL**. If **DEF_LABEL** is not specified, it is set to **MAX_READ_LABEL**.
- **POLICY_NAME:** Specifies the policy
- **USER_NAME:** Specifies the user name
- **ROW_LABEL:** It is the label string including level, compartments, and groups put on data that the user inserts if he or she does not specify the data label as a field in the **INSERT** statement. If **ROW_LABEL** is not specified, it is set to **DEF_LABEL**, with only the compartments and groups authorized for write access.

Apply the Policy to a Table

- Add the FACILITY policy to the LOCATIONS table.
- TABLE_OPTION => NULL implies that the policy default options are used.

```
EXEC SA_POLICY_ADMIN.APPLY_TABLE_POLICY (
    POLICY_NAME => 'FACILITY',
    SCHEMA_NAME => 'HR',
    TABLE_NAME   => 'LOCATIONS',
    TABLE_OPTIONS => NULL,
    LABEL_FUNCTION => NULL)
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

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

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

Use the APPLY_TABLE_POLICY procedure or Enterprise Manager to add the specified policy to a table. A policy label column is added to the table if it does not exist, and is set to NULL. When a policy is applied, it is automatically enabled. The policy enforcement options are specified with a comma-delimited list in the TABLE_OPTIONS parameter. To change the table options, labeling function, or predicate, you must first remove the policy, and then reapply it.

Adding Labels to Data

- Labels are defined by the administrator.
- Access mediation requires all rows to have labels.
- Labels are set on rows.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The components of the labels have been created. The data labels have been created and marked as valid. For access mediation to work properly, the individual rows must have a label assigned.

- **Labels must be defined:** The labels that are assigned to data rows must first be created. There are usually many more permutations of the different components of the labels than are actually used. Most sites require that the labels that are actually used be created by an administrator to control the proliferation of labels.
- **All rows must have labels:** When creating policies, the label column for existing rows is initially `NONE`. The `NONE` value does not match any label, so the data is not accessible, except by users with the `FULL` access privilege.
- **Set labels by updating rows:** For existing rows, a user who has full access privileges (typically, the security administrator) updates the rows, setting the label column to the proper label value for that row. For new rows, users or the application supplies the label, either directly by a pick list, defaulted to the session labels, or by a policy function.

Access Mediation

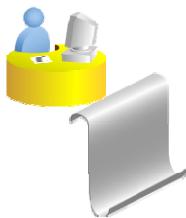
The user authorization is assigned the Confidential level:

C :: WR_S

The user label dominates the data labels:

- Public => Access Granted
- Confidential => Access Granted

The user label does not dominate **S ::** => Access Denied



Location	Storage	OLS Label
Nevada	Conventional	S ::
Montana	Nuclear	C ::
Colorado	Medical	P ::

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The process of comparing user authorization and data label to decide what access is granted is called *access mediation*.

There are two types of access mediation for protected tables: read and write. A user can read any data up to his or her maximum level. Write access is a subset of read access. A user cannot write lower than his or her minimum level. This controls the user's ability to disseminate data by lowering its sensitivity.

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

You can further customize user data access by granting policy privileges and setting policy-enforcement options.

Quiz

Which of the following steps are optional when implementing OLS?

- a. Create the policy.
- b. Define data label levels.
- c. Define data label groups.
- d. Define data label compartments.
- e. Apply the policy to a table.
- f. Assign user authorization labels.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: c, d

OLS Special User Privileges

- OLS supports these privileges that allow authorized users to bypass certain parts of the policy.
- Set the following privileges with the `SA_USER_ADMIN.SET_USER_PRIVS` procedure:
 - READ
 - FULL
 - COMPACCESS
- Set `PROFILE_ACCESS` with the `SA_POLICY_ADMIN.SET_ACCESS_PROFILE` procedure.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The first set of Label Security privileges are set with the `SA_USER_ADMIN.SET_USER_PRIVS` procedure.

- **READ:** Allows read access to all data protected by the policy
- **FULL:** Allows full read and write access to all data protected by the policy
- **COMPACCESS:** Allows a session access to data authorized by the row's compartments, independent of the row groups

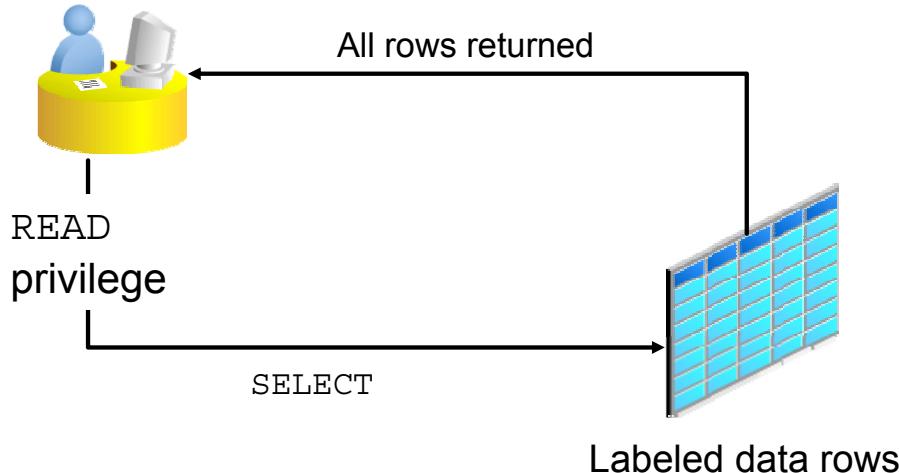
`PROFILE_ACCESS` is set with the `SA_POLICY_ADMIN.SET_ACCESS_PROFILE` procedure.

PROFILE_ACCESS: Allows a user to change the OLS authorizations and privileges of the database session to those of the specified user

There are additional special privileges, described in the *Oracle Database Label Security Administrator's Guide*.

Example: READ Privilege

User-label authorizations: *None*



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

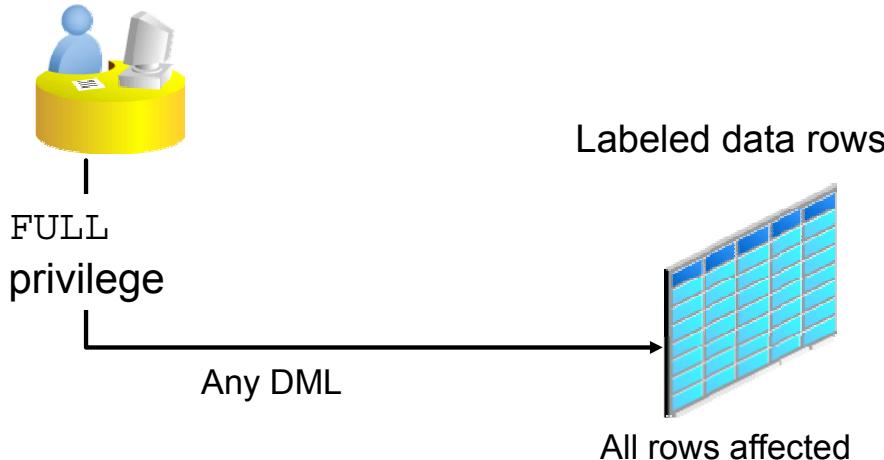
The READ privilege enables the user to bypass the OLS policy entirely for read access to data. A user with the READ privilege can read all data protected by the policy, regardless of his or her authorizations or session label. The user does not even need to have label authorizations. However, access mediation is still enforced on the UPDATE, INSERT, and DELETE operations. A user with the READ privilege can write only to data rows for which he or she has write access, based on any label authorizations.

Application uses:

- Data export
- Report generation
- Executive management privilege

Example: FULL Privilege

User-label authorizations: Any



ORACLE

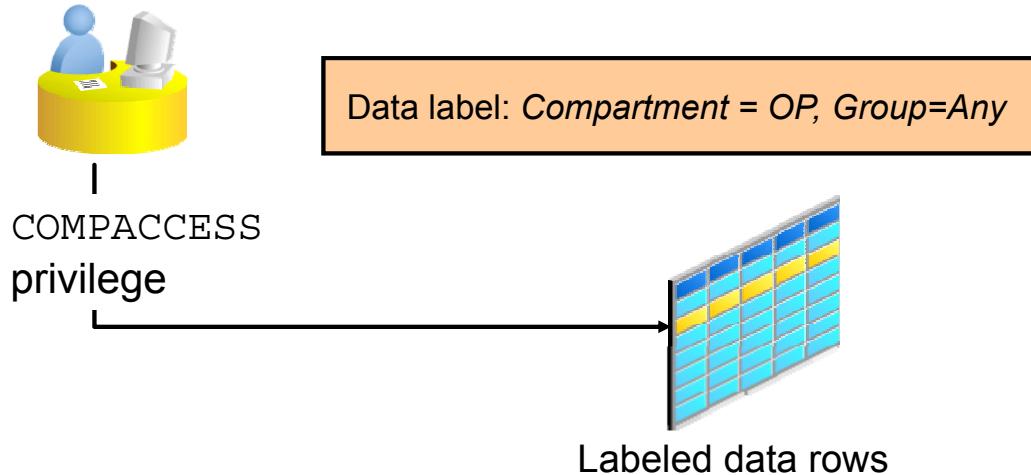
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The FULL privilege has the same effect and benefits as the READ privilege, with one difference: a user with the FULL privilege can also write to all the data. The ability to write effectively bypasses all OLS controls. Oracle Database discretionary access controls still protect the underlying table. For example, if a user does not have the UPDATE privilege on the underlying table and attempts to update a table directly with an UPDATE SQL statement, the statement would fail.

This is a very powerful privilege and should be reserved for only users that require it. A classifier (someone who reviews data to determine its security classification) would need this privilege to be allowed to see the data and change the classification freely.

Example: COMPACCESS Privilege

User-label authorizations: *Compartment = OP*



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The COMPACCESS privilege allows a session to access rows on the basis of the row's compartments, independent of the row groups. If a row has no compartments, access is determined by the group authorizations. However, when compartments exist, and access to them is authorized, the group authorization is bypassed. Level authorizations are still enforced.

As in the example in the slide, if the row has a data label of Confidential:Operations:Western_Region and the user label is Confidential:Operations:Central_Region, the user can access the row on the basis of the compartment. The group is ignored.

This privilege is required only in special situations—for example, where a compartment is created for a project that crosses groups but does not include all members of each group.

Using the PROFILE_ACCESS Privilege

The SA_SESSION.SET_ACCESS_PROFILE function in OLS:

- Allows an application session to assume a different OLS authorization

```
SQL> connect appuser/mypassword
SQL> begin
      sa_session.set_access_profile('facility', 'jim');
    end;
```

- Is used when application users do not have real database accounts

Note: Users who are assigned OLS authorizations do not need to be real database users.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The SA_SESSION.SET_ACCESS_PROFILE procedure sets the OLS authorizations and privileges of the database session to those of the specified user. (Note that the originating user retains the PROFILE_ACCESS privilege.)

The user executing the procedure must have the PROFILE_ACCESS privilege. Note that the database user (the session user ID) who has logged in does not change. That user assumes only the authorizations and privileges of the specified user.

Shared Schema Support

User accounts defined in Oracle Internet Directory cannot be given individual OLS authorizations. However, authorizations can be given to the shared schema to which the directory users are mapped. The OLS SET_ACCESS_PROFILE function can be used programmatically to set the label authorization profile after a user has been authenticated and mapped to a shared schema. OLS does not enforce a mapping between users who are given label authorizations in OLS and actual database users.

This administrative procedure is useful for various tasks:

- The administrator can see the result of the authorization and privilege settings for a particular user.
- Applications that have proxy accounts connect as (and assume the identity of) application users for purposes of accessing labeled data. The proxy account can act on behalf of application users.

Trusted Stored Package Units

Trusted stored package units can extend the privileges of users in a specific and controlled manner.

To create a trusted stored package unit, you must:

- Have the special *<policy>*_DBA role
- Grant the OLS privileges to a program unit
- Use Enterprise Manager or the SA_USER_ADMIN package to grant privileges

```
SQL> EXEC SA_USER_ADMIN.SET_PROG_PRIVS( -  
      POLICY_NAME=>'HR', -  
      SCHEMA_NAME=>'MYSCHHEMA', -  
      PROGRAM_UNIT_NAME =>'SUM_PURCHASES', -  
      PRIVILEGE=>'READ') ;
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

A trusted stored program unit is a stored procedure, function, or package that has been granted one or more OLS privileges. Trusted stored program units are typically used to enable users to downgrade information in a controlled manner, or update data at several levels. This is the optimal way in which users can be enabled to access data beyond their authorization.

To grant privileges to a stored program unit, you must have the special *<policy>*_DBA role (where *policy* is the name of a policy) and the EXECUTE permission on the program unit.

Use either Enterprise Manager or the SA_USER_ADMIN package to grant privileges to a program unit. The SA_USER_ADMIN.SET_PROG_PRIVS procedure sets policy-specific privileges for program units.

In the example in the slide, the SUM_PURCHASES procedure has been granted the READ privilege. When the SUM_PURCHASES procedure is called, it executes with the READ privilege as well as the current user's OLS privileges. This allows the total purchases to be calculated.

Exporting and Importing with OLS

- Full database export and import
 - LBACSYS metadata is included in the dumpfile.
 - You must register and enable OLS before importing.
- Schema and table-level export:
 - Only rows with labels authorized for read access are exported.
 - Privileges such as FULL or READ give complete access.
 - Users must have EXEMPT ACCESS POLICY privilege to export all rows in the table, or else **no rows** are exported.
 - The HIDE property of a policy is not exported.
 - You must define in the import database all of the label components and individual labels used in tables being imported.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Starting with Oracle Database 12c, OLS metadata export and import provide the ability to move OLS policies and protected tables between databases.

Full Database Export and Import

OLS metadata in the LBACSYS schema is included when doing a **full** database export and import operation. The source database can be Oracle Database 11g Release 2 (11.2.0.3), or higher, but the target database must be Oracle Database 12c or higher. Before starting the Data Pump import on the target database, you must enable OLS.

Schema and Table-Level Export and Import

- Only rows with labels authorized for read access are exported. To export all the data, the user must be granted the READ or FULL access. A user with the EXEMPT ACCESS POLICY privilege can also export all the data.
- The HIDE property is not exported. The label columns of protected tables are exported (as numeric values). If a label column is *hidden*, it is exported as a normal, unhidden column.
- Create any OLS policies that protect the data to be imported. The policies must use the same column names as in the export database.
- In the import database, define all the label components and individual labels used in the tables imported. Tag values assigned to the policy labels in each database must be the same. To successfully import, the user importing must be authorized for all the labels required to insert the data and labels contained in the export file.

Performance Tips

- Limit policies to the required tables.
- Plan a label tag strategy.
- Analyze the LBACSYS schema.
- Consider adding a label column to the existing indexes.
- Consider applying a bitmap index on the label column.
- Partition on the basis of the label.
- Allow time to tune your application after applying OLS.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

- **Limit policies to required tables:** In most cases, only a small subset of the tables require row-level security. Carefully identify these tables and limit the policies to these. **Warning:** The policies you add will directly affect performance. Use them wisely.
- **Planning a Label Tag Strategy:** For optimal performance, you can plan a strategy for assigning values to label tags. In general, it is best to assign higher numeric values to labels with higher sensitivity levels. Usually, many more users can see data at comparatively low levels; fewer users at higher levels can see many levels of data.
- With READ_CONTROL set, OLS generates a predicate that uses a BETWEEN clause to restrict the rows to be processed by the query. If the higher-sensitivity labels do not have a higher label tag than the lower-sensitivity labels, the query potentially examines a larger set of rows. This affects performance by requiring more reads.
- **Analyzing the LBACSYS schema:** Run the DBMS_STATS.GATHER_SCHEMA_STATS procedure on the LBACSYS schema, so that the cost-based optimizer can improve execution plans on queries.
- **Indexing the Policy Label Column:** Create a bitmap index on the policy label column on the basis of the number of distinct values.
- **Partitioning on the Basis of the Label:** If you are using a numeric ordering strategy with the numeric label tags that you have applied to the labels, you can use this as a basis for data partitioning. Depending on the application, partitioning data on the basis of label values may or may not be useful.

Summary

In this lesson, you should have learned how to:

- Enable Oracle Label Security (OLS)
- Describe OLS
 - Label concepts
 - Access mediation
- Determine when to use OLS
- Implement a simple OLS policy by:
 - Creating policies
 - Defining data labels
 - Setting up user authorizations
 - Applying policies to tables



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Practice: Overview

This practice covers the following topics:

- 13-1: Registering and enabling OLS
- 13-2: Implementing and testing using EM Cloud Control and packages
- 13-3: Dropping OLS policies



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Error : You are not a Valid Partner use only

Module

Data Confidentiality

A solid red horizontal bar spanning most of the page width, positioned below the main title area.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

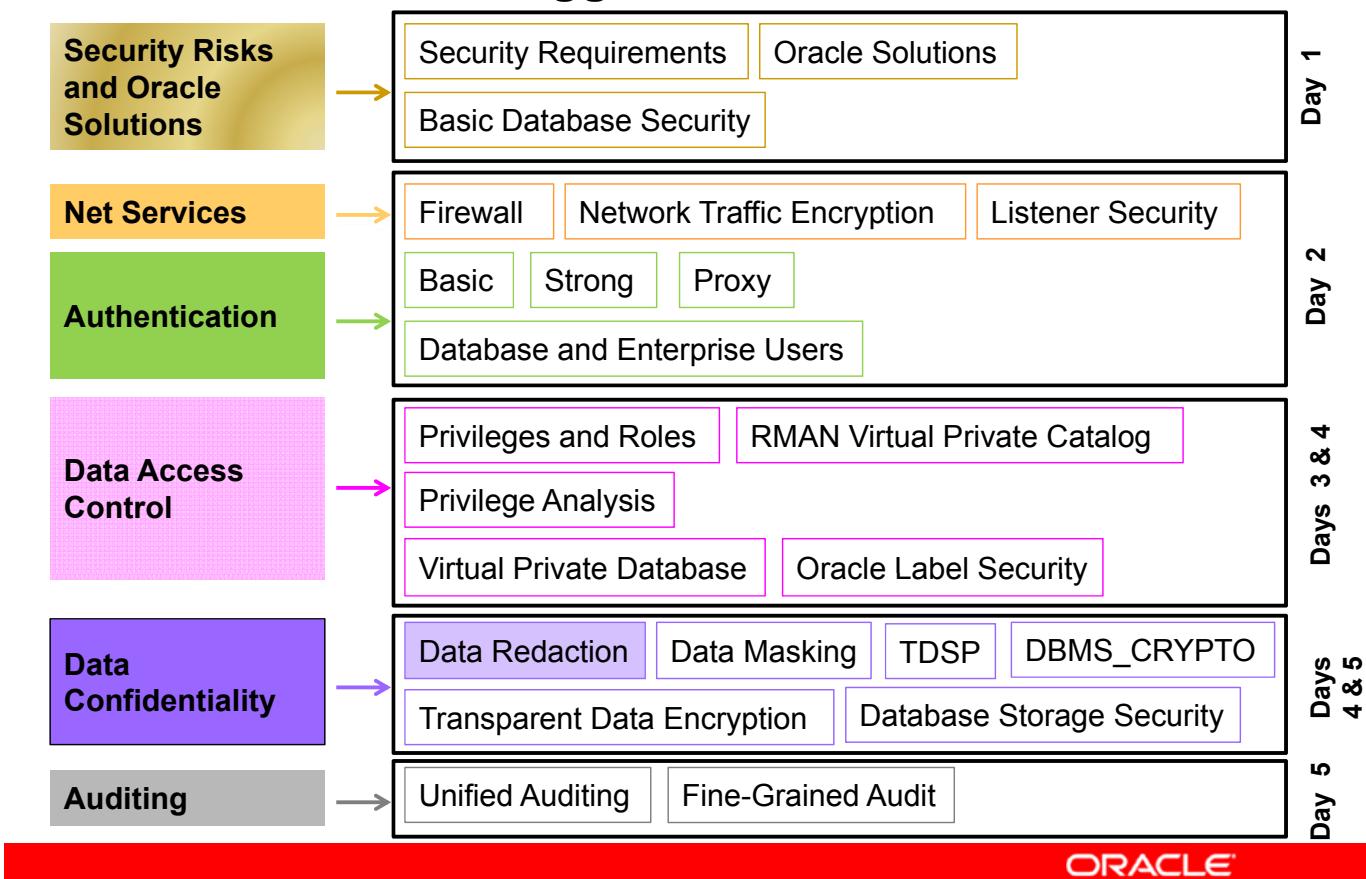
14

Oracle Data Redaction

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Suggested Schedule



ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Describe Oracle Data Redaction
- Manage redaction policies
- View redaction policy information in the data dictionary



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Note

For a complete understanding of the Oracle Data Redaction new feature usage, refer to the following guide in the Oracle documentation:

- *Oracle Database Advanced Security Guide 12c Release 1 – “Part II, Using Oracle Data Redaction”*
- *Oracle Database 2 Day course – “Limiting Access to Sensitive Data Using Oracle Data Redaction”*
- *Oracle Database PL/SQL Packages and Types Reference 12c Release 1 (12.1) – “DBMS_REDACT” chapter*

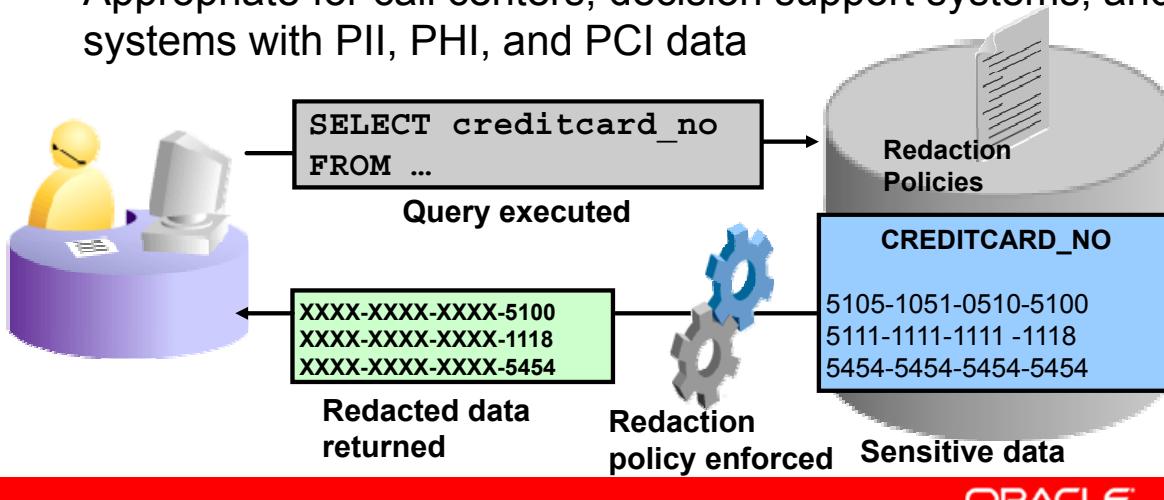
Refer to other sources of information available under Oracle Learning Library:

- *Oracle Database 12c New Features Demo Series* demonstrations:
 - *Data Redaction Basics*
- *Oracle By Example (OBE)*:
 - *Protecting Data with Data Redaction*
 - *Redaction Management in Oracle SQL Developer*

Note: Oracle Data Redaction is being backported to release 11.2.0.4.

Oracle Data Redaction: Overview

- On-the-fly redaction based on username, IP address, application context, and other factors
- Transparent, consistent enforcement in the database
- High performance for production applications
- Appropriate for call centers, decision support systems, and systems with PII, PHI, and PCI data



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Many Oracle Database customers currently prevent the display of sensitive data to end users by performing redacting in each application. Oracle Data Redaction moves this functionality from the application to the database. This approach has several benefits over redacting data in the application tier, and it is useful in a variety of application scenarios.

Oracle Data Redaction is a transparent, flexible, and simple solution. It modifies sensitive data columns contained in SQL query results on-the-fly right before the results are returned to applications. The columns are redacted according to flexible policies that provide conditional redaction. The policies are managed directly within the database. For maximum transparency, redaction preserves the returned column data type and formatting, and it does not alter the underlying data blocks on disk or in cache. Oracle Data Redaction is designed to be fast so that it can be used on production systems. In addition, it is embedded in the database management system, so no separate installation is required.

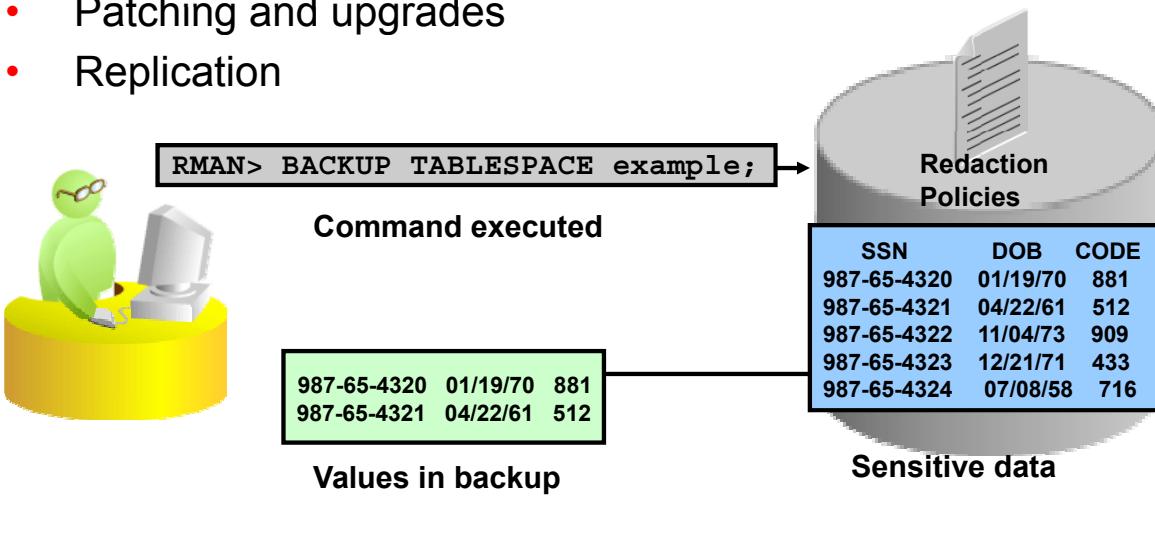
Oracle Data Redaction is useful for many different scenarios. It can be applied to a range of packaged and custom applications to redact application screens, dashboards, and reports. It helps you avoid making code changes in existing call center, human resources, sales, and financial and healthcare applications. These applications frequently manage payment card (PCI), protected health (PHI), or personally identifiable (PII) information that is subject to regulation. Oracle Data Redaction is also useful for decision support systems that aggregate large and diverse sets of data in a single repository for running analytics.

When an application issues a query, data is retrieved from the database and then the redaction policy is applied. Redaction takes place immediately preceding the return of selected data, and only at the top level of the `SELECT` list.

Oracle Data Redaction and Operational Activities

Operational activities that are not subjected to redaction:

- Backup and restore
- Import and export
- Patching and upgrades
- Replication



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Data retrieved during certain operational activities should not be redacted. Connections as the `SYS` user are always exempt from redaction policies. Utilities such as Data Pump perform recursive SQL as the `SYS` user, so redaction policies are not applied for export and import operations.

In this example, an RMAN command to back up the `EXAMPLE` tablespace is executed. The data is retrieved from the database, but is not redacted in the backup file.

Note: Data Redaction policies associated with tables and views are included in the export and import operations. Therefore, the policies are enabled and the data is redacted when users query the objects in the imported database.

Available Redaction Methods

Type	Description
None	No redaction is performed.
Full	Columns are redacted to constant values based on the column data type.
Partial	User-specified positions are replaced by a user-specified character.
Random	Data type is preserved and different values are output each time.
Regular Expression	A “match and replace” is performed based on parameters.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Oracle Data Redaction supports four types of redaction: full, partial, random, and regular expression.

Support for a redaction type of `NONE` is provided so that you can create redaction policy definitions and evaluate their impact on the application functionality and performance. Type `NONE` also functions as an override. For example, when multiple hierachal views are being redacted, `NONE` can override redaction that was inherited from a lower level view and cause the column to display un-redacted data.

Oracle Data Redaction: Examples

	Data in the Database	→ Redacted Values
Full Redaction	05/24/75 11 Rock Bluff Drive	→ 01/01/01 → xxxxxxxx
Partial Redaction	068-35-2299 D1L86YZV8K	→ ***-**-2299 → D1*****8K
Regular Expression Redaction	94025-2450 Tom.Lee@acme.com	→ 94025-[hidden] → [redacted]@acme.com
Random Redaction	6011111111111117 09/30/73	→ 4222222222222226 → 11/30/85

The Oracle logo, consisting of the word "ORACLE" in white capital letters on a red horizontal bar.

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

This slide provides examples of data redaction based on the type of redaction that is configured. More detail about defining the redaction type is provided later in the lesson.

What Is a Redaction Policy?

A redaction policy specifies:

- What to redact: Specified by a schema, object (table or view), and column
- How to redact: Specify a redaction method for the column and required parameters for that method
- When to redact: Specified by a SQL expression that is evaluated for all columns in the table or view and depends on values from:
 - `SYS_CONTEXT()` for the database environment and context passed by applications
 - `XS_SYS_CONTEXT()` for Oracle Real Application Security
 - `V()` and `NV()` for Oracle Application Express
 - `dominates()` for Oracle Label Security



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

To implement Oracle Data Redaction, you define a redaction policy that specifies what should be redacted, when the redaction should take place, and how the redaction should be applied.

First, you identify a schema, table or view, and column to be redacted. Second, you specify the redaction method to use for the column, along with any required parameters. Only one column can be set when you first create the policy; however, you can later modify the policy to add additional columns. Last, you specify the exact conditions that must be met for the redaction to be enforced. For example, you can choose to redact the columns by default and show actual data only for certain users or IP addresses. You can join several of these conditions together by using logical operators. This “when” condition is set once for the table or view, so it applies to all columns that are added to the policy.

Note that it is a good practice to create a white list when defining security policies. A white list enables you to deny by default and add only authorized exceptions to the policy. In the redaction SQL expression, you can insert negative comparisons to build your white list. You can specify that redaction should take place for all users, IP addresses, and so forth that do not match values written into your expression. Black lists, which are the inverse of white lists, also are supported.

Managing Redaction Policies

- You use the procedures in the DBMS_REDACT package to manage redaction policies:
 - ADD_POLICY: Add a redaction policy to a table.
 - DROP_POLICY: Remove a redaction policy from a table.
 - ALTER_POLICY: Change a redaction policy.
 - ENABLE_POLICY: Enable a redaction policy after it is disabled.
 - DISABLE_POLICY: Disable a redaction policy.
- EXECUTE privilege on DBMS_REDACT is required to execute the procedures.
- Enterprise Manager Cloud Control 12c supports Oracle Data Redaction.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You use the DBMS_REDACT PL/SQL package and its procedures to define and manage redaction policies. You can also use Oracle Enterprise Manager Cloud Control 12c to define and manage redaction policies.

To use the procedures in the DBMS_REDACT package, you must be granted the EXECUTE privilege for the package. You do not need any privileges on the tables or views that you specify in the redaction policy.

Best Practice: Restrict the EXECUTE privilege on the DBMS_REDACT package to those users who need to perform data redaction to ensure that policies are not defined or altered inappropriately.

Defining a Redaction Policy

The redaction policy dictates:

- What to redact, as specified by:
 - *Schema name* (OBJECT_SCHEMA)
 - *Object name* (OBJECT_NAME)
 - *Column name* (COLUMN_NAME)
- When to redact, as specified in a *policy expression* (EXPRESSION)
- How to redact, as specified by:
 - *Function type* (FUNCTION_TYPE)
 - *Function parameters* (FUNCTION_PARAMETERS) or *regular expression parameters* (REGEXP_*)



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You define the redaction policy by using the DBMS_REDACT.ADD_POLICY procedure and providing the appropriate parameters, as listed in the slide.

If you do not provide a value for FUNCTION_TYPE, full redaction is used for the specified column.

Restrictions:

- You cannot redact SYS, nor SYSTEM schema objects.
- Both redacting virtual columns and redacting base table columns that are used in virtual column expressions are not supported.
- You cannot redact certain less commonly used column data types.
- Oracle Virtual Private Database (VPD) and Oracle Label Security (OLS) can be used in conjunction with Data Redaction. To avoid circular logic, the VPD and OLS predicates should not directly reference redacted columns.
- You cannot redact Edited Views.

Adding a Redaction Policy to a Table or View

Defining a redaction policy for the `COMMISSION_PCT` column in the `HR.EMPLOYEES` table:

```
SQL> exec DBMS_REDACT.ADD_POLICY
      ( policy_name    => 'EMPSAL_POLICY', -
        object_schema => 'HR', -
        object_name   => 'EMPLOYEES', -
        column_name   => 'COMMISSION_PCT', -
        expression     => -
'SYS_CONTEXT(''USERENV'', 'CLIENT_IDENTIFIER') != ''HR''', -
        function_type  => DBMS_REDACT.FULL)
```



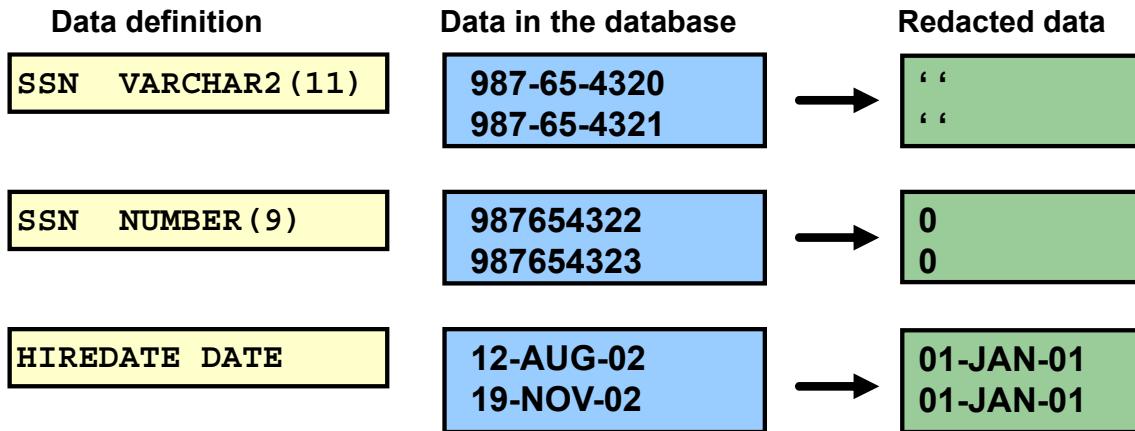
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In this example, a redaction policy named `EMPSAL_POLICY` is defined on the `COMMISSION_PCT` column of the `HR.EMPLOYEES` table. This policy specifies that full redaction is used for the `COMMISSION_PCT` column values that are returned to all users whose identifier is not `HR`.

The `REDACTION_POLICIES` and `REDACTION_COLUMNS` views provide the list of redaction policies and redacted columns, respectively.

The `REDACTION_VALUES_FOR_TYPE_FULL` view shows all of the current values for full redaction.

Full Redaction: Examples



Change the default value of full redaction:

```
SQL> exec DBMS_REDACT.UPDATE_FULL_REDACTION_VALUES
      (varchar_val    => 'N');
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

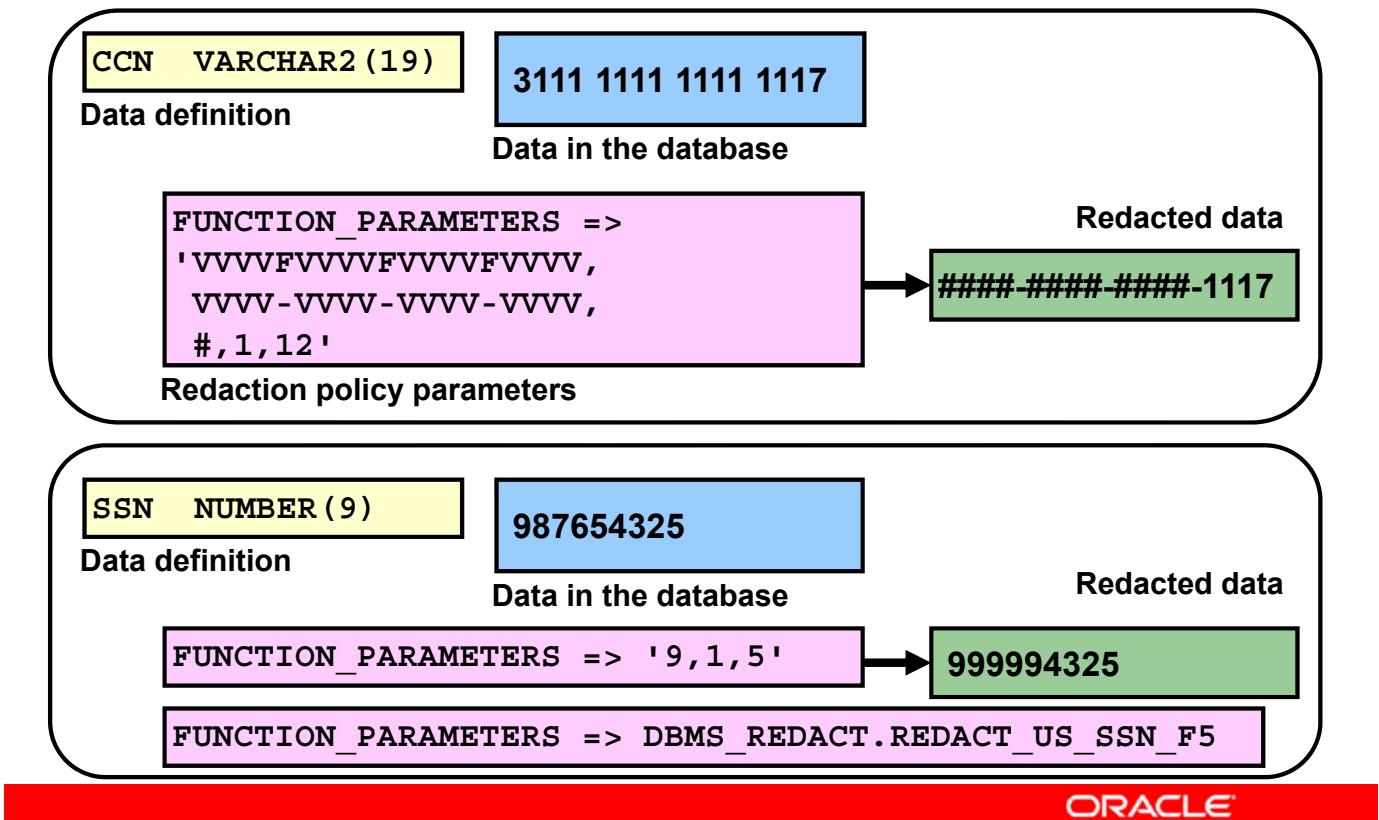
You use full redaction to redact the returned data to a fixed value. As shown in the examples:

- Character data is redacted to a single, blank space.
- Number data is redacted to a single zero.
- Datetime data is redacted to January 1, 2001.

You specify full redaction by setting FUNCTION_TYPE => DBMS_REDACT.FULL. Full redaction is the default if you do not specify a redaction type in the redaction policy.

Use the UPDATE_FULL_REDACTION_VALUES procedure to modify the default displayed values for full redaction. 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.

Partial Redaction: Examples



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You specify partial redaction by setting `FUNCTION_TYPE => DBMS_REDACT.PARTIAL`. In addition, you specify parameters so that a portion of the data is redacted and part of the original data is preserved.

The graphic in the slide shows the use of an input parameter that specifies the format characters (F) and values (V), which may be redacted. The output format parameter specifies where to insert formatting characters. The other parameters indicate the redaction character that should be used and the values that should be redacted.

For a number data type, you can specify only numbers 0 through 9 as redaction characters.

For a date data type, lower case letters indicate month, day, year, hour, minute, and second. The value following each letter specifies the redaction value. An uppercase letter indicates that no redaction should be performed for that element.

In the CCN example, the first 12 positions of the VARCHAR2 column are changed to “#” and blanks are replaced with “-”, and the remaining four digits are left as they are.

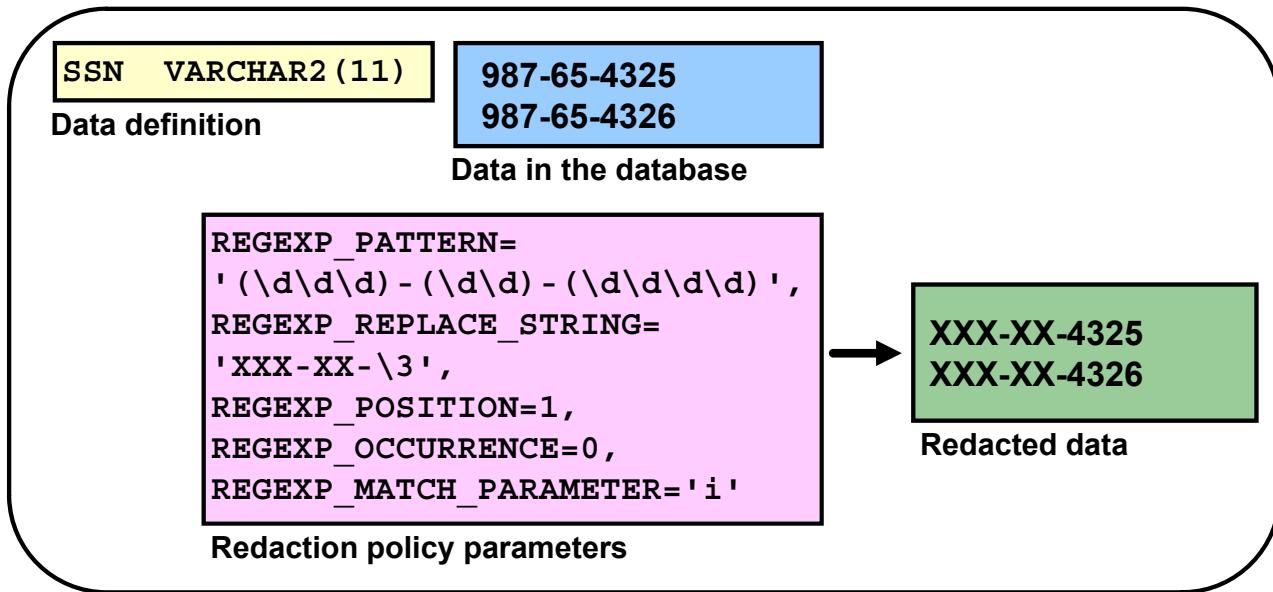
In the SSN numeric example, the first five positions get numeral 9.

There are partial fixed-character redaction shortcuts that you can use for commonly redacted Social Security numbers and postal codes.

Examples of redaction shortcuts:

- DBMS_REDACT.REDACT_US_SSN_F5: Redacts the first five 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 four 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 five 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 four 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 XXXXXXXXXX.
- DBMS_REDACT.REDACT_ZIP_CODE: Redacts a five-digit postal code when the column is a VARCHAR2 data type. For example, 95476 becomes XXXXX.
- DBMS_REDACT.REDACT_NUM_ZIP_CODE: Redacts a five-digit postal code when the column is a NUMBER data type. For example, 95476 becomes XXXXX.
- DBMS_REDACT.REDACT_DATE_MILLENNIUM: Redacts dates that are in the DD-MON-YY format to 01-JAN-00 (January 1, 2000).
- DBMS_REDACT.REDACT_DATE_EPOCH: Redacts all dates to 01-JAN-70.
- DBMS_REDACT.REDACT_CCN16_F12: Redacts a 16-digit credit card number, leaving the last four digits displayed. For example, 5105105105105100 becomes XXXXXXXXXXXX5100.
- 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_EMAIL_NAME: Redacts the email name as specified by setting the regexp_pattern parameter with the RE_PATTERN_EMAIL_ADDRESS shortcut. The redaction replaces the email name with four x characters. For example, the email address psmith@example.com could be replaced with xxxx@example.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 shortcut. For example, the IP address 192.0.2.254 could be replaced with 192.0.2.999, which is an invalid IP address.

Regular Expression



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You specify regular expression redaction by setting FUNCTION_TYPE => DBMS_REDACT.REGEXP.

Use the following parameters to define regular expression redaction:

- Use REGEXP_PATTERN to specify a regular expression that represents the column data that will be redacted and that describes the pattern of data that must be matched. In the example in the slide, the \d indicates one digit, but not a “d” letter. The whole pattern consists of three digits, then a “-”, then two digits, then a “-”, then four digits.
- Use REGEXP_REPLACE_STRING to define how the data should be replaced. In the example in the slide, the \3 indicates that the data in the third set of parenthesis should be preserved, but the first two sets of digits are replaced by XXX-XX.
- Use REGEXP_POSITION to indicate the starting position for the string search.
- Use REGEXP_OCCURRENCE to specify how the search and replace operation is to be performed. Use 0 to indicate that all occurrences of the match should be replaced. Use n to indicate that the nth occurrence of the match should be replaced.
- Use REGEXP_MATCH_PARAMETER to specify a text literal to change the default matching behavior of the function. The behavior of this parameter is the same as for the REGEXP_REPLACE SQL function. Refer to *Oracle Database SQL Language Reference* for detailed information.

Modifying the Redaction Policy

Use the DBMS_REDACT.ALTER_POLICY procedure to alter an existing redaction policy as follows:

- Modify the policy expression.
- Modify the type of redaction for a specified column.
- Modify the function parameters for a specified column.
- Add a column to the redaction policy.
- Remove a column from the redaction policy.

```
DBMS_REDACT.ALTER_POLICY
( policy_name    => 'EMPSAL_POLICY',
  object_schema  => 'HR',
  object_name    => 'EMPLOYEES',
  action          => DBMS_REDACT.MODIFY_EXPRESSION,
  expression      =>
  'SYS_CONTEXT(''USERENV'', ''CLIENT_IDENTIFIER'') = ''HR'''
);
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Use the DBMS_REDACT.ALTER_POLICY procedure to modify an existing redaction policy. Specify the type of modification by setting the ACTION parameter. Specify values for EXPRESSION, COLUMN_NAME, and FUNCTION_PARAMETERS or REGEXP_* depending on the value in ACTION.

Use the DBMS_REDACT.ALTER_POLICY procedure to:

- Modify the policy expression by specifying DBMS_REDACT.MODIFY_EXPRESSION for the ACTION parameter. Each policy can have only one policy expression. So when you modify the policy expression, you are replacing the existing policy expression.
- Modify the type of redaction for a specified column by specifying DBMS_REDACT.MODIFY_COLUMN for the ACTION parameter
- Modify the redaction function parameters for a specified column by specifying DBMS_REDACT.MODIFY_COLUMN for the ACTION parameter
- Add a column to the redaction policy by specifying DBMS_REDACT.ADD_COLUMN for the ACTION parameter. You must also specify the redaction type and any required parameters.
- Remove a column from the redaction policy by specifying DBMS_REDACT.DROP_COLUMN for the ACTION parameter

Exempting Users from Redaction Policies

- Conditions included in policy expressions may allow users to see actual data.
- SYS is exempt from all redaction policies.
- Grant the EXEMPT REDACTION POLICY system privilege to exempt other users from all redaction policies.
- Best Practices:
 - Use default deny (white list) conditions in policy expressions.
 - Grant the EXEMPT REDACTION POLICY privilege judiciously to ensure that the redaction policies are enforced appropriately.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The SYS user is exempt from redaction policies and is able to view actual values for data.

If other users need access to the actual values, you must grant them the EXEMPT REDACTION POLICY system privilege. This privilege exempts the users from all redaction policies.

Users granted the DBA role are also exempt from redaction policies because the DBA role contains the EXP_FULL_DATABASE role, which is granted the EXEMPT REDACTION POLICY system privilege.

Because applications may need to perform CREATE TABLE AS SELECT operations that involve redacted source columns, you can grant the application the EXEMPT REDACTION POLICY system privilege.

Defining Data Redaction Policies by Using Cloud Control 12c

The screenshot shows the Oracle Enterprise Manager Cloud Control 12c interface. The top navigation bar includes links for Enterprise, Targets, Favorites, History, Setup, Help, SYSMAN, and Log Out. The main menu on the left is set to 'database'. The central content area is titled 'Data Redaction' with a brief description: 'Data Redaction provides a way to easily protect sensitive information by presenting valid masked data to a low-privilege user or application. It masks data as close to display time as possible, making the sensitive data still useful. All data processing are performed normally, and backend referential integrity is preserved.' Below this is a search bar with fields for Schema (%), Table/View (%), and Policy Name (%), and a 'Go' button. A large yellow callout box with the text 'Click Create to create a new policy.' is overlaid on the search bar area. Below the search bar is a toolbar with buttons for Format, Create, Edit, View, Enable, Disable, Delete, and Detach. A table titled 'Data Redaction Policies' is displayed, showing columns for Schema, Table/View, Policy Name, Enabled, and Masked Columns. The table currently has no data.

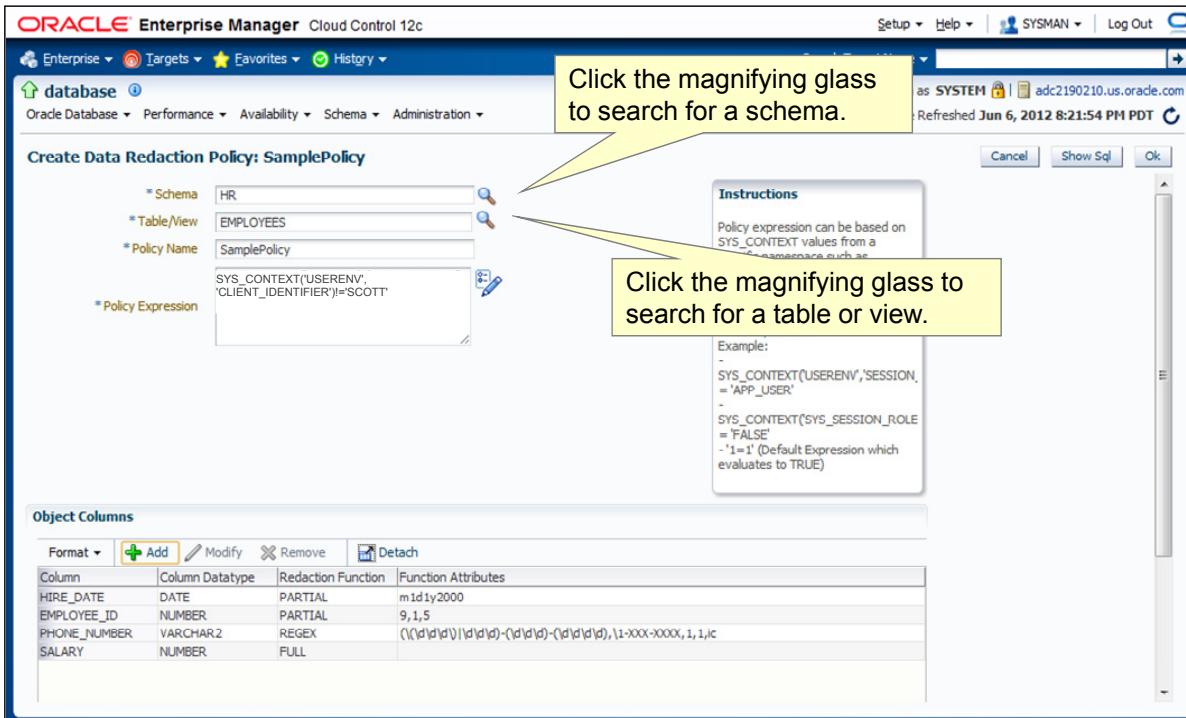
ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Enterprise Manager Cloud Control 12c provides a graphical user interface that enables you to define data redaction policies. You access the Data Redaction page using the top-level menu items.

To define a new redaction policy, click Create.

Creating a Data Redaction Policy



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

On the Create Data Redaction Policy page, specify the schema and table that the redaction policy applies to. You can click the magnifying glass next to the Schema and Table/View fields to open a search window. Provide a name for the policy and specify the policy expression. By default, policy expression is set to 1=1, which evaluates to true meaning that redaction will always take place.

After specifying the schema and table or view name, add columns to the redaction policy. The example in the slide specifies the HIRE_DATE column of the HR.EMPLOYEES table. For this column, partial redaction is indicated and the redacting format is specified for a DATE data type. Predefined redaction templates are available in Oracle Enterprise Manager to enable you to easily specify redaction for common types of sensitive data. In this example, the "Date to Millennium" redaction template is selected. This template sets the redaction format to m1d1y2000. Date values will be redacted to 01-JAN-00, the first day of the millennium. Note that you can manually change the template's default value. You can select the Custom template if you want to manually enter all values.

Add the EMPLOYEE_ID column to the redaction policy. Again, partial redaction is indicated with the required function attributes for a column of NUMBER data type.

Specify the policy expression so that redaction is in effect when the client identified is not SCOTT.

As a best practice, you should also add a condition to the policy expression to handle NULL return values. Generally, you should redact if the function in the policy expression returns a NULL. This is important when end users attempt to view the redacted data from some other application or channel besides the one that you planned for when you created the original policy expression.

Finally, create the redaction policy by clicking OK. A message shows that the redaction policy was successfully created and enabled by default.

Using Oracle Data Redaction with Other Oracle Database Security Solutions

- Serves different use cases
- Is complementary to other Oracle Database security solutions and can be used with them in tandem

	Oracle Advanced Security Transparent Data Encryption (TDE)
	Oracle Database Vault
	Oracle Database Firewall
	Oracle Audit Vault
	Oracle Label Security (OLS)
	Oracle Virtual Private Database (VPD)
	Oracle Enterprise Manager Data Masking Pack
	Oracle Real Application Security (RAS)

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Oracle Data Redaction serves different use cases than other Oracle Database security solutions and should be viewed as complementary to them. Oracle Database security offerings can be paired with Oracle Data Redaction to provide a more secure environment.

Oracle Database Security Features

Oracle Database Security Product or Feature	Description
Oracle Data Masking Pack	Physically alters and masks production data before you distribute copies of it to development and test environments
Oracle Virtual Private Database	Dynamically rewrites the SQL query to filter the data returned Can prevent certain rows from being returned, but for columns, it is limited to setting them to NULL
Oracle Advanced Security Transparent Data Encryption (TDE)	Transparently encrypts and decrypts data in storage without impacting applications Does not alter the SQL query results Requires managing the encryption keys outside of the database
Real Application Security	Powerful fine-grained features for controlling the data returned Requires uptake of the programming framework and custom coding Suitable for applications where you are able to modify the code
Oracle Database Vault	Applies an additional layer of access control to highly privileged database user accounts Disallows selected operations and returns an error code



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Oracle Data Masking Pack is an optional feature of Oracle Enterprise Manager Cloud Control 12c. By irreversibly replacing original, sensitive data with fictitious data, it enables customers to safely share data with IT developers and business partners. It provides end-to-end secure automation for provisioning test databases from production in compliance with regulations.

You use Oracle Virtual Private Database (VPD) to create security policies that control database access at the row and column level. Essentially, VPD adds a dynamic WHERE clause to a SQL statement that is issued against the table, view, or synonym to which a VPD security policy was applied.

You can use VPD to redact column data, but only to change columns to NULL, which may cause errors in applications.

Oracle Data Redaction and VPD are complementary features that you can use together. For example, you can use VPD to filter the rows that are returned by the query, and you can use Oracle Data Redaction to redact the columns that are displayed.

Transparent Data Encryption (TDE) is a feature of Oracle Advanced Security. You use TDE to encrypt sensitive data that is stored in data files. It uses a TDE master encryption key that is stored in a keystore.

For detailed information on TDE, refer to *Oracle Database Advanced Security Guide 12c Release 1*. This topic is also covered in another lesson.

Oracle Data Redaction and TDE are complementary features. They can be used together to protect the data that is stored in the database and to mask the data when it is returned to a query. Both TDE and Data Redaction are part of the Oracle Advanced Security license.

Oracle Database Real Application Security is a new feature of Oracle Database 12c. It is a database authorization model that supports declarative security policies and enables end-to-end security for multitier applications. It provides an integrated solution for securing the database and application user communities. Also, it advances the security architecture of Oracle Database to meet existing and emerging demands of applications that are developed for the Internet.

Best Practices: Preventing Unauthorized Policy Modifications and Exemptions

- Limit EXECUTE privilege on the DBMS_REDACT package.
- Limit EXEMPT REDACTION POLICY privilege.
- Limit SELECT on REDACTION_POLICIES and REDACTION_COLUMNS.
- If you use Oracle Database Vault to restrict privileged user access, you can create realms to restrict granting of the EXEMPT REDACTION POLICY privilege.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

To prevent unauthorized policy modifications and exemptions, limit grants of the EXECUTE privilege on the DBMS_REDACT package and granting of the EXEMPT REDACTION POLICY privilege. Also, limit SELECT privilege on the Data Redaction views (that is, REDACTION_POLICIES and REDACTION_COLUMNS).

Oracle Database Vault restricts access to certain areas of the database even for highly privileged users. If you use Database Vault, you can use realms to restrict granting of the EXEMPT REDACTION POLICY privilege.

For additional information on Oracle Database Vault, refer to the *Implementing Oracle Database Vault* course and *Oracle Database Vault Administrator's Guide 12c*.

Best Practices: Considerations

- Choose the columns to redact selectively; redact only what is needed.
- Keep the policy expression logic as simple as possible.
- When defining regular expression policies, keep the regular expressions simple.
- Partial and full redaction policies generally have higher performance than regular expression policies, which must be compiled each time they are used.
- You cannot perform CTAS operations where any of the selected columns are being redacted.
- You can apply only one policy on a table or view.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

To maximize performance, redact only the columns that are needed and keep the logic in policy expressions and regular expressions as simple as possible. Note that per-row processing time (CPU time) increases as more columns are redacted in a given table or view. Also, the evaluation of policy expressions occurs once for each query execution, so applications that repeat queries to fetch one row of data at a time (rather than fetching multiple rows at once) may detect overhead due to redaction.

Summary

In this lesson, you should have learned how to:

- Describe Oracle Data Redaction
- Manage redaction policies
- View redaction policy information in the data dictionary



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Practice: Overview

This practice covers the following topics:

- 14-1: Using full data redaction
- 14-2: Using partial data redaction
- 14-3: Changing the default value for FULL redaction
- 14-4: Cleaning up redaction policies



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Error : You are not a Valid Partner use only

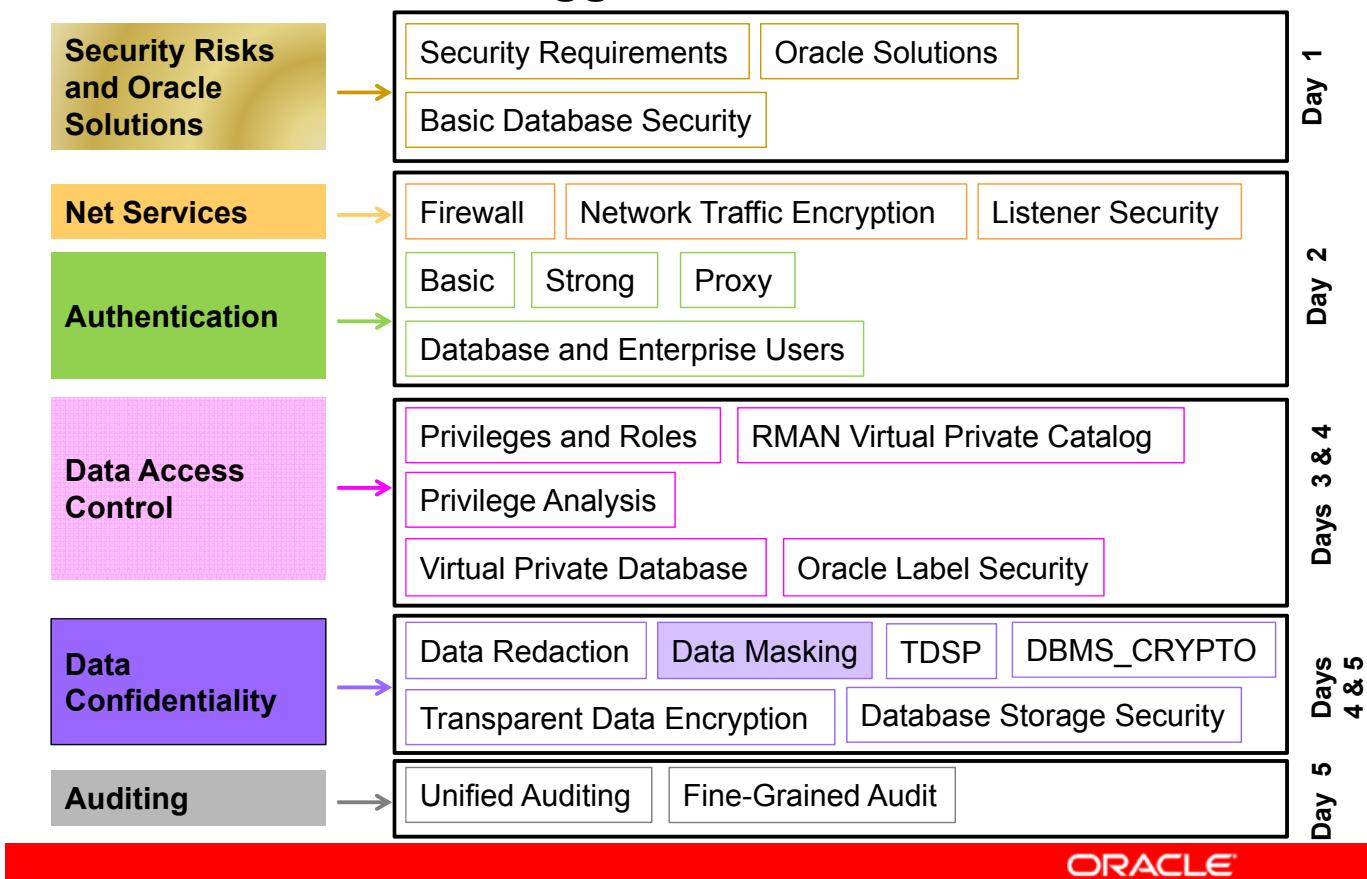
15

Application Data Model and Oracle Data Masking

ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Suggested Schedule



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this module, you should be able to:

- Explain the purpose of an Application Data Model
- Create sensitive column types
- Identify and discover sensitive columns in an ADM
- Create an ADM ready for data masking
- Explain the data masking process
- Use and create data mask formats
- Create a data masking definition
- Mask sensitive and confidential data



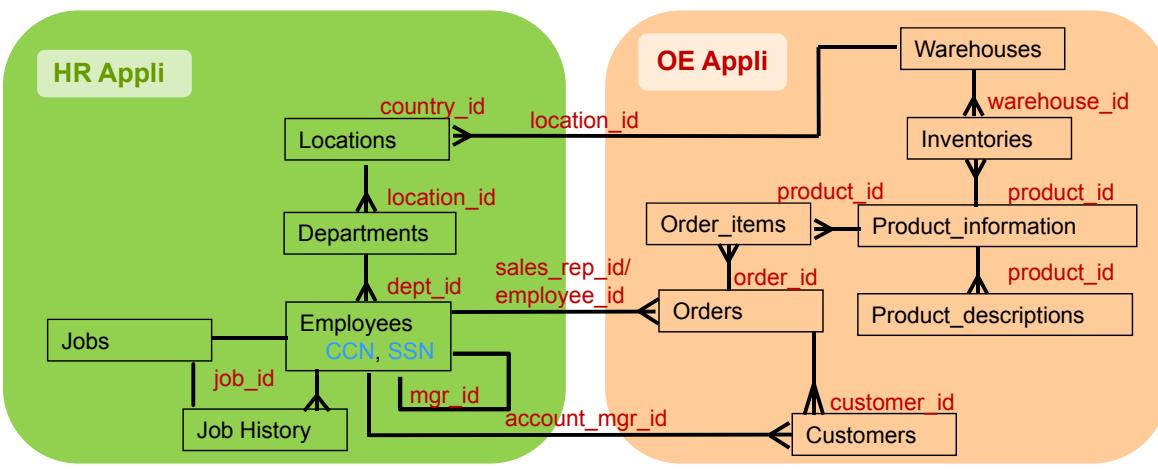
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

For a complete understanding of the Oracle Data Masking feature usage, refer to the following guide in the Oracle documentation:

- *Oracle Database Testing Guide 12c Release 1 (12.1)*

Application Data Model: Overview

- ADM scans application schemas to model relationships between tables and columns stored in EM repository.
- Data subsetting and data masking benefit from ADM to automatically retrieve data relationships.
- ADM maintains knowledge of application data privacy attributes such as credit card numbers, SSN#.



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Secure Test Data Management uses Enterprise Manager's Data Discovery and Modeling (DDM) capability to enable operations such as:

- Sensitive Data Discovery
- Data Masking
- Data Subsetting

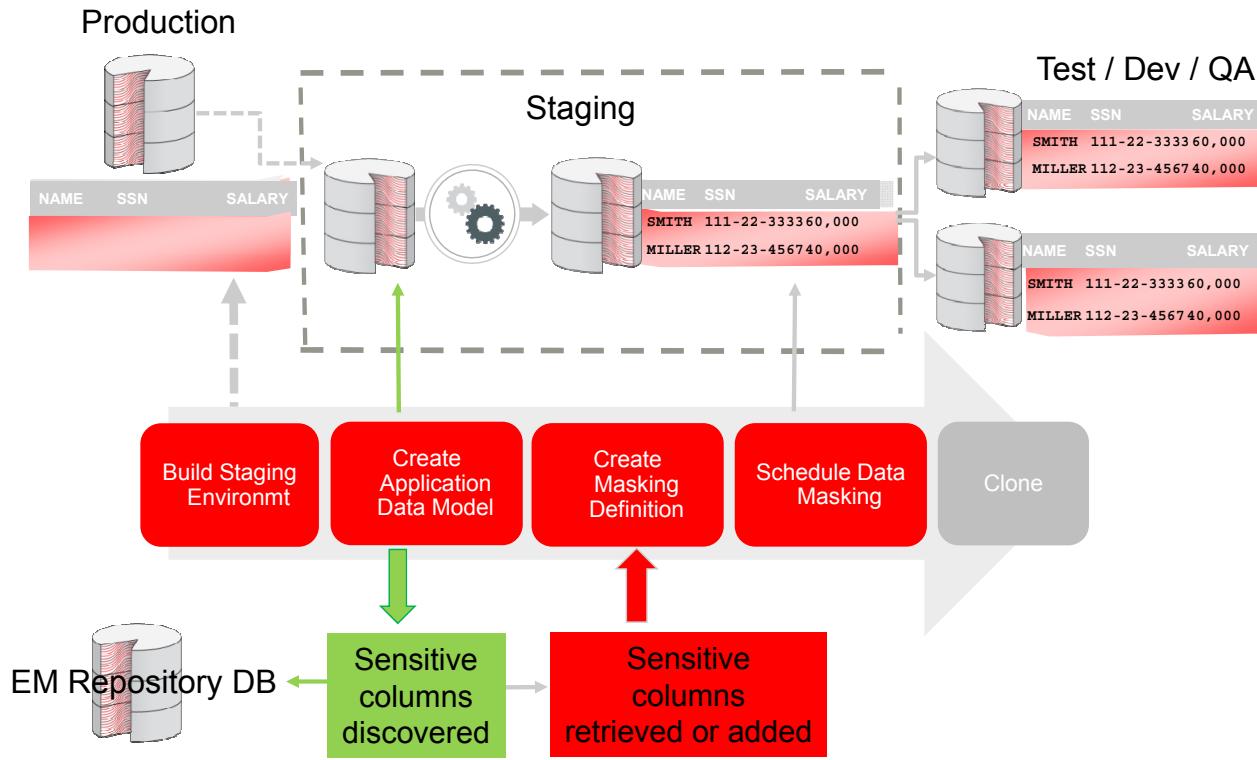
DDM enables scanning and tagging of sensitive data and modeling of data relationships that are encapsulated within an Application Data Model (ADM). The ADM stores the list of applications, tables, and relationships between table columns that are declared in the data dictionary, imported from application metadata (custom applications, Oracle eBusiness Suite, Fusion Applications), or user-specified.

The ADM maintains sensitive data types and their associated columns used by Data Masking and Data Subsetting, to produce test data securely.

In the example of the slide, the ADM contains:

- All HR, OE schema tables
- All referential relationships found between HR schema tables, OE schema tables and between both HR and OE schemas
- Identified sensitive columns such as CCN, SSN of HR.EMPLOYEES table

ADM and Data Masking Process



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

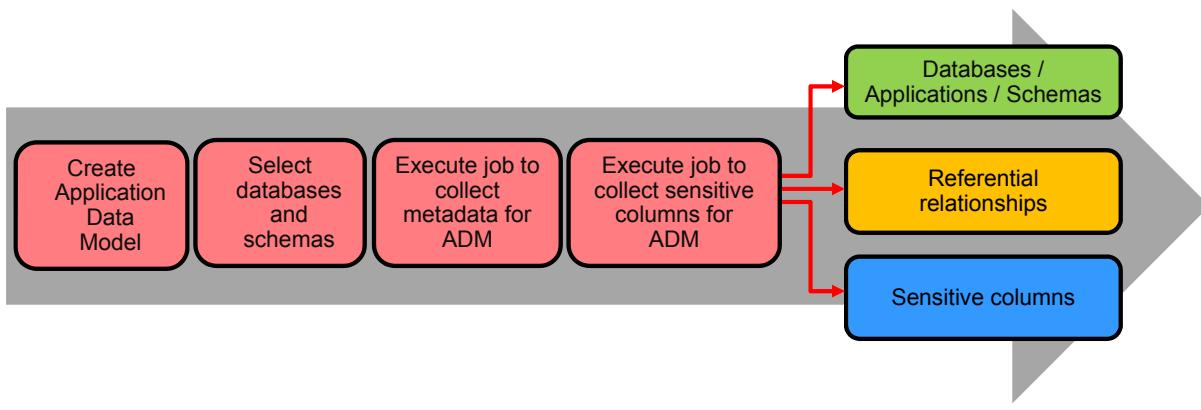
Where does ADM come into play in the process of data masking?

During installation, before masking the data, perform the following steps:

1. Build a staging environment.
2. Optionally, create sensitive column types.
3. Create an ADM.
4. Discover sensitive columns. You can use an automatic discovery job that will add the discovered columns to the ADM or manually add sensitive columns to the ADM.
5. Create the masking definition which relies on an ADM. The ADM provides the list of identified sensitive columns, ready to be masked with formats such as the new encrypt masking format to obfuscate sensitive column data.
6. Schedule data masking to run. Now the staging database is ready to be cloned in other test environments.
7. Clone the staging database to a test environment.

Creating an ADM

- ADM create job automatically:
 - Retrieves tables and columns of selected schemas
 - Builds application data relationships on specified schemas
- ADM sensitive column discovery job identifies sensitive columns in specified schemas.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The creation an ADM requires three pieces of information:

- The name of the ADM
- The source database from which to collect schemas definitions, referential constraints, and relationships between tables
- The type of retrieval. If you choose to retrieve custom application information, you can choose to either create one application for each schema that you select or add the application tables manually. Creating one application for each chosen schema is recommended because you will not have to manually define every application table and referential relationship, if any.

When you schedule the ADM job, the job automatically interrogates the source database to retrieve schemas information and fills the ADM with metadata about schema tables, referential relationships between tables, and columns from within the same schema or across schemas, storing the resulting metadata in the EM repository.

When is completed, you can edit the ADM to view three types of information retrieved:

- Applications and Tables
- Referential Relationships
- Sensitive Columns

Viewing ADM Content

The screenshot shows the Oracle Application Data Models interface with three main tabs:

- Applications and Tables**: Lists tables from the HR and OE schemas. The HR schema contains COUNTRIES, DEPARTMENTS, EMPLOYEES, JOBS, LOCATIONS, and REGIONS. The OE schema contains CUSTOMERS and ORDERS. All tables have 'Transaction' as their Table Type and 'Dictionary' as their Source.
- Referential Relationships**: Displays relationships between tables. It shows that HR.EMPLOYEES has a Parent relationship via EMPLOYEE_ID to HR.EMPLOYEES and a Dependent relationship via MANAGER_ID to HR.EMPLOYEES. OE.CUSTOMERS has a Dependent relationship via ACCOUNT_MGR_ID to HR.EMPLOYEES. OE.ORDERS has a Dependent relationship via SALES REP ID to HR.EMPLOYEES. OE.CUSTOMERS has a Parent relationship via CUSTOMER_ID to OE.ORDERS. OE.ORDERS has a Dependent relationship via CUSTOMER_ID to OE.CUSTOMERS.
- Sensitive Columns**: Shows no sensitive columns.

ORACLE

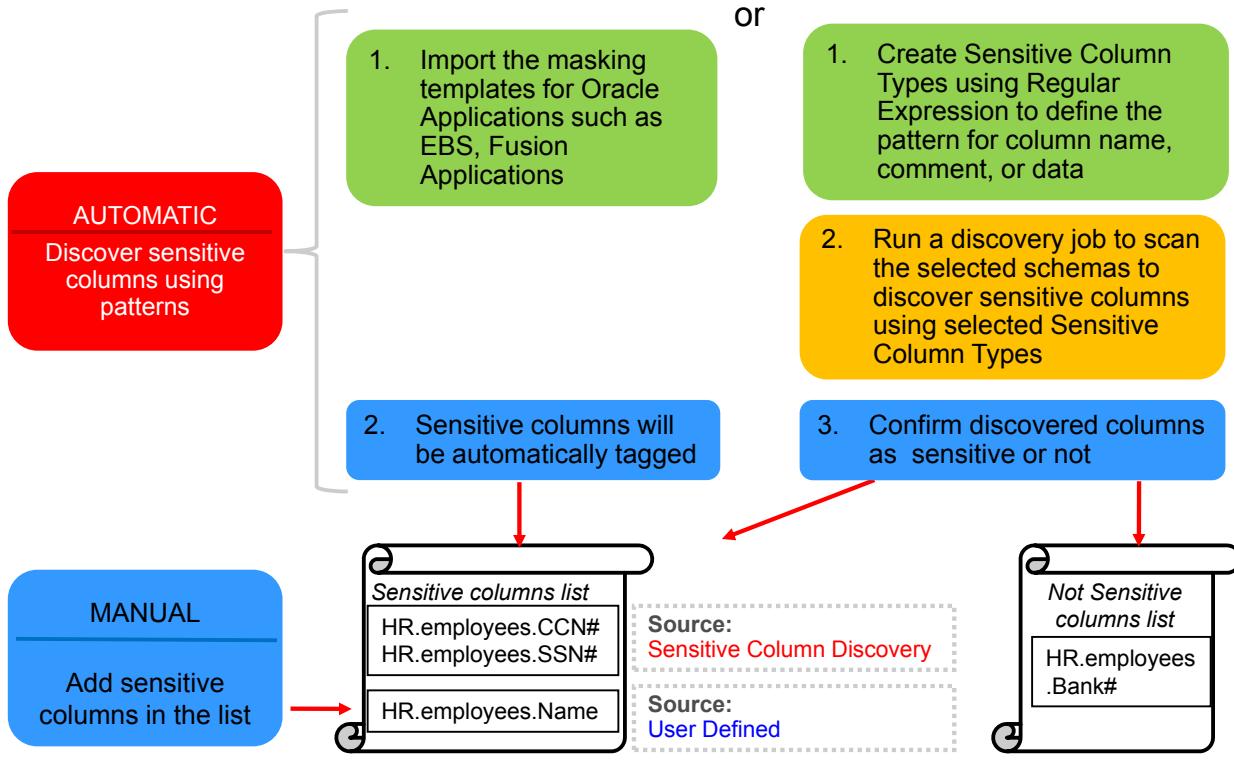
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Viewing the ADM created, the first tab “Applications and Tables” lists all tables retrieved from the schemas defined in the ADM creation step. The source is “Dictionary” because the job retrieved the information in the data dictionary of the source database. You can still manually add (or delete) schemas and tables.

The second tab “Referential Relationships” displays all relationships collected from the source database for each table referenced or referencing another table within the HR schema and also each table referenced or referencing another table within the OE schema and also each table referenced or referencing another table across both schemas such as OE.CUSTOMERS table ACCOUNT_MGR_ID column referencing HR.EMPLOYEES table EMPLOYEE_ID column. You can still manually add or remove referential relationships.

The third tab “Sensitive Columns” does not show any sensitive column. If you want sensitive columns to be discovered and stored in the ADM, you have to schedule another job, the “Sensitive Column Discovery Job.”

Discovering Sensitive Columns



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Discovering the sensitive data in the applications is important for masking data. Data masking works on top of ADM to scan the databases for columns that are known to be sensitive, for example, credit card numbers. You can define your own columns as being sensitive, and you have the ability to import from data masking templates for eBusiness Suite or Fusion Applications, and possibly other Oracle applications, going forward.

Executing the “Sensitive Column Discovery Job” identifies the sensitive columns and includes them in the ADM.

1. The first step is to create sensitive column types by using regular expressions to define the pattern for searching column name, comment, or data.
2. Then submit a discovery job that connects to the source database of the ADM , applies the search criteria on the applications selected from the ADM and identifies the columns as being sensitive upon the search pattern criteria selected for the current discovery job.
3. Next step is to edit the list of sensitive columns identified and mark all or some of them as either sensitive or not sensitive (these are eliminated from the ADM sensitive list). The source of the columns kept is “Sensitive Column Discovery”.

Another way to set this list is to manually add the names of the columns in the list. The columns to be added manually can be identified by a specific comment set on the column by the application or security administrator. The source of the columns added is “User Defined.”

A third way to establish this list is to import masking templates for Oracle.

Data Masking: Overview



What is data masking?

- Replacing sensitive information with realistic data based on masking rules

Why use data masking?

- It provides the ability to share data with non-production users such as testing companies.

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	PHONE_NUMBER
100	King	90	515.123.4567
105	Austin	60	590.423.4569



EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	PHONE_NUMBER
468	Jefferies	90	510.555.1256
975	Smith	60	650.555.9753

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

A number of regulations mandate that confidential, sensitive, and personally identifiable data of a company must be protected and access to this data must be restricted.

There is often a need to provide production data, or realistic looking data to in-house developers and testing organizations during application development.

Data masking is a way to meet these two conflicting needs. Data masking is the act of *anonymizing* customer, financial, or confidential data of a company to create new, legible data, which retains the original data's properties, such as width, type, and format.

In the example in the slide, three columns of the HR. EMPLOYEES table have been masked so that the data can be provided for testing or development without compromising the security of the information.

Using the Data Masking Pack

Implement data masking in an Oracle database by using the Data Masking Pack:

- Separately licensed Oracle Enterprise Manager Management Pack
- Available with Enterprise Manager Cloud Control

The Data Masking Pack includes:

- Data Discovery and Modeling
- Data Masking



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The Data Masking Pack is a separately licensed Oracle Enterprise Manager management pack available with Oracle Enterprise Manager Cloud Control.

This pack includes:

- **Data Discovery and Modeling (DDM):** Offers the ability to discover and design ADMs that serve as a knowledge base for creating Data Masking definitions. DDM features enable you to export and import the ADMs, and verify and associate definitions with multiple targets.
- **Data Masking:** Offers the ability to mask regulated or confidential data on test and development systems

Data Masking Pack: Features

Mask

- Replaces the sensitive data based on masking rules
 - Create or use or export/import masking definitions:
 - Specify which tables and columns are to be masked
 - Specify the formats used to mask columns
 - Mask Real Application Testing Workloads such as workload capture files and SQL Tuning Sets
 - Define and execute pre and post masking scripts
- Ensures that the original data cannot be retrieved, recovered, or restored
- Enables you to define a central definition for common data masking formats to be used with all databases



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

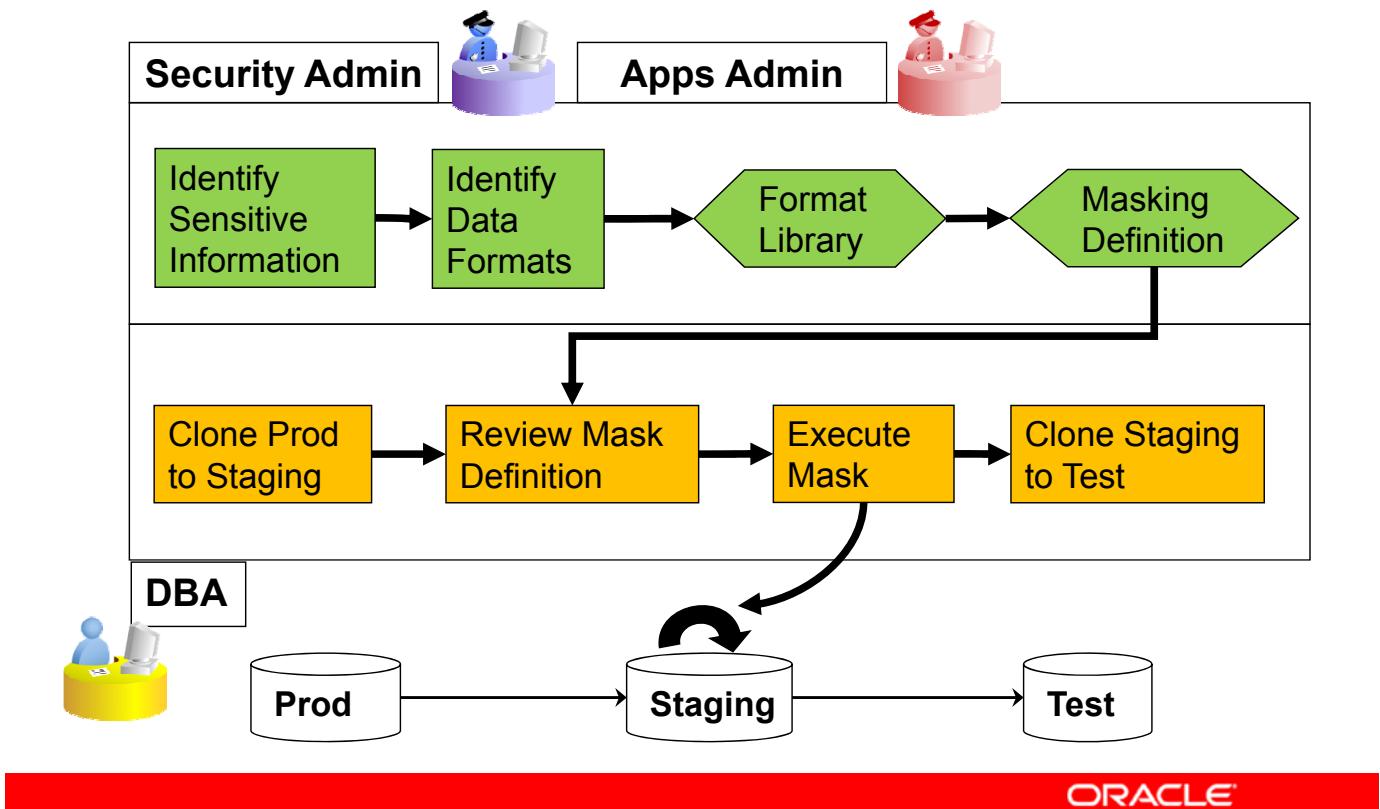
The Data Masking Pack uses an irreversible process to replace sensitive data with realistic-looking, scrubbed data based on masking rules and ensures that the original data in a production database cannot be retrieved, recovered, or restored.

You can create or use masking definitions:

- Specify which tables and columns are to be masked as well as the formats used to mask columns
- Mask Real Application Testing workloads such as workload capture files and SQL Tuning Sets:
 - STS bind data (used with SPA)
 - Workload Capture files (used with DB Replay)
 - AWR sensitive bind data is purged
- Define and execute pre and post masking scripts

Defining format libraries for reuse of standard formats across targets, and exporting or importing applications masking templates from one database to another allow data privacy rules application consistently to all production data and ensure compliance with regulations.

Implementing Data Masking



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In this example, the production database is first cloned to a staging database and then masked for use in a testing environment.

The security administrator performs the following tasks:

1. Review the application database and identify sensitive data.
2. Define mask formats for sensitive data.
3. Create a masking definition to associate table columns to the defined mask formats.

The database administrator performs the following tasks:

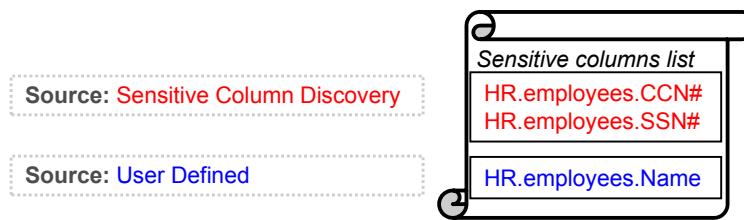
4. Clone the production database to a staging database.
5. Create a masking definition if this task is not performed by the application database administrator or security administrator and review the mask definition.
6. Execute the masking job in the staging database.
7. Verify that the masked data meets the information security requirements.
8. Refine the masking definition as necessary. If the masking definition is changed, the database administrator performs the following tasks:
 - i. Restore the altered tables.
 - ii. Reapply the masking definition until the optimal masking definitions are identified.

9. Export the masking definition for future use.
10. Clone the staging database to a test database.

Identifying Sensitive Data for Masking

The **Security Admin**  and **Apps Admin**  identify data that needs to be masked to ensure regulation compliance.

- Using ADM discovered sensitive columns



- Selecting which columns to mask



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The security or application administrator would typically direct the process by identifying what types of information must be masked to comply with various regulations. The sensitive columns that were automatically discovered or manually added in the ADM automatically appear. The administrator can select specific columns in the list that should be masked.

Note: When you want to delete an ADM, you need to delete dependent objects such as Data Masking definitions relying on the ADM.

Creating or Using Masking Formats

Data masking choices:

- Use Oracle-supplied mask formats
- Create a specific masking format using built-in data masking primitives and routines
- Apply a mask:
 - On a set of related columns masked together
 - Based on conditions
 - Based on a SQL-expression: `SUBSTR(%ORIG_VALUE%, 1, 3) || '-111'`
- Create a function for more complicated masking.

Address	City	State	Zip	Phone
---------	------	-------	-----	-------

A format library can be exported (as an XML file) to be reused or shared with another EM installation



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The Data Masking Pack includes a format library containing:

- Predefined formats for various types of sensitive data
- Built-in masking primitives that can be used directly to mask column data

You can also use the predefined formats and or built-in masking routines to directly mask the column data.

The built-in masking primitives and routines can also be used to build a more sophisticated masking format. If the built-in masking primitives and routines do not satisfy your data masking requirements, you can create a PL/SQL function to use for masking.

The format library can be saved to an XML file so that it can be reused or shared with another installation of Enterprise Manager.

Using Oracle-Supplied Mask Formats and Built-in Masking Routines

Format Library

Format Library						
The Format Library contains a collection of ready-to-use masking formats which can be used when creating a masking definition.						
Search Format		Go		Export Import Create		
View Create Like Edit Delete						
Select	Format	Data Type	Sensitive Column Type	Sample	Description	Owner
<input checked="" type="radio"/> American Express Credit Card Number	Character	CREDIT_CARD_NUMBER	3780908154440865	~10 billion unique American Express credit card numbers	SYSMAN	
<input type="radio"/> Discover Card Credit Card Number	Character	CREDIT_CARD_NUMBER	6011508687980355	~10 billion unique Discover Card credit card numbers	SYSMAN	

Routines

Routines	Definition
ArrayList	List of values that will be selected randomly
Fixed Number / String	Number or literal string that will be used
Random Date / Digit / Number	Range of date / digit / number that will be used randomly
Random String	Literals in the specified range
Shuffle	Shuffling of original data
Table Column	The specified column that is used randomly

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The Data Masking Pack format library includes predefined formats for a number of types of sensitive data, such as credit card numbers, national identifiers, and phone numbers. The mask formats are built on mask routines. An example of a predefined mask format is the American Express Credit Card Number built on a random number routine and a post-processing pre-defined function ensuring that the resulting data is realistic. The post-processing uses either predefined functions of DBSNMP.DM_FMTLIB package installed on the repository database or your own functions.

If you intend to use the formats using predefined functions, verify that the DM_FMTLIB package is installed on your target database. Copy the following scripts to your target database and execute them in SQL*Plus as a user who has privileges to create the package in the DBSNMP schema:

```
$PLUGIN_HOME/sql/db/latest/masking/dm_fmtlib_pkgdef.sql
$PLUGIN_HOME/sql/db/latest/masking/dm_fmtlib_pkgsbody.sql
```

where PLUGIN_HOME can be any of the locations returned by the following statement executed in the EM repository:

```
select PLUGIN_HOME from sysman.gc_current_deployed_plugin
where plugin_id='oracle.sysman.db' and destination_type='OMS';
```

The Data Masking Pack includes built-in masking routines for various types of data.

You can create your own mask formats by using built-in masking routines:

- Shuffling: This routine is useful when the range of values in a column is not known and you determine that the shuffling of values in the same table provides a sufficient degree of privacy protection.
- Substitute: Uses a hash-based substitution for the original value and always yields the same mask value for any given input value (deterministic masking)
- Substring: Substring is similar to the database `substr` function with a specified start position and length.
- Table Column: A table column enables you to select values from the chosen column as the replacement value or part thereof. The data type and uniqueness must be compatible. Otherwise, a failure occurs when the job runs. This format is combinable.

Refer to *Oracle Database Testing Guide 12c Release 1 (12.1)* for additional information and examples of format entry routines.

Example: Data Masking of the EMPLOYEES Table

The diagram illustrates the masking process for the EMPLOYEES table. The top table shows the original data, and the bottom table shows the data after masking. Arrows point from four boxes below the top table to specific columns in the bottom table:

- EMPLOYEE_ID: Random Number
- LAST_NAME: Table Column
- DEPARTMENT_ID: Dept Format
- PHONE_NUMBER: USA Phone Number

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	PHONE_NUMBER
100	King	90	5151234567
105	Austin	60	5904234569
110	Chen	100	5151244269

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	PHONE_NUMBER
468	Jefferies	1090	5105551256
975	Smith	1060	6505559753
396	Allen	10100	9255553597

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

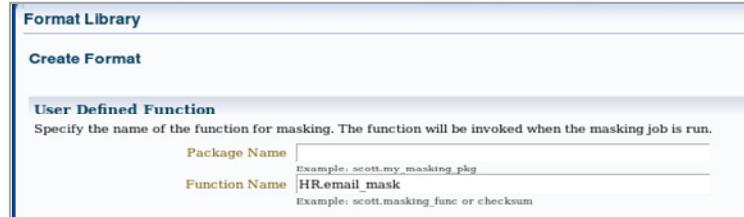
The slide shows several examples of masking data.

- Masking of the EMPLOYEE_ID column is accomplished by using one of the built-in masking routines.
- The mask for the LAST_NAME column is constructed by referencing another column in the database.
- Masking the DEPARTMENT_ID column is accomplished by using a new mask created by the administrator. The new mask is based on several of the built-in masking routines.
- The PHONE_NUMBER column is masked by using the USA Phone Number mask from the Oracle-supplied format library.

Creating a Masking Format Using a User-Defined Function

```
CREATE OR REPLACE FUNCTION hr.email_mask
  (orig_value VARCHAR2) RETURN VARCHAR2
IS   emailadd varchar2(100);
BEGIN
  SELECT first_name || '.' || employee_id || '.' ||
         last_name || '@not_realco.com' INTO emailadd
    FROM hr.employees
   WHERE email = orig_value;
  RETURN emailadd;
END;
```

- Specify User Defined Function as the format entry type.
- Specify the previously created function.



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

A PL/SQL function of your own design can be used to mask data. The function in the slide is used to generate a mask for the email address.

This code is executed on the target database.

Notice that this function does not hide the user's real name, but uses the actual first and last names, and the actual employee ID, in the masked email address.

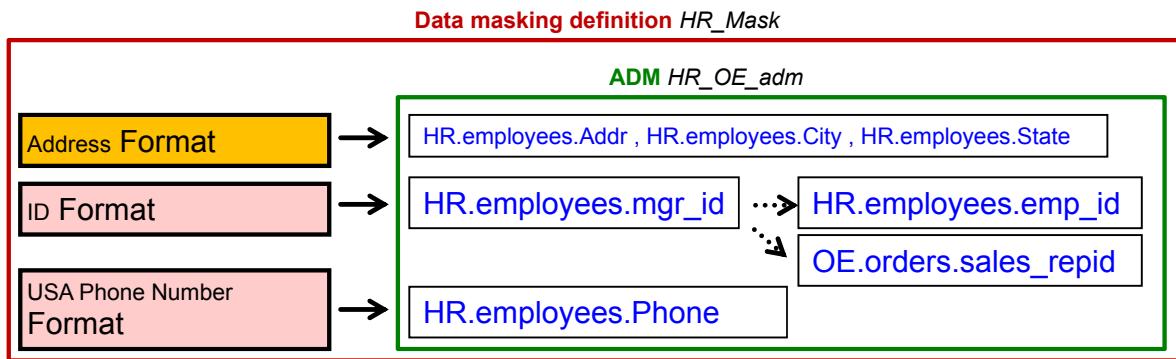
The masking functions must be reviewed, and results checked, to ensure that the functions actually provide the level of protection required.

If a reversible function is required, it can be implemented as a user-defined function. You must assume that if an obfuscation or encryption algorithm is reversible, it will be reversed. That is, if the original data can be extracted from the masked or encrypted data, someone will extract the original data.

After creating the PL/SQL function that you intend to use for masking, you can specify it in a data mask format.

Creating Data Masking Definitions

- A data masking definition associates a data masking format with database columns:



- Foreign key relationships are automatically identified and columns are masked.
- Application-enforced constraints can be defined as related columns for automatic masking.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

A data masking definition associates a data masking format with a specified column in the database.

If you select a column with referential integrity constraints defined on it, the Data Masking Pack automatically applies the same masking rule to the associated tables and columns.

If the referential integrity of the data is maintained in the application instead of the database, you can add the associated tables and columns via the Dependent Column capability. This ensures that the data remains consistent whether the referential integrity of the application is maintained in the database or in the application.

Importing Formats and Modifying Properties

Import Format

Select	Format	Data Type	Sensitive Column Type	Sample	Description	Owner
<input type="radio"/>	Auto Mask Format	Character	UNDEFINED	1	Masking by scrambling the characters and numbers	SYSMAN
<input checked="" type="radio"/>	French_Name_format	Character	UNDEFINED	Marie-Hélène		SYSMAN

Logged In As: SYS
Column: FIRST_NAME
Data Type: VARCHAR2(20)

Cancel **Import**

The format is imported from the format library:

Import Format

Database: orcl
Owner: HR
Table: EMPLOYEES

Logged In As: SYS
Column: FIRST_NAME
Data Type: VARCHAR2(20)

Define Column Mask

Owner: HR
Column: FIRST_NAME

Table: EMPLOYEES
Data Type: VARCHAR2(20)

By default all records in the table will be masked using the specified format. You can optionally identify more than one subset of records using conditions. Each subset can be masked using a corresponding masking format. The subsets will be masked in the order they are specified. A subset will not be masked again even when it matches a subsequent condition.

Format Entry Properties

Format Entry

Format Entry Properties

Select	Condition	Property	Value	Property	Value	Sample	Remove
<input checked="" type="radio"/> Default Condition		Fixed String	Marie-			Marie-Louise	
		List of Values	Anne,Cécile,Françoise,Hél				

Add Condition

Import Format **Format Entry** **ArrayList** **Add**

Cancel **OK**

ORACLE

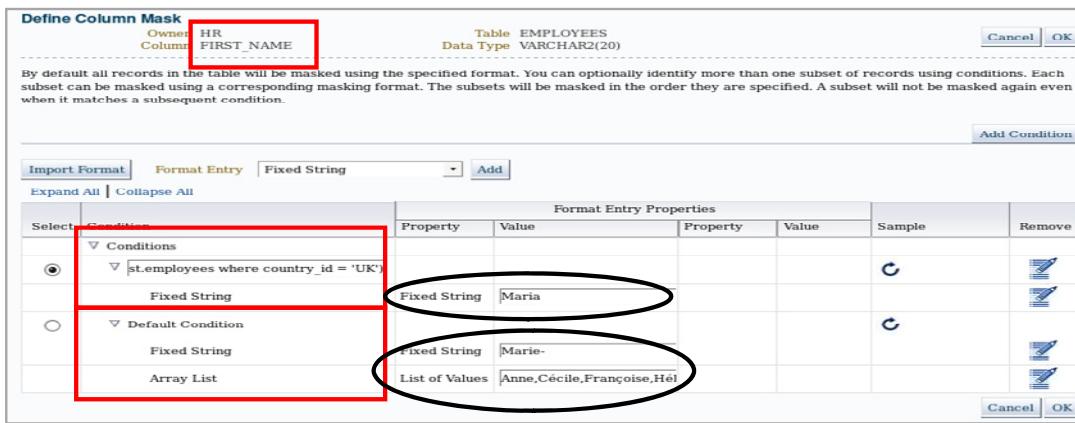
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can associate a masking format for a column by using a previously defined data masking format.

The data masking format is imported from the library and associated with the specified column. You can modify the format entry properties as necessary.

Using Condition-Based Masking

- Assign multiple mask formats to the data in a column based on conditions you specify through a SQL query.
- Separate SQL queries identify the rows for each specified mask format.
- Specify a default masking format for data in rows that are not selected based on the defined conditions.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In some cases, you may want to apply different masking routines to a column based on specific conditions. The Data Masking Pack supports condition-based masking so that you may assign multiple mask formats to the data in a column based on conditions you specify through a SQL query.

For each masking format, you specify a separate SQL query. You must also define a default masking format that applies to any rows not selected via the conditional SQL queries.

Using Compound Masking

Mask related columns so that the entire set of mask columns remains consistent and valid.

The screenshot shows two steps in the Oracle Data Masking Pack:

Step 1: A list of columns from the 'CUSTOMERS' table. The 'CUST_FIRST_NAME' and 'CUST_LAST_NAME' columns are selected (indicated by checkboxes) and highlighted with a red box. Step 1 also highlights the 'Mask selected columns as a group' checkbox at the top left of the list.

Select	Owner	Table Name	Column Name	Sensitive Column Type	Data Type	Comment
<input type="checkbox"/>	OE	CUSTOMERS	ACCOUNT_MGR_ID	UNDEFINED	NUMBER(6)	References hr.employees.employee_id.
<input type="checkbox"/>	OE	CUSTOMERS	CREDIT_LIMIT	UNDEFINED	NUMBER(9,2)	Check constraint.
<input type="checkbox"/>	OE	CUSTOMERS	CUSTOMER_ID	UNDEFINED	NUMBER(6)	Primary key column.
<input type="checkbox"/>	OE	CUSTOMERS	CUST_EMAIL	UNDEFINED	VARCHAR2(40)	
<input checked="" type="checkbox"/>	OE	CUSTOMERS	CUST_FIRST_NAME	UNDEFINED	VARCHAR2(20)	NOT NULL constraint.
<input checked="" type="checkbox"/>	OE	CUSTOMERS	CUST_LAST_NAME	UNDEFINED	VARCHAR2(20)	NOT NULL constraint.

Step 2: A 'Define Group Mask' dialog box. It shows the 'Format Type' as 'Shuffle' and the 'Grouping Columns' as 'CUST_FIRST_NAME, CUST_LAST_NAME'. The 'CUST_FIRST_NAME' and 'CUST_LAST_NAME' columns are listed with their data types (VARCHAR2(20)). The 'Preserve Original Data' checkboxes are unchecked for both columns, and the 'Remove' checkboxes are checked. Red arrows point from the 'CUST_FIRST_NAME' and 'CUST_LAST_NAME' labels in the grouping columns section to their respective rows in the table below.

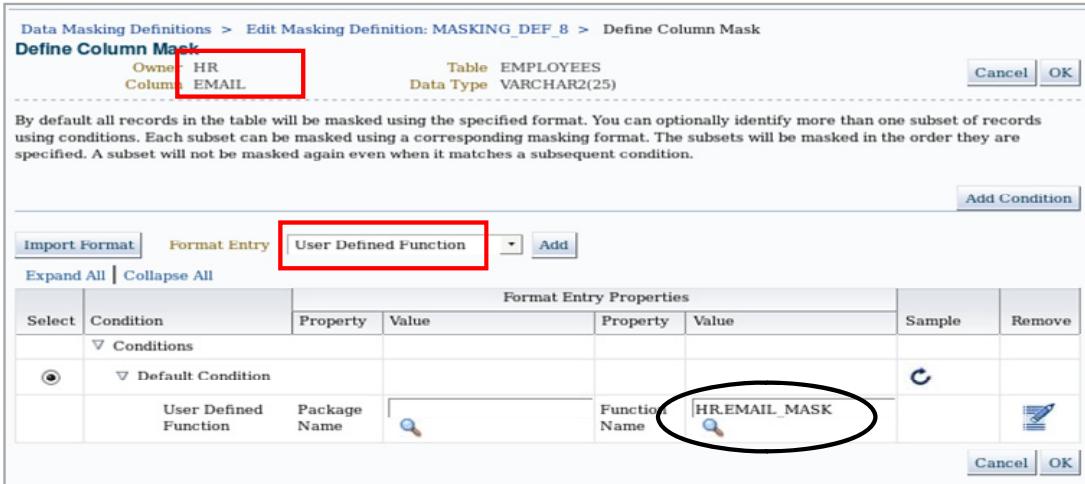
ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In many cases, the columns in database tables contain data that is related to other columns in that same table. When one or more of these columns contain sensitive data that needs to be masked, these intercolumn dependencies need to be maintained so that the masked data as a whole is consistent with the original data. For example, if the table has address information consisting of a street address, city, state, and ZIP code, the masked copy of the data must be valid so that the city, state, and ZIP code are consistent.

By using the compound masking capability of the Data Masking Pack, you can group the related data elements together and mask the group with another set of related data elements.

Using a User-Defined Masking Function and Post-Processing Masking Function



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Using a User-Defined Function

You can specify a user-defined function for the format entry. After selecting “User Defined Function” from the Format Entry menu, specify the name of the package (optional) or function in the Format Entry Properties Value field.

Creating a Post-Processing Function

You can create a PL/SQL function such as a checksum function which returns a value. The function would be executed after the data is masked.

The value returned is the masked data which can be concatenated to the checksummed data. This could be combined with other format functions to provide another piece of a mask definition.

Implementing a Post-Processing Function

A predefined PL/SQL function can be specified as a post-processing function. This function will execute against the data after it is masked. This function is created in the Enterprise Manager repository.

Generating the Data Masking Script

The screenshot shows the Oracle Database Data Masking Definitions interface. At the top, there's a search bar labeled 'Search' and a dropdown menu set to 'Masking Definition'. To the right are buttons for 'Go', 'Import', 'Import from Software Library', and 'Export from'. Below this is a toolbar with buttons for 'View', 'Edit', 'Generate Script' (which is highlighted with a red box), 'Schedule Job', 'Delete', 'Actions', 'Clone Database', and 'Go'. The main area is a table with columns: Select, Masking Definition, Application Data Model, Description, Columns, Status, Most Recent Job Ended, and SQL Performance Analyzer Task. A row is selected for 'MASKING_DEF_B' with 'HR_OE_ADMIN' in the Application Data Model column. The status column shows '2 Script Not Generated'.

- Validation checks are performed:
 - Ensures that uniqueness can be maintained
 - Ensures that data mask formats match column data types
 - Checks space availability
 - Warns about check constraints
 - Checks for the presence of default partitions
- PL/SQL-based masking script is generated
- Review and schedule masking

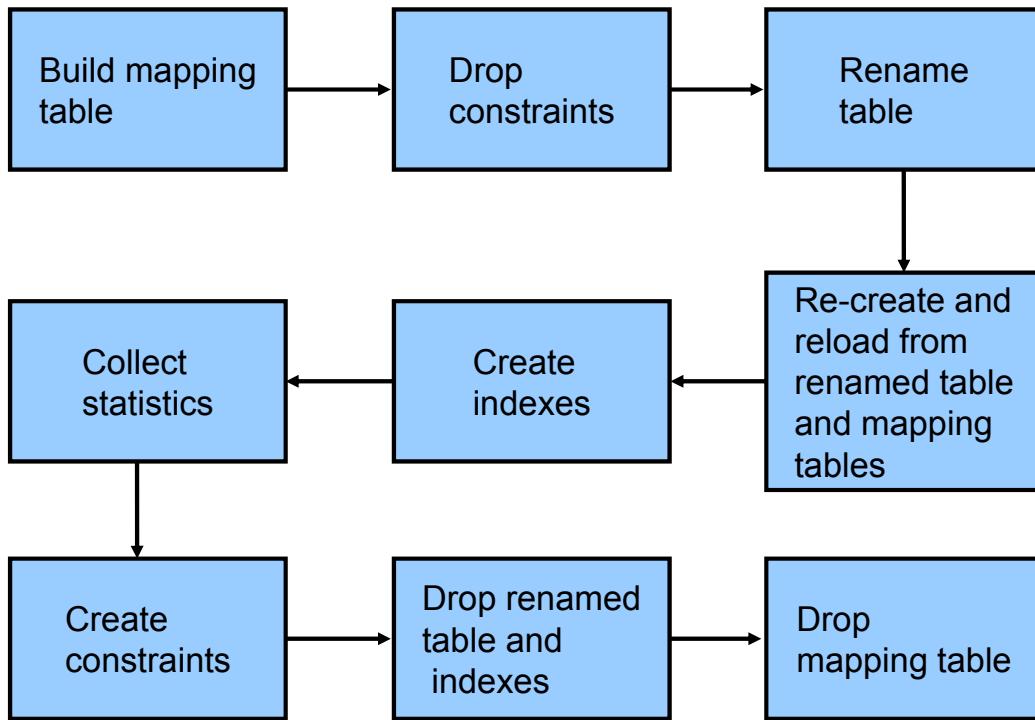
ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

After creating the data masking definitions, perform the following steps:

1. Generate the masking script.
 - i. When you generate the script, the Data Masking Pack first performs a series of validation checks to ensure that the data masking process will complete successfully.
 - ii. After the validation checks have successfully completed, a PL/SQL-based script is generated.
2. The impact report alerts you to many possible error conditions that may occur before running the data masking definition script. In particular, it checks the space available to the various user schemas that are involved in the masking process.
3. The script generation step produces a script that you can review and save.
4. After the data masking script has been generated, you can immediately schedule the data masking job. You can also schedule it at a later date. The original table, the new table plus mapping tables, and indexes on both original and new tables exist at the same time during the masking process.

Understanding the Data Masking Process



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The data masking job performs bulk operations to rapidly replace the table containing sensitive data with an identical table containing masked data while retaining the original database constraints, referential integrity and associated access structures, such as indexes, partitions, and access permissions, such as grants. The script takes advantage of the built-in optimizations in the database. It disables database logging and runs in parallel to quickly create a masked replacement for the original table. The original table containing sensitive data is dropped from the database completely and is no longer accessible.

Steps in the masking process:

1. Build a mapping table for each column to be masked. The mapping table contains (original_value, mask_value).
2. Drop constraints and revoke grants.
3. Rename the table.
4. Create a new table using mapping tables joined to the original table.
5. Create indexes on the new table.
6. Gather statistics.
7. Replace constraints and grants.
8. Drop indexes on the original table and drop the original table with the purge option.
9. Drop mapping tables.

Creating an Application Masking Template

An Application Masking Template:

- Is an XML file of the data masking definition
- Serves as a backup of the masking definition
- Allows data masking definitions to be shared with other Enterprise Manager environments using another repository

Data Masking Definitions
Data masking is the process of making sensitive information in test or non-production databases safe. It discloses data of a similar type. A masking definition defines the columns to be masked and the format of masked data during a masking operation. The Format Library contains a collection of ready-to-use masking formats.

Select	Masking Definition	Application Data Model	Description	Columns	Status
<input checked="" type="radio"/>	MASKING_DEF_8	HR_OE_ADMIN			2 Script Generated

- Serves to import a data masking definition

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Creating an Application Masking Template

After you have completed the masking definition, you can save the definition in a portable XML format called an Application Masking Template. This enables you to restore the masking definition if needed and to share the masking definition with another Enterprise Manager installation.

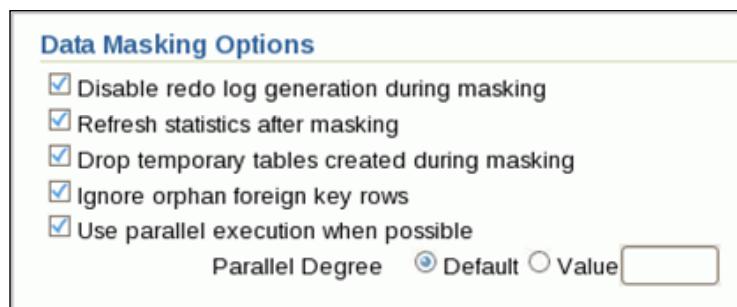
The Application Masking Template is created by using the Export Masking Definition feature.

Importing Data Masking Definitions

You can import a previously exported masking definition that is stored in an XML file (an Application Masking Template) into the Enterprise Manager repository. This enables you to use the masking definition for new masking definitions.

Controlling Data Masking Operations

- Disable redo log generation during masking
- Refresh statistics after masking
- Drop temporary (mapping) tables created during masking
- Ignore orphan foreign key rows
- Customize degree of parallelism



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

When you create a data masking definition you can modify the following options:

- **Disable redo log generation during masking:** By default, redo logging and flashback logging are disabled during masking. If you only want to test your data masking definition, you may choose to enable redo log generation and use flashback database to restore the database to its pre-masking state.
- **Refresh statistics after masking:** Disable this option if you want to use special options when collecting statistics, such as histograms or different sampling percentages.
- **Drop temporary (mapping) tables created during masking:** Temporary tables that map the original sensitive data values to mask values are created during the masking process. You may want to keep these tables until the business users have validate the masked data. Because the mapping tables show how masking changed the tables, the mapping tables must be dropped before the database is available for unprivileged users.
- **Ignore orphan foreign key rows.**
- **Customize the degree of parallelism:** You can customize the degree of parallelism for the masking process.

Summary

In this module, you should have learned to:

- Use ADM
- Create sensitive column types
- Identify and discover sensitive columns in an ADM
- Create an ADM ready for data masking
- Explain the data masking process
- Use and create data mask formats
- Create a data masking definition
- Mask sensitive and confidential data



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Practice: Overview

This practice covers the following topics:

- 15-1: Creating an ADM
- 15-2: Creating sensitive column types
- 15-3: Discovering sensitive columns
- 15-4: Creating a data masking definition and mask formats
- 15-5: Applying a masking definition



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

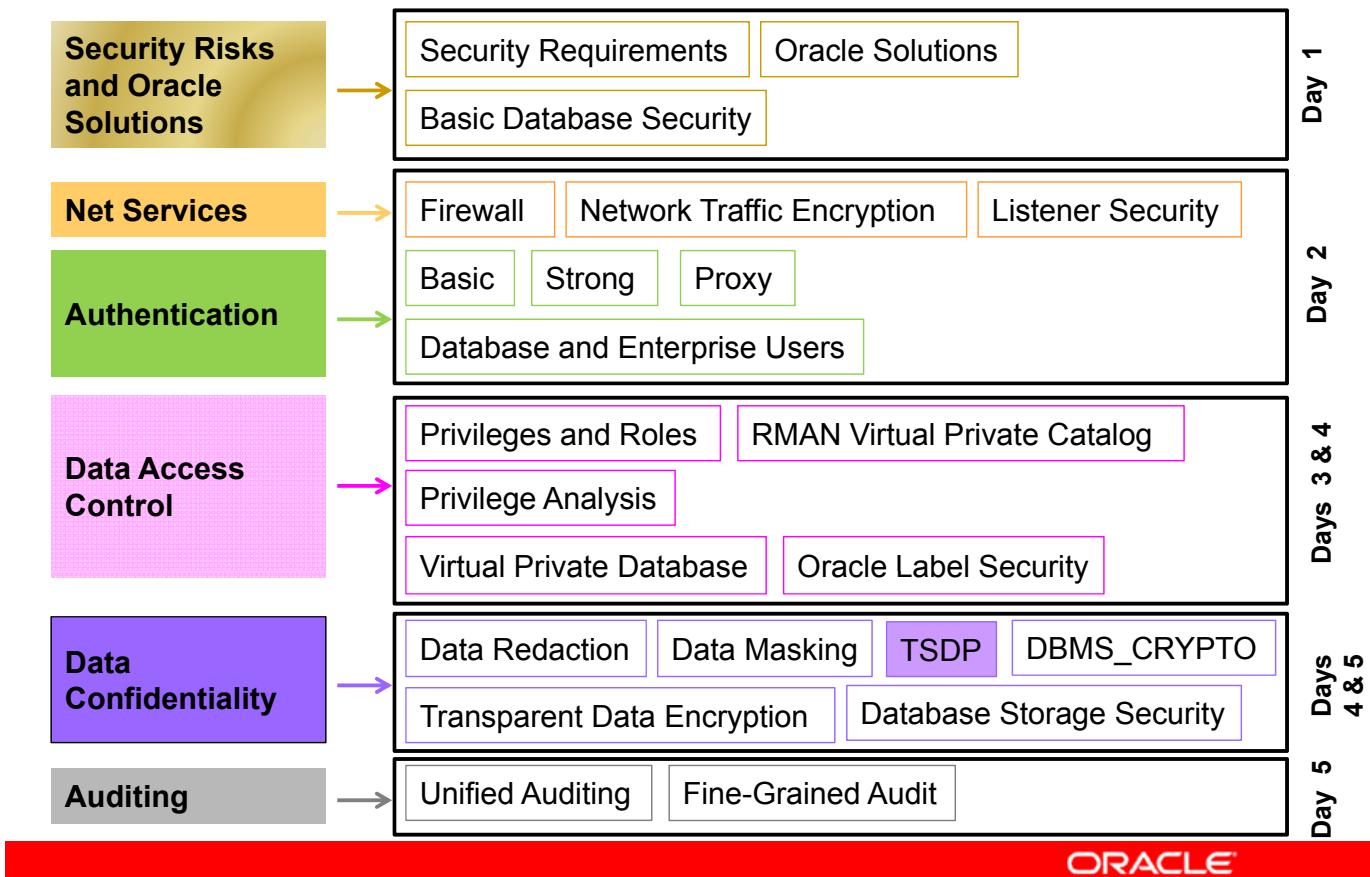
16

Implementing Transparent Sensitive Data Protection

ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Suggested Schedule



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Describe the benefits of Transparent Sensitive Data Protection (TSDP)
- Configure a TSDP policy for VPD
- Configure a TSDP policy for Data Redaction
- Configure a TSDP in CDB and PDBs
- Export and import a TSDP policy
- Use the AUDIT_REDACT policy



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Benefits of TSDP

- Configure the sensitive data protection once, and then deploy this protection
 - Specify the target data when necessary
 - Add new sensitive columns matching the sensitive type
 - Export / import the TSDP policies into another database
- Manage protection of multiple sensitive columns based on:
 - An ADM source
 - A sensitive type
 - A specific schema, table, or column
- Protect the sensitive columns discovered by DB or ADM with:
 - VPD
 - Data Redaction

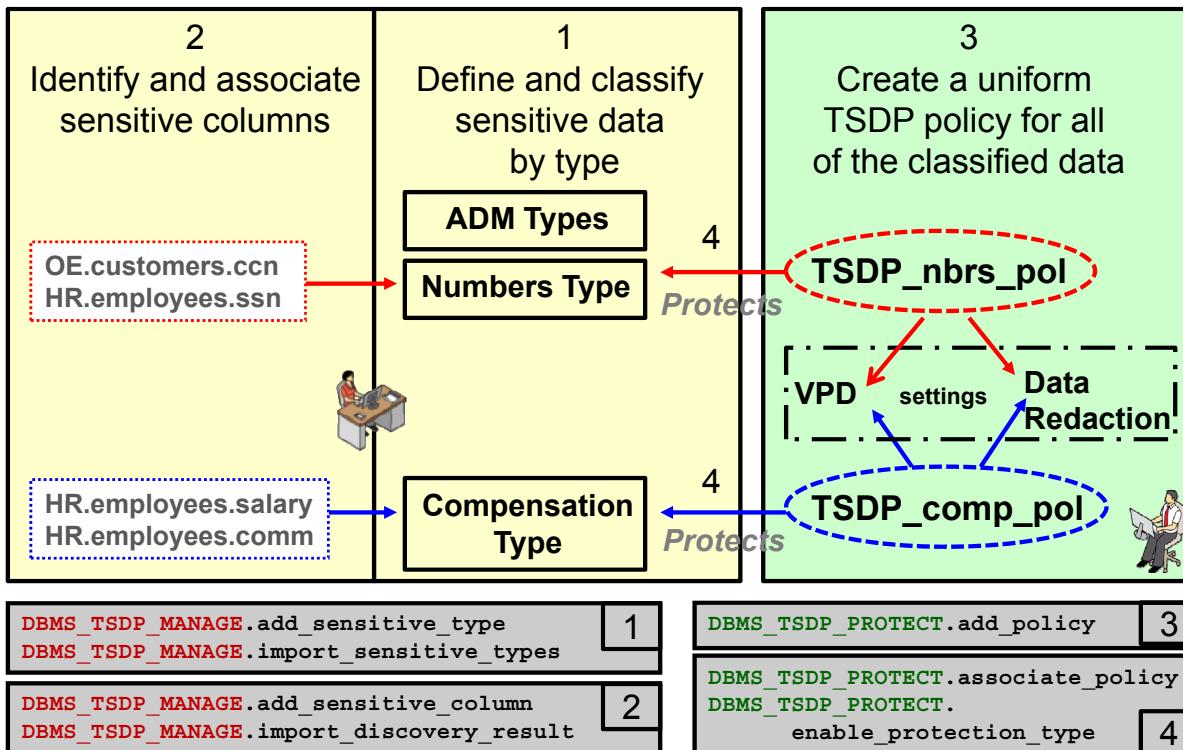


Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

TSDP automatically protects sensitive information as per data security compliance guidelines in databases across an enterprise.

- You can configure TSDP policies to designate how a class of data (for example, credit card numbers) must be protected without actually having to specify the target data. You do not need to include references to the actual target columns that you want to protect when you create the TSDP policy. The TSDP policy finds these target columns based on a list of sensitive columns in the database and the policy's associations with the specified sensitive types. **This can be useful when you add more sensitive data to your databases after you have created the TSDP policies.**
Deploy the TSDP policy by exporting or importing the policy to or from another database, performing a full export or import of the database that contains the policy.
- You can enable or disable protection for multiple sensitive columns based on a suitable attribute (such as the source database of the identification, the sensitive type itself, or a specific schema, table, or column). This granularity provides a high level of control over data security. You can configure data security based on a specific category rather than for each individual column.
- Data Discovery and Modeling discovers sensitive columns in databases and stores these columns, categorized by type, in ADMs. Using EM Cloud Control, use export to TSDP Catalog to copy sensitive columns and sensitive column types to TSDP-enabled databases, and in your databases, import the sensitive types and columns from the catalog.

TSDP: Overview



Transparent sensitive data protection (TSDP) enables:

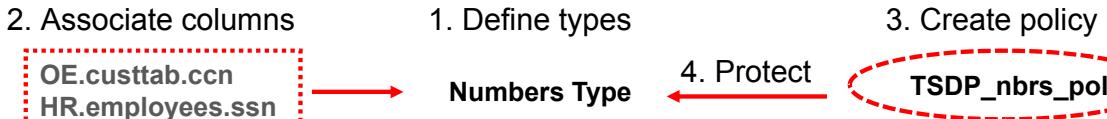
- The application administrator to use the DBMS_TSDP_MANAGE procedures to:
 - Define and classify sensitive data by creating sensitive types or by retrieving the list of discovered sensitive column types in the Enterprise Manager Cloud Control Application Data Modeling (ADM). The SOURCE_TYPE column in the DBA_SENSITIVE_COLUMN_TYPES view displays either DB or ADM. If you manually create the types, use the DBMS_TSDP_MANAGE.ADD_SENSITIVE_TYPE procedure. If you use an ADM to retrieve the existing types, use the DBMS_TSDP_MANAGE.IMPORT_SENSITIVE_TYPES procedure import the list of types.
 - Find all table columns in a database that hold sensitive data (such as credit card or Social Security numbers) and associate the sensitive columns to the sensitive types. If you manually identify the columns, use the DBMS_TSDP_MANAGE.ADD_SENSITIVE_COLUMN procedure. If you use an ADM, use the DBMS_TSDP_MANAGE.IMPORT_DISCOVERY_RESULT procedure import the list of sensitive columns. The table columns association to a TSDP sensitive type can be completed before or after the policy creation.

- The security administrator to use the DBMS_TSDP_PROTECT procedures to :
 3. Create a policy that protects this data as a whole for a given class. Use the DBMS_TSDP_PROTECT.ADD_POLICY procedure. The TSDP policy protects the sensitive data in the table columns by using either Oracle Data Redaction or Oracle Virtual Private Database settings. You can create a uniform TSDP policy for all of the data that you classify, and then modify this policy as necessary, as compliance regulations change.
 4. Associate sensitive table types to TSDP policies. Use the ASSOCIATE_POLICY procedure. And finally enable the TSDP policy protections at different levels by using the following procedures:
 - ENABLE_PROTECTION_SOURCE to protect all sensitive columns of an ADM
 - ENABLE_PROTECTION_TYPE to protect all sensitive columns of a sensitive type
 - ENABLE_PROTECTION_COLUMN to protect all sensitive columns of a specific schema and or tables and or columns

Enabling the TSDP policy either creates VPD policies or Data Redaction policies on the protected objects and columns associated with sensitive types.

DBMS_RLS and DBMS_REDACT packages are used to create the VPD or Data Redaction policies. Therefore grant the security administrator the EXECUTE privilege on the packages.

Using a TSDP Policy with VPD



3.1 Configure settings for VPD

`vpd_feature_options:`

- VPD function **mandatory**: `vpdfunc`
- `function_schema`: SEC
- `sec_relevant_cols_opt`: `DBMS_RLS.ALL_ROWS`
- `statement_types`: `SELECT, INSERT ...`
- `policy_type`: DYNAMIC
- `update_check`: TRUE

```

DBMS_TSDP_PROTECT.ADD_POLICY
('TSDP_nbrs_pol',
 DBMS_TSDP_PROTECT.VPD,
 vpd_feature_options,
 policy_conditions)
    
```

3

3.2 Conditions which should be satisfied before the policy can be enabled

`policy_conditions`

- `DBMS_TSDP_PROTECT.DATATYPE`: NUMBER

VPD policy automatically generated

```

dbms_rls.add_policy(
    object_schema => 'oe',
    object_name   => 'custtab',
    policy_name  => 'ORA$VPD_xxx',
    function_schema => 'sec',
    policy_function => 'vpdfunc',
    statement_types => 'select',
    sec_relevant_cols => 'ccn'
    sec_relevant_cols_opt =>
        DBMS_RLS.ALL_ROWS)
    
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

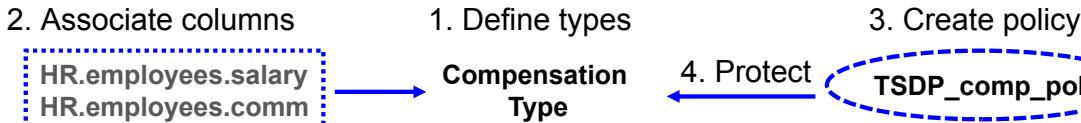
You can incorporate Oracle VPD protection with TSDP policies. This requires that a VPD function is created before you configure the settings for the VPD in the TSDP creation.

- You create a TSDP policy with the necessary VPD settings similar to the VPD policy function. The TSDP policy uses `vpd_feature_options` settings from the `DBMS_RLS.ADD_POLICY` procedure to provide VPD protection. For example, you can set the `sec_relevant_cols_opt`, `statement_types` to values that will be used in the VPD policy creation when later the TSDP policy is associated and enabled with sensitive types. You can also set policy conditions to test when the policy is enabled. For example, in the example of the slide, the data type of the protected columns should be NUMBER.
- You associate the TSDP policy with the necessary sensitive types by using the `DBMS_TSDP_PROTECT.ASSOCIATE_POLICY` procedure.
- You enable the TSDP policy by using any of the `DBMS_TSDP_PROTECT.ENABLE_PROTECTION_SOURCE`, `ENABLE_PROTECTION_TYPE`, `ENABLE_PROTECTION_COLUMN` procedures. This automatically generates a VPD policy using the settings defined in the `vpd_feature_options` argument. Notice that the `sec_relevant_cols` parameter (of `DBMS_RLS.ADD_POLICY`) is not set in the TSDP policy: it is set to the name of the sensitive columns on which TSDP enables the VPD policy. Enabling protection enforces the VPD policies on all columns identified as NUMBERS type.

When users query the tables, the output for the columns is based on both the VPD protections and the TSDP policy that are in place. Disabling the TSDP policy automatically drops the VPD policy for the type or column. Re-enabling the TSDP policy recreates the VPD policy.

In the example on this slide, the TSDP policy is enabled on the `OE.CUSTTAB.CCN` and `HR.EMPLOYEES.SSN` columns if these columns are of the `NUMBER` data type and have a length of 14. Even though the `OE.CUSTTAB.CCN` and `HR.EMPLOYEES.SSN` columns are associated with the TSDP policy, the policy is not enabled if any of these columns fail to satisfy the conditions (data type and length).

Using a TSDP Policy with Data Redaction



3.1 Configure settings for Data Redaction

`redact_feature_options:`

- `redact_type: DBMS_REDACT.PARTIAL`
- `expression: '1=1'`
- `function_parameters: '0,1,6'`

```

DBMS_TSDP_PROTECT.ADD_POLICY(
    'TSDP_comp_pol',
    DBMS_TSDP_PROTECT.REDACT,
    redact_feature_options,
    policy_conditions)

```

3

3.2 Conditions which should be satisfied before the policy can be enabled

`policy_conditions`

- `DBMS_TSDP_PROTECT.DATATYPE: NUMBER`
- `DBMS_TSDP_PROTECT.LENGTH: 14`

Data redaction policy generated

```

DBMS_REDACT.ADD_POLICY(
    object_schema => 'HR',
    object_name   => 'EMPLOYEES',
    column_name   => 'SALARY',
    policy_name=>'ORA$REDACT_xxx',
    function_type =>
        DBMS_REDACT.PARTIAL,
    expression     => '1=1',
    function_parameters => '0,1,6')

```

4.1 Associate sensitive types to policy

4.2 Enable policy

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

You can incorporate Oracle Data Redaction protection with TSDP policies

- You create a TSDP policy with the necessary redaction settings similar to the data redaction policy function. The TSDP policy uses `redact_feature_options` settings from the `DBMS_REDACT.ADD_POLICY` procedure to provide data redaction protection. For example, you can set the `function_type`, `function_parameters` to values that will be used in the redaction policy creation when later the TSDP policy is associated and enabled with sensitive types. You can also set policy conditions to test when the policy is enabled. For example, in the example in this slide, the data type of the protected columns should be `NUMBER` with a length of 14.
- You associate the TSDP policy with the necessary sensitive types by using the `DBMS_TSDP_PROTECT.ASSOCIATE_POLICY` procedure.
- You enable the TSDP policy by using any of the `DBMS_TSDP_PROTECT.ENABLE_PROTECTION_SOURCE`, `ENABLE_PROTECTION_TYPE`, `ENABLE_PROTECTION_COLUMN` procedures. This automatically generates a redaction policy using the settings defined in the `redact_feature_options` argument. Notice that the `column_name` parameter (of `DBMS_REDACT.ADD_POLICY`) is not set in the TSDP policy; it is set to the name of the sensitive columns on which TSDP enables the data redaction policy. Enabling protection enforces the data redaction policies on all columns identified as `COMPENSATION` type.

When users query the tables, the output for the columns is based on both the data redaction protections **and** the TSDP policy that are in place. Disabling the TSDP policy automatically drops the data redaction policy for the type or column. Reenabling the TSDP policy recreates the data redaction policy.

In the example of the slide, the TSDP policy is enabled on the `HR.EMPLOYEES.SALARY` and `HR.EMPLOYEES.COMM` columns if these columns are of the `NUMBER` data type and have a length of 14. Even though the `HR.EMPLOYEES.SALARY` and `HR.EMPLOYEES.COMM` columns are associated with the TSDP policy, the policy is not enabled if any of these columns fail to satisfy the conditions (data type and length).

Using the Predefined REDACT_AUDIT TSDP Policy

Use the predefined REDACT_AUDIT TSDP policy to mask bind values with an ‘*’:

- Bind variables and sensitive columns in the expressions of conditions

```
SELECT count(*) FROM hr.emp WHERE sal IN (:VAR1, :VAR2); 
```

```
SELECT emp_id FROM hr.emp  
WHERE sal > (SELECT sal FROM hr.emp WHERE mgr_id = :VAR1); 
```

- A bind variable and a sensitive column appearing in the same SELECT item

```
SELECT (sal*:VAR1), (comm*:VAR2), (emp_id+:VAR3)  
FROM payroll.account; 
```

- Bind variables in expressions assigned to sensitive columns in INSERT or UPDATE operations

```
UPDATE payroll.account SET acct_num = :VAR1, sal= :VAR2; 
```

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The bind values of the bind variables that are used in SQL statements can appear in the following places:

- The audit records when auditing is configured
- The trace files when the appropriate event is set
- The V\$SQL_BIND_DATA dynamic view

The predefined REDACT_AUDIT TSDP policy masks bind values of bind variables that are considered to be sensitive or "associated" with sensitive columns with an ‘*’ value.

By default the REDACT_AUDIT policy is associated with every sensitive type in the database.

Bind variables are considered to be sensitive or "associated" with sensitive columns in the cases listed in the slide.

- In comparison conditions, a sensitive column and a bind variable appear in the expressions that are being compared. In the first example, the sensitive column and the bind variable are the arguments of the IN condition. The bind values in VAR1 and VAR2 are masked because VAR1, VAR2, and SAL appear as arguments to the IN condition.
- The second example shows a condition with a nested subquery as an argument. The bind variables and sensitive columns that appear in the nested subquery are not considered to be associated with the condition.

- If a column in a `SELECT` item is sensitive, then all the binds in the `SELECT` item are considered sensitive. The third example shows a bind variable `VAR1` considered sensitive because it appears in the same `SELECT` item as `SAL`. `VAR2` is considered sensitive because it appears in the same `SELECT` item as the sensitive column `COMM`.
- You can assign multiple bind variables to different columns in one `INSERT` or `UPDATE` statement.

Disabling the REDACT_AUDIT TSDP Policy

Change the behavior of the predefined REDACT_AUDIT TSDP policy.

- Masking bind variables is the default behavior.

TSDP_comp_pol	SYS.UNIFIED_AUDIT_TRAIL
<pre>SELECT (sal*:VAR1), (comm*:VAR2) FROM hr.employees;</pre>	<pre>SQL_BINDS: VAR1 => * VAR2 => *</pre>

- Disabling the policy displays the bind values.

```
DBMS_TSDP_PROTECT.DISABLE_PROTECTION_COLUMN ( policy => 'REDACT_AUDIT' )
```

TSDP_comp_pol	SYS.UNIFIED_AUDIT_TRAIL
<pre>SELECT (sal*:VAR1), (comm*:VAR2) FROM hr.employees;</pre>	<pre>SQL_BINDS: VAR1 => 100 VAR2 => 10</pre>

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The predefined REDACT_AUDIT TSDP policy masks bind values of bind variables that are considered to be sensitive or "associated" with sensitive columns with an '*' because each REDACT_AUDIT is enabled.

You can disable it for a specific sensitive column or all sensitive columns using the DBMS_TSDP_PROTECT.DISABLE_PROTECTION_COLUMN procedure as follows:

- Disable the policy for all sensitive columns:

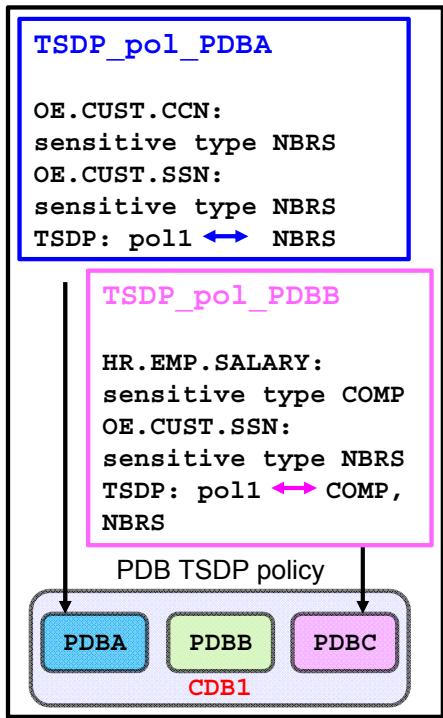
```
SQL> exec DBMS_TSDP_PROTECT.DISABLE_PROTECTION_COLUMN -
      ( policy => 'REDACT_AUDIT' )
```

- Disable the policy for a sensitive column:

```
SQL> exec DBMS_TSDP_PROTECT.DISABLE_PROTECTION_COLUMN -
      ( schema_name => 'HR', -
        table_name  => 'EMPLOYEES', -
        column_name => 'SAL', -
        policy       => 'REDACT_AUDIT' )
```

As soon as you re-enable the REDACT_AUDIT policy, the values for the bind variables are masked with an '*'.

TSDP Policies in a CDB and PDBs



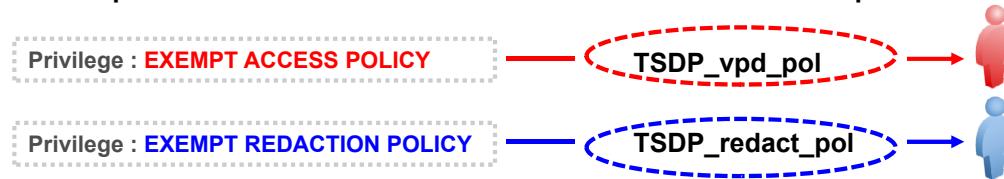
- Create TSDP policies at pluggable database (PDB) level only:
 1. Connect to a PDB.
 2. Create TSDP policies in the PDB.
 3. Enable TSDP policies in the PDB.
- Cannot apply TSDP policies at the multitenant container database (CDB) level.

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

With multitenant container databases (CDB) and pluggable databases (PDB), you can apply TSDP policies to the current PDB only. You cannot apply them to the entire multitenant environment.

Exempting Users from TSDP Policies

- Conditions included in policy expressions may allow users to see actual data.
- SYS is exempt from all TSDP policies.
- Grant a system privilege to exempt other users from all TSDP policies associated to VPD or redaction policies.



- Best Practices:
 - Use default deny (white list) conditions in policy expressions.
 - Grant the EXEMPT ACCESS/REDACTION POLICY privilege judiciously to ensure that the VPD or redaction policies are enforced appropriately.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

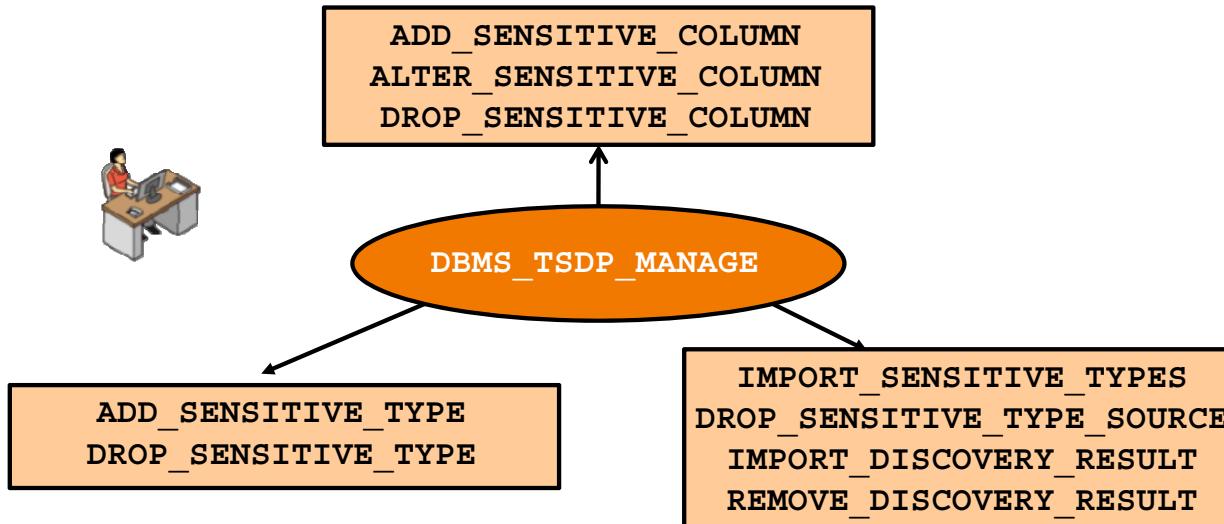
Because the SYS user is exempt from VPD and redaction policies, the user is able to view actual values for data protected by TSDP policies.

If other users need access to the actual values, you must grant them either the EXEMPT ACCESS POLICY system privilege or the EXEMPT REDACTION POLICY system privilege. These privileges exempt the users from all VPD and redaction policies.

Users granted the DBA role are exempt from redaction policies but not from VPD policies because the DBA role contains the EXP_FULL_DATABASE role, which is granted the EXEMPT REDACTION POLICY system privilege.

Because applications may need to perform CREATE TABLE AS SELECT operations that involve redacted source columns, you can grant the application the EXEMPT DDL REDACTION POLICY system privilege.

Maintaining TSDP Types and Sensitive Columns

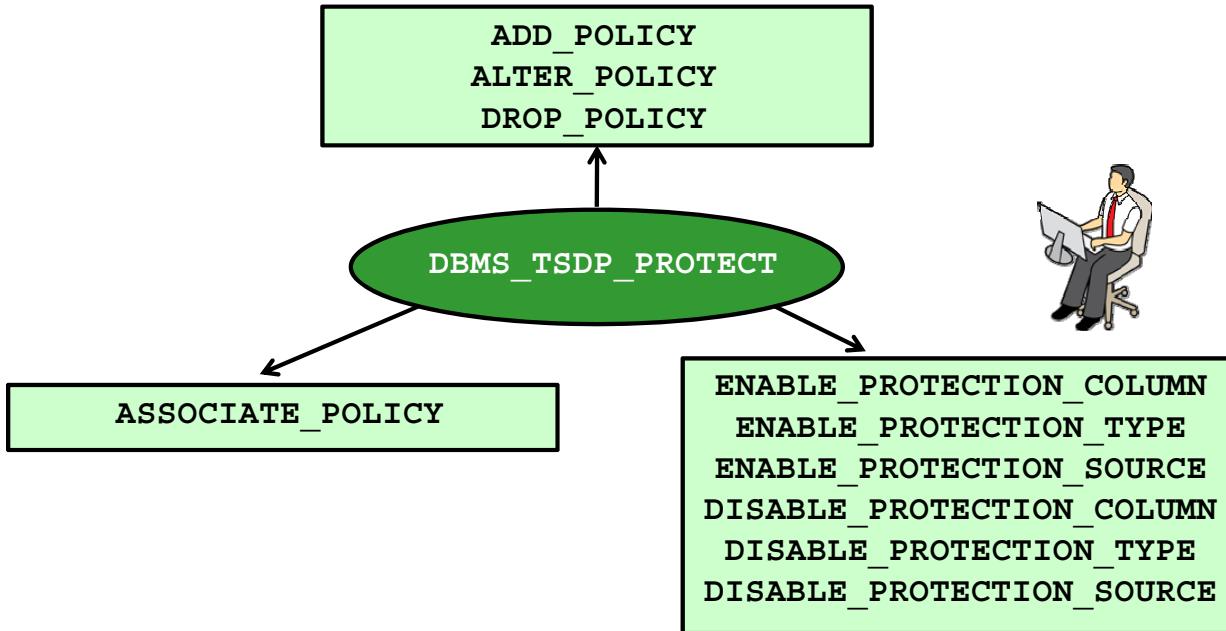


Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The DBMS_TSDP_MANAGE package provides an interface to import and manage sensitive columns and sensitive column types in the database. In most cases, the application database administrator is granted privileges for this package. You can perform the following operations:

- Add a column to the sensitive column list using ADD_SENSITIVE_COLUMN.
- Alter the sensitive type and/or the comment of a column in the sensitive column list using ALTER_SENSITIVE_COLUMN.
- Remove columns from the sensitive column list using DROP_SENSITIVE_COLUMN.
- Create and add a sensitive column type to the list of sensitive column types in the database using ADD_SENSITIVE_TYPE.
- Drop a sensitive column type from the list sensitive column types in the database using DROP_SENSITIVE_TYPE.
- Import a list of sensitive column types from a source using IMPORT_SENSITIVE_TYPES.
- Drop sensitive column types corresponding to a source from the list sensitive column types in the database using DROP_SENSITIVE_TYPE_SOURCE.
- Import sensitive columns from an external source using IMPORT_DISCOVERY_RESULT. This can be ADM from an Oracle EM Cloud Control instance.
- Remove sensitive columns corresponding to an ADM using IMPORT_DISCOVERY_RESULT.

Maintaining TSDP Policies



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The DBMS_TSDP_PROTECT package provides an interface to configure TSDP policies. In most cases, the security database administrator is granted privileges for this package. You can perform the following operations, by using the appropriate procedure:

- Create, alter or remove a TSDP policy using **ADD_POLICY** or **ALTER_POLICY** or **DROP_POLICY**.
- Associate or disassociate a TSDP policy with a sensitive column type using **ASSOCIATE_POLICY**. Setting the **associate** attribute to **TRUE** or **FALSE** associates or disassociates a TSDP policy with a sensitive column.
- Enable protection for columns using **ENABLE_PROTECTION_COLUMN** or for a sensitive column type using **ENABLE_PROTECTION_TYPE** or based on the source of truth for the sensitive columns using **ENABLE_PROTECTION_SOURCE**.
- Disable protection for columns using **DISABLE_PROTECTION_COLUMN** or for a sensitive column type using **DISABLE_PROTECTION_TYPE** or based on the source of truth for the sensitive columns using **DISABLE_PROTECTION_SOURCE**.

Data Dictionary Views for TDSP

- DBA_DISCOVERY_SOURCE: Discovery import information
- DBA_SENSITIVE_COLUMN_TYPES: Sensitive column types
- DBA_SENSITIVE_DATA: Sensitive columns
- DBA_TSDP_IMPORT_ERRORS: Errors encountered during an import of discovery result

- DBA_TSDP_POLICY_FEATURE: TDSP policy security feature mapping (only VPD and Data Redaction)
- DBA_TSDP_POLICY_CONDITION: TDSP policy and condition mapping
- DBA_TSDP_POLICY_PARAMETER: Parameters of TDSP policies

- DBA_TSDP_POLICY_TYPE: Policy to sensitive column type mapping
- DBA_TSDP_POLICY_PROTECTION: List of columns that are protected



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

- After you create the list of sensitive types, you can display the list by using the DBA_SENSITIVE_COLUMN_TYPES view .
- After you add columns to the sensitive column list, you can display the list of identified sensitive columns using the DBA_SENSITIVE_DATA view . If you imported sensitive types and sensitive columns from different sources, these can be known using the DBA_DISCOVERY_SOURCE view .
The type of the source can be either ADM (Import performed from an ADM TSDP catalog) or CUSTOM or DB (discovered within the database).
- After you create the TSDP policy for VPD or Data Redaction, you can view the type of TSDP using the DBA_TSDP_POLICY_FEATURE view . You can view the settings for VPD or Data Redaction using the DBA_TSDP_POLICY_PARAMETER view . Conditions can be set when you create the policy, you can view the conditions using the DBA_TSDP_POLICY_CONDITION view .
- After you associate the TSDP policy with sensitive types, you can view the association using the DBA_TSDP_POLICY_TYPE view .
- After you enable the TSDP policy, you can find the list of columns that are protected using the DBA_TSDP_POLICY_PROTECTION view .
- DBA_TSDP_IMPORT_ERRORS, which shows information about the errors that were encountered during an import of discovery result. It lists the error code, schema name, table name, column name, and sensitive type.

Summary

In this module, you should have learned to:

- Describe the benefits of Transparent Sensitive Data Protection (TSDP)
- Configure a TSDP policy for VPD
- Configure a TSDP policy for Data Redaction
- Configure a TSDP in CDB and PDBs
- Export and import a TSDP policy
- Use the AUDIT_REDACT policy



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Practice: Overview

This practice covers the following topics:

- 16-1: Implementing a TDSP policy associated with a VPD policy
- 16-2: Using the REDACT_AUDIT predefined policy masking bind values
- 16-3: Disabling TDSP policies



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

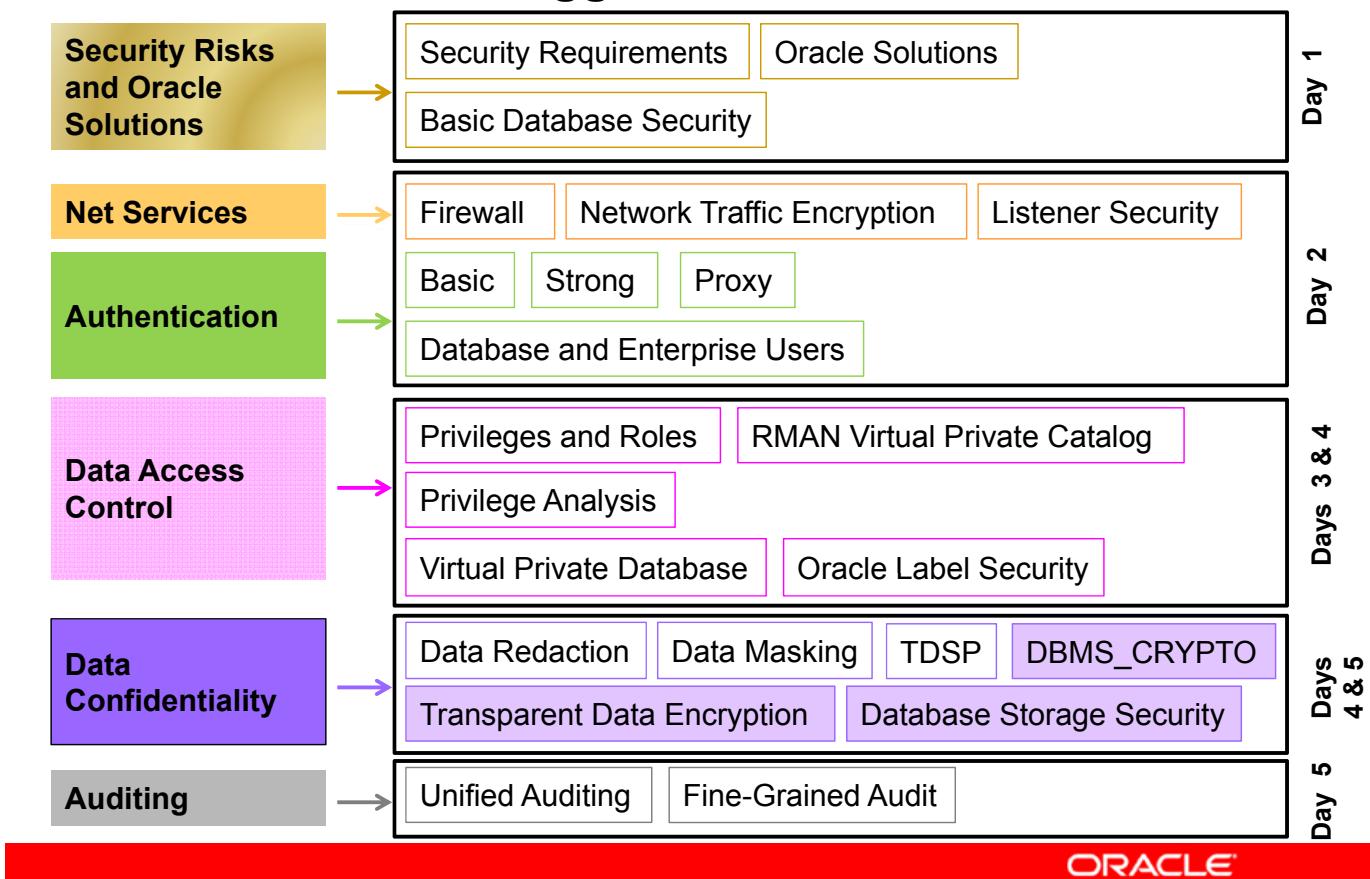
17

Encryption Concepts

ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Suggested Schedule



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

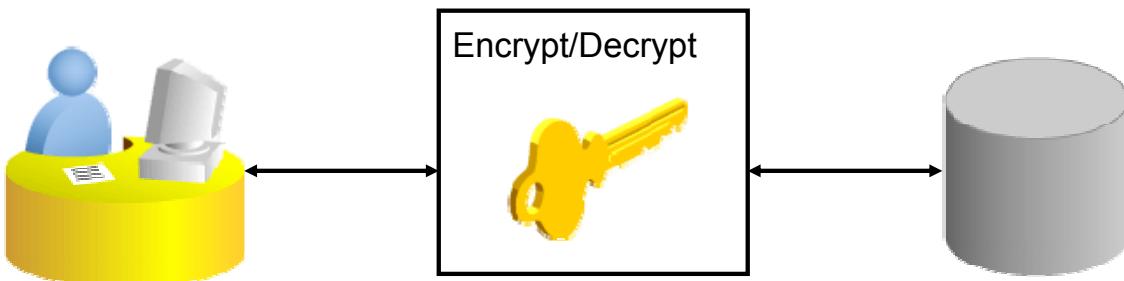
- Discuss issues related to the use of encryption
- Discuss the challenges of encryption
- Describe key management solutions
- Describe the encryption options available with Oracle Database 12c



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Understanding Encryption

- Data is encrypted and decrypted:
 - With an encryption algorithm
 - Using a key or “secret”
- Key management is critical.
 - Secure transmission
 - Secure storage (avoiding loss)



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Encryption in various forms has been around for centuries. All encryption has two parts: an algorithm, which is a procedure or method of manipulating the data, and a key or a secret that allows the data to be decrypted. In the past, for many algorithms, the method was the secret. Modern algorithms are typically public and depend on mathematics to make the algorithm sufficiently complex that the key cannot be guessed or derived from the encrypted data in a reasonable time frame. In modern algorithms, the key is typically a string of numbers or characters.

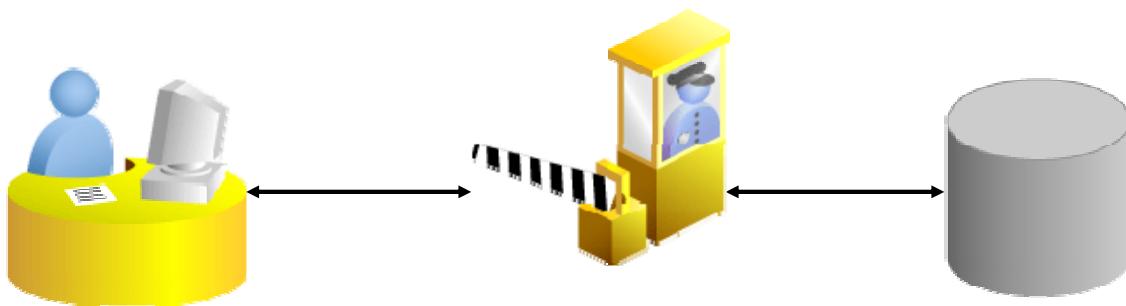
Key management becomes the critical issue with encryption. If the key is lost, the data cannot be decrypted. If the key is mishandled and compromised, the data could be decrypted by unauthorized persons.

The administrative costs of encryption include periodically decrypting and encrypting the data with a new key, keeping the keys secure, and transmitting the keys to an authorized user in a secure manner.

The processing costs of encryption include the time that is required to encrypt and decrypt the data. Generally, a more complex algorithm produces a more secure data set, and requires more processing to encrypt and decrypt the data.

Encryption Is Not Access Control

- Do not use encryption in place of access control.
- Strong data access mechanisms are available.
- Encryption must not interfere with access control.



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Most organizations need to limit data access to those who have a need to know. For example, an HR system may limit employees to reviewing only their own employment records, whereas managers of employees may see the employment records of those employees working for them. HR specialists may also need to see employee records for multiple employees.

These requirements are typically addressed by access control mechanisms, rather than by encryption. The Oracle database provides strong, independently evaluated access control mechanisms.

A basic principle of encryption is that encrypting stored data must not interfere with access control. For example, a user who has the `SELECT` privilege on the `EMPLOYEES` table should not be limited by the encryption mechanism from seeing all the data he or she is otherwise allowed to see.

Access by Privileged Users

- DBAs can access all data. Limit and monitor the DBA by:
 - Using SYSOPER with limited privileges
 - Creating assistant DBA roles to limit access
 - Auditing the actions of the DBA
 - Running background checks on the DBAs
 - Encrypting sensitive columns
 - Using Oracle Database Vault to secure application data.
- The system administrator has access to all data files.
 - Use column Transparent Data Encryption
 - Use tablespace Transparent Data Encryption
- Backup media may be compromised.
 - Use Recovery Manager (RMAN) to create encrypted backups.
 - Use Data Pump Export to create encrypted dump files.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

DBAs Can Access All Data

Some organizations are concerned that DBAs (privileged users) can see all the data in the database, because they typically have all privileges. These organizations feel that DBAs should merely administer the database, but should not be able to see the data that the database contains. Some organizations are also concerned about the concentration of privileges in one person, and would prefer to partition the DBA function and enforce a separation of duties. Limiting the privileges of the DBA is discussed in the lesson titled “Basic Database Security.” The Oracle Database instance can audit all the actions taken by users with the SYSDBA privilege.

Oracle Database Vault resolves these issues, with strong access control.

The DBA job position by its nature requires a trustworthy person. Organizations with the most sensitive data, such as intelligence agencies, scrutinize their DBAs closely because it is a position of trust. If an untrustworthy user has significant privilege, he or she may pose far more significant threats to an organization than viewing unencrypted credit card numbers.

System Administrator Has Access to All Data Files

The system administrator can always access all files on a machine. If the security of the machine is compromised or if data files do not have the proper permissions, others may directly access the data files. Anyone with access to data files and a hex editor can read and decode the information in a data file, given enough time. Encrypting sensitive data in the data files can mitigate a breach and limit the view of the system administrator or someone who manages to gain administrator privileges.

Oracle Advanced Security provides Transparent Data Encryption (TDE) for both columns and tablespaces.

Protecting Backup Media

Backup media must be protected. A knowledgeable user can use a full backup to re-create your database. Encrypting sensitive data can prevent access of sensitive data in backup files in case of lost or stolen backup media. Backup and dump files can be encrypted using RMAN and Data Pump Export respectively.

What Problems Does Encryption Solve?

Encryption protects:

- Data at rest (in files)
- Data in transit (network)



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Encryption is cited as a method to meet Payment Card Industry Data Security Standard (PCI DSS) requirements for data at rest and data in transit.

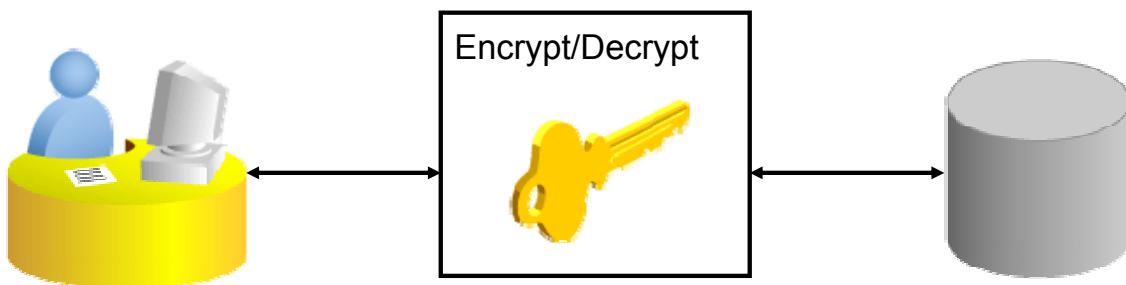
In an Oracle database, the data files without encryption are available to OS privileged users. OS privileged users are those users who may bypass the file permissions and access any file on the operating system. These users may have the ability to read the data files and convert them into intelligible data.

Data transmitted without some form of network encryption is available to anyone who has physical access to the network. These individuals can capture transmitted data. In both cases, encrypted data prevent the intruder from being able to convert it into intelligible data.

Encryption is not an access control solution.

Cost of Encryption

- Encryption and decryption of data
 - Accessibility
 - Performance
- Management of encryption keys
 - Secure transmission
 - Administrative overhead



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Although there are some valid reasons to encrypt data, encryption does not solve all security problems, and may even make some problems worse. Because there is some overhead associated with the encrypting and decrypting of data, it should be applied only in the appropriate situation. The encryption and decryption of data uses CPU resources, whereas the management of encryption keys requires personnel resources.

Because the human resources (HR) records are considered sensitive information, it is tempting to think that this information should be encrypted for better security. However, encryption does not enforce granular access control and may actually hinder data access. In the human resources example, an employee, his or her manager, and the HR clerk need to access the employee's record. If employee data is encrypted, each person must also be able to access the data in unencrypted form. Therefore, the employee, the manager, and the HR clerk would have to share the same encryption key to decrypt the data. Encryption would not provide any additional security in the sense of better access control, and the encryption may actually hinder the proper functioning of the application.

There is the additional issue of securely transmitting and sharing the encryption keys among multiple users of a system.

What to Encrypt

- Encrypting everything does not make data secure.
- Data is unavailable during key changes.
- Lost keys means lost data.
- The management of keys becomes critical.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

It is a pervasive tendency to think that if encrypting some data strengthens security, encrypting everything can make all data secure. Encryption does not make data secure if the issues of key management and transmission are not adequately addressed. It just changes the point of attack. Encryption does not address access control issues. Consider the implications of encrypting an entire production database. All data must be decrypted to be read, updated, or deleted, and the encryption must not interfere with normal access controls. Encryption is a performance-intensive operation; encrypting all data significantly affects performance. Availability is a key aspect of security. If the data is not available in a timely manner because it is encrypted, you have created a new security problem.

Encryption keys must be changed regularly as part of a good security practice, which means that the data is inaccessible while it is being decrypted and re-encrypted with a new key or keys. This also adversely affects availability.

The management of encryption keys is critical. If a key is lost, the encrypted data is also lost. If a key is compromised, the data must be re-encrypted. This is no different from changing the locks when an employee that has the keys leaves the company.

Quiz

Encryption enables you to control who has access to the encrypted data.

- a. True
- b. False



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: b

Data Encryption: Challenges

- Key management:
 - Generation
 - Changing
 - Transmission
 - Storage
- Encrypting special types of data:
 - Indexed
 - Large objects (LOBs)



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Key Management

Key management, including both generation and secure storage of cryptographic keys, is arguably one of the most important aspects of encryption. If keys are poorly chosen or stored improperly, it makes it far easier for an attacker to break the encryption or steal the keys. Rather than searching through all the possible keys in hope to find the correct decryption key, hackers can seek weaknesses in the choice of keys or the way in which keys are stored. Any cryptographic scheme can be broken if there is sufficient time. So changing the keys on a regular basis reduces the time during which it may be possible to discover the key. When a key must be known to multiple persons or machines, how will the keys be protected during transmission?

Encrypting Special Types of Data

Indexes need to be fast to be useful. If the index is on a column that is encrypted, the index should be encrypted as well. This causes performance problems and limits the types of index lookups that are possible.

Large objects have a unique set of problems due to their possible size and considerations for access methods.

Encryption Key Management: Key Generation

Keys are generated with random numbers. Use an approved random-number generator:

- DBMS_CRYPTO.RANDOMBYTES is based on RSA Security Inc. x9.31 PRNG.
- DBMS_OBFUSCATION_TOOLKIT.GETKEY is available for backward compatibility.
- DBMS_RANDOM is *not* approved.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Key generation is an important aspect of encryption. Typically, keys are generated automatically through a random-number generator, from a cryptographic seed. Provided that the random-number generation is sufficiently strong, this can be a secure form of key generation. However, if random numbers are not well formed, but have elements of predictability, the security of the encryption may be easily compromised.

To address the issue of secure cryptographic key generation, Oracle Database 11g supports secure random-number generation. The RANDOMBYTES procedure of DBMS_CRYPTO calls a secure random-number generator based on the RSA Security Inc. x9.31 Pseudo-Random Number Generator (PRNG) standard, which is an approved random-number generator as per the Federal Information Processing Standard (FIPS)-140-2 Annex C.

The GETKEY procedure of the DBMS_OBFUSCATION_TOOLKIT package is an approved FIPS-140 random-number generator and is available for backward compatibility.

Developers should not use the DBMS_RANDOM package that generates pseudo-random numbers, rather than random numbers. As "RFC1750 - Randomness Recommendations for Security" states, "the use of pseudo-random processes to generate secret quantities can result in pseudo-security."

Best Practice: Revoke EXECUTE on DBMS_RANDOM from PUBLIC to prevent any developer from using this package.

Encryption Key Management: Key Modification and Transmission

- Modify periodically, as you would do for a password, to:
 - Reduce the possibility of brute force key discovery
 - Reencrypt the data
- Transmit the keys in a secure manner:
 - Electronic transmission (encrypt the key)
 - Physical transmission



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Key Modification

Prudent security practice dictates that you periodically change encryption keys. For stored data, this requires periodically decrypting the data and reencrypting it with another well-chosen key. This should be done while the data is not being accessed. This creates another challenge for applications that need to be continuously available. Consider a Web-based application encrypting credit card numbers. You do not want to make the entire application unavailable while you switch encryption keys.

Key Transmission

If the key is used outside the database, is used by multiple users, or is changed, the key must be transmitted in some manner. The method depends on where and to whom the key is to be passed. If the key is to be passed over the network, it must be encrypted. Otherwise, a network snooper may grab the key as it is being transmitted. The use of network encryption protects all data in transit from modification or interception, including cryptographic keys. The physical transmission of the key poses the risk of losing the key copy or having it stolen.

Note: As of the release of Oracle Database 12c, network encryption (native network encryption and SSL/TLS) and strong authentication services (Kerberos, PKI, and RADIUS) are no longer part of Oracle Advanced Security and are available in all licensed editions of the Oracle database.

Encryption Key Management: Storage

Store the keys by using one of the following methods:

- Store the key in the database.
- Store the key in an operating system file.
- Let the user manage the key.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Key storage is one of the most important, yet difficult, aspects of encryption. To recover data encrypted with a symmetric key, the key must be accessible to the application or user seeking to decrypt data. Users must be able to access encrypted data without significant performance degradation. Authorized users must be able to retrieve the key quickly, but the key must be secure enough so that an unauthorized user cannot easily recover it.

There are three basic options:

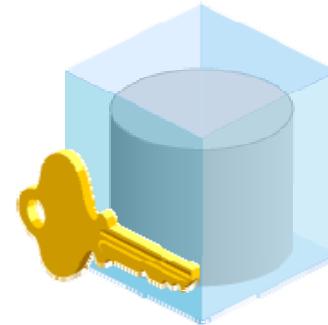
- Store the key in the database.
- Store the key in an operating system file.
- Let the user manage the key.

The details of each of these options are covered later in this lesson.

Storing the Key in the Database

The techniques for protecting keys in the database are:

- Store keys in a separate table and schema.
- Perform additional data transformation.
- Wrap the PL/SQL package that performs the encryption.
- Use one key per row.
- Combine the techniques.



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Storing the keys in the database cannot protect data against the DBA accessing encrypted data. An all-privileged DBA can access tables containing encryption keys. However, it can often provide good security against the casual snooper or against someone reading the database file from a compromised operating system account. If an intruder gets access to the encryption key, he or she can decrypt and read the encrypted data. So it is important to keep the key private.

The techniques for providing additional security for encryption include the following:

- Never store the key in the same table as the encrypted data. When hackers have access to the table, they can decrypt the data. Store the key in a separate schema, and use PL/SQL techniques to control access to the key.
- Encrypt the data with a technique that performs some additional data transformation, such as an XOR operation on the encrypted data with another value. You can use another column, but that column must not change. For example, you can use the primary key of the row as the additional value.
- Wrap the PL/SQL package body that performs encryption. The wrap utility transforms the code so that the package body is not human readable. For example, to wrap a package body stored in the `keymanage.sql` file, enter the following command:

```
wrap iname=keymanage.sql
```

Even in cases where a different key is supplied for each encrypted data value (so that the value of the key is not embedded within a package), wrapping the package that performs key management is recommended. Additional information about the wrap utility is available in the *PL/SQL User's Guide and Reference*.

Note that even when the key is wrapped in a package, it may still be viewed in the DBA_SOURCE view. Therefore, any user who has the SELECT privilege on this view can see the key.

- Rather than using one key for the entire column, create a separate key for each row and store the encryption key in another table. The key table can be joined to the data table by using a primary key–foreign key relationship. For example, SALES_ID is the primary key in the SALES table, which stores sales transactions and the encrypted CREDIT_CARD_NO value. SALES_ID is a foreign key to the SALES_KEYS table, which stores the encryption keys for each credit card number.

The strengths of this approach are the following:

- Users who have direct table access cannot see the sensitive data unencrypted, nor can they retrieve the keys to decrypt the data.
- Access to decrypted data can be controlled through a procedure that selects the encrypted data, retrieves the decryption key from the key table, and transforms the key before it can be used to decrypt the data.
- The data transformation algorithm is hidden from casual snooping by wrapping the procedure, which obfuscates the procedure code.
- Read access to both the data table and the key table does not guarantee that the user with this access can decrypt the data, because the key is transformed before use.

The weakness in this approach is that a user who has SELECT access to both the key table and the data table (who can derive the key transformation algorithm) can break the encryption scheme.

This approach does not guarantee complete security; it can protect against easy retrieval of sensitive information that is stored encrypted (for example, credit card numbers) and the data is encrypted in the data files protecting the data from access through OS files and backup media.

Storing the Key in the File System

Use this method to restrict DBA access to the keys:

- Set up the file storing the keys so that the DBA does not have access to the file.
- Retrieve the data from the database without decrypting the data.
- Decrypt the data in the application accessing the data. The DBA must also be denied access to this application.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Storing keys in a file system is another option. You can use the UTL_FILE package to make callouts from PL/SQL, which you can use to retrieve encryption keys. However, if you store keys in a file system and make callouts to it, your data is only as secure as the protection on the file system. If you are encrypting data stored in the database because the database can be broken into from the file system, storing the keys in a file system makes it easier for an intruder to retrieve encrypted data than if the keys were stored in the database itself.

The following technique may be used to hide the data from the DBA:

1. Set up the file storing the keys so that the DBA does not have access to the file.
2. Retrieve the data from the database without decrypting the data.
3. Decrypt the data in the application accessing the data. The DBA must also be denied access to this application.

Letting the User Manage the Key

User-managed keys have the following problems:

- Users forget the key.
- Users archive the key in an insecure manner.
- Users must use secure transmission methods, such as network encryption.



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Having the user supply and manage the key is another technique for managing keys. However, considering the estimate that 70% of the help-desk calls are from the users who have forgotten their passwords, you can see the risks of having users manage encryption keys. In all likelihood, users either forget an encryption key or write the key down, which creates a security weakness. If a user forgets an encryption key or leaves the company, your data is unrecoverable.

If you elect to have user-supplied or user-managed keys, you must use network encryption so that the key is not passed from the client to the server unencrypted. You must also develop key archive mechanisms, which pose another difficult security problem. Arguably, key archives create a security weakness that encryption attempts to address in the first place.

Solutions

- Oracle Database Vault
- Transparent Data Encryption (TDE)
- File encryption
 - RMAN backup encryption
 - Oracle Secure Backup (OSB)
- Application encryption



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Depending on the problem that you are trying to solve, there are several solutions. Each solution solves a particular set of problems.

Oracle Database Vault: If the reason for encryption is to limit access by the DBA, Database Vault provides powerful tools to enforce separation of duties and limit access of schema data to authorized persons.

Transparent Data Encryption: It is an option that is integrated with the database when the Oracle Advanced Security option is installed. With column-level encryption, the data is encrypted in the database files and System Global Area (SGA). Columns are encrypted as declared in the table definitions. With tablespace encryption, the entire tablespace is encrypted, and the encryption and decryption is handled in the I/O layer. The encryption keys are stored encrypted in the database, and a master key is stored in an external wallet. All access is controlled by standard database access control methods.

File encryption: It provides the ability to encrypt sensitive data in backup files and dump files.

Application Encryption: It is the last resort. It has the advantage of limiting the access of sensitive data to only those users that have access to the encryption key, but has all the disadvantages inherent in a custom application, plus the issues of key management.

Summary

In this lesson, you should have learned how to:

- Discuss the issues of encryption
- Discuss the challenges of encryption
- Describe key management solutions
- Describe the encryption options available with Oracle Database 12c



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Error : You are not a Valid Partner use only

18

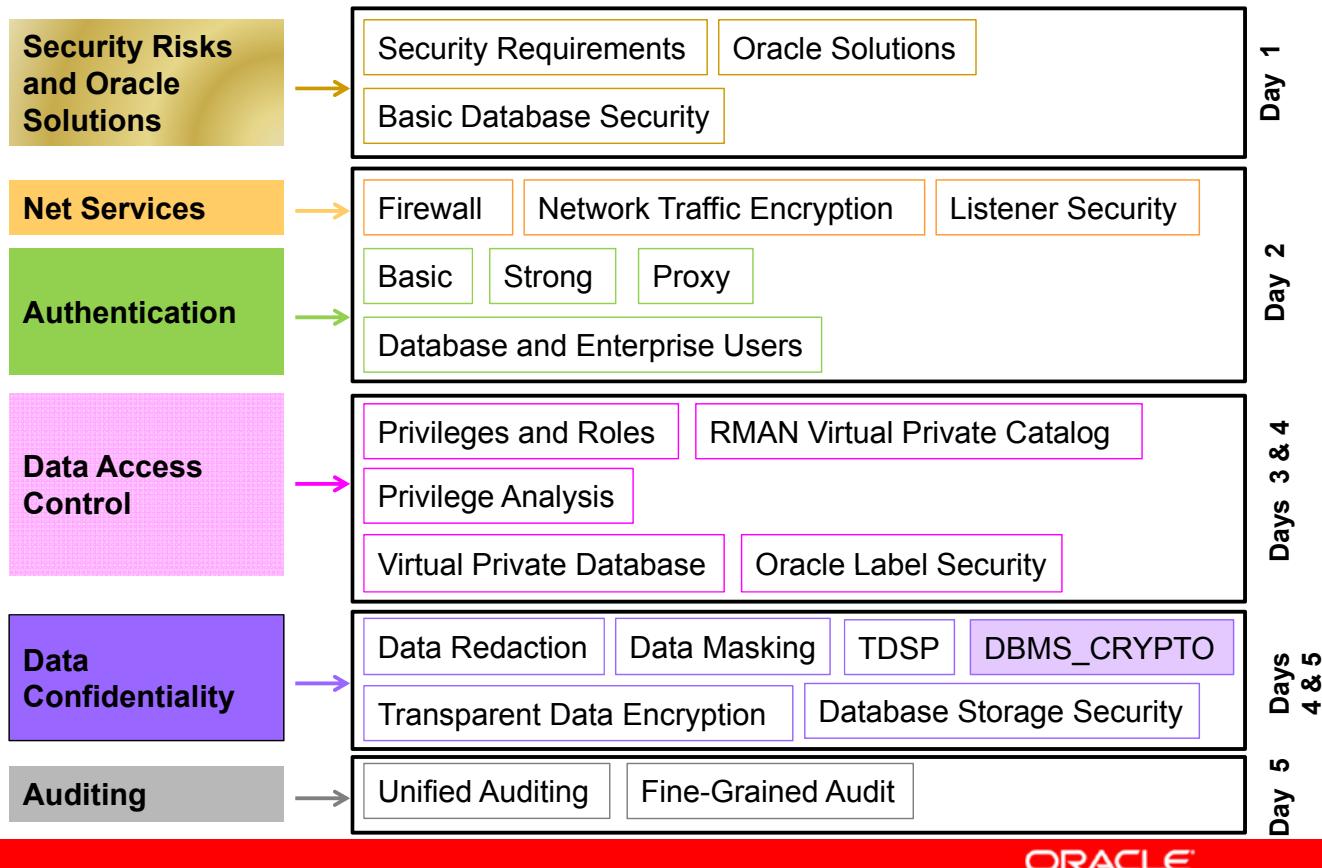
Using Application-Based Encryption



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Suggested Schedule



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to use DBMS_CRYPTO to:

- Generate random encryption keys
- Encrypt and decrypt table columns

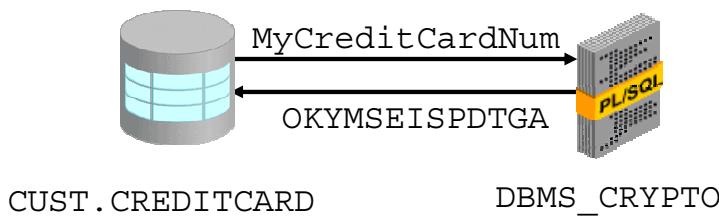


Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Overview

DBMS_CRYPTO package:

- Encrypts column data
- Decrypts column data
- Supersedes DBMS_OBFUSCATION_TOOLKIT



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Among other security technologies, Oracle Database protects data through strong, standards-based encryption. Encryption of network data is supported through Oracle Advanced Security. Encryption of column data is supported using Transparent Data Encryption (TDE) within the database.

Note

This lesson addresses encryption of data that is stored in the database. It does not consider the issue of encryption of data as it passes through the network. For information about the topic of secure transmission, see the lesson titled “Oracle Net Services: Security Checklists.”

Oracle Database 11g provides two methods for encrypting column data: application-based encryption and Transparent Data Encryption. This lesson examines application-based encryption.

The DBMS_CRYPTO package is provided to perform encryption and decryption. This package supports bulk data encryption and includes procedures to encrypt and decrypt data, as well as a random number generator for generating secure encryption keys.

This lesson is provided for users who require support for application-based encryption. Oracle Corporation does not recommend using application-based encryption.

DBMS_CRYPTO Package

- Functionality:
 - Random-number generation for encryption keys
 - Encryption and decryption by using various algorithms
 - Multiple cipher block chaining modes
 - Multiple cryptographic hash algorithms
 - Multiple padding forms
- Procedures and functions in the package include:
 - DECRYPT to decrypt columns or LOBs
 - ENCRYPT to encrypt columns or large objects (LOBs)
 - HASH to apply a hash algorithm to data
 - RANDOMBYTES to create random keys



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

DBMS_CRYPTO contains basic cryptographic functions and procedures. To use this package correctly and securely, a general level of security expertise is assumed.

The DBMS_CRYPTO package enables encryption and decryption for common Oracle data types, including RAW and large objects (LOBs), such as images and sound. Specifically, it supports binary large objects (BLOBs) and character large objects (CLOBs). In addition, it provides globalization support for encrypting data across different database character sets.

The following cryptographic algorithms are supported:

- Data Encryption Standard (DES), Triple DES (3DES: 2-key and 3-key)
- Advanced Encryption Standard (AES)
- MD5, MD4, and SHA-1 cryptographic hashes
- MD5 and SHA-1 Message Authentication Code (MAC)

Block cipher modifiers are also provided with DBMS_CRYPTO. You can choose from several padding options, including Public-Key Cryptographic Standard (PKCS) #5, and from four block cipher chaining modes, including Cipher Block Chaining (CBC).

The DBMS_CRYPTO package is installed in the SYS schema. You can grant package access to existing users and roles as needed.

The `ENCRYPT` and `DECRYPT` *procedures* are used to encrypt and decrypt the `LOB` data types (overloaded for `CLOB` and `BLOB` data types). In contrast, the `ENCRYPT` and `DECRYPT` *functions* are used to encrypt and decrypt the `RAW` data types. Data of the `VARCHAR2` type must be converted to `RAW` before you can use the `DBMS_CRYPTO` functions to encrypt it.

The package includes the following program units:

- The `DECRYPT` function decrypts the `RAW` data by using a stream or block cipher with a user-supplied key and optional initialization vector (IV).
- `DECRYPT` procedures decrypt the `LOB` data by using a stream or block cipher with a user-supplied key and optional IV.
- The `ENCRYPT` function encrypts the `RAW` data by using a stream or block cipher with a user-supplied key and optional IV.
- `ENCRYPT` procedures encrypt the `LOB` data by using a stream or block cipher with a user-supplied key and optional IV.
- The `HASH` function applies one of the supported cryptographic hash algorithms (MD4, MD5, or SHA-1) to data.
- The `MAC` function applies Message Authentication Code algorithms (MD5 or SHA-1) to data to provide keyed message protection.
- The `RANDOMBYTES` function returns a `RAW` value containing a cryptographically secure pseudo-random sequence of bytes, and can be used to generate random material for encryption keys.
- The `RANDOMINTEGER` function returns a random `BINARY_INTEGER`.
- The `RANDOMNUMBER` function returns a random 128-bit integer of the `NUMBER` data type.

Note: An initialization vector (IV) is a block of plain text that is used with block ciphers. The IV is combined with the text to be encrypted. How the IV is used depends on the block cipher algorithm.

Generating Keys Using RANDOMBYTES

- Generate a key:

```
raw_key := dbms_crypto.randombytes (
    number_bytes => 24);
```

- Encrypt:

```
encrypted_raw := dbms_crypto.encrypt (
    src => raw_input,
    typ => DBMS_CRYPTO.DES3_CBC_PKCS5
    key => raw_key);
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

This function returns a `RAW` value containing a cryptographically secure pseudo-random sequence of bytes, which can be used to generate random material for encryption keys. The `RANDOMBYTES` function is based on RSA Security Inc. X9.31 Pseudo-Random Number Generator (PRNG), and it draws its entropy (seed) from the `sqlnet.ora` file parameter, `SQLNET.CRYPTO_SEED`.

NUMBER_BYTES: It is the number of bytes returned. This parameter allows the `RANDOMBYTES` function to provide key values for various encryption algorithms. In the previous example, the DES algorithm requires a key at least 8 bytes long. In this example, the key length for DES3 is 24. The AES algorithms can use 16-, 24-, or 32-byte key lengths. The AES keys must be exactly the correct number of bits; the DES algorithms discard extra key bits.

Example:

The code in the slide uses functions to produce a key and to encrypt and decrypt the data. It is taken from the example on the following page and from the `/home/oracle/labs/demos/demo_randombytes.sql` script.

```
-- demo_randombytes.sql ---
DECLARE
    input_string      VARCHAR2(16) := 'CreditCardNumber';
    raw_input         RAW(128) := UTL_I18N.STRING_TO_RAW(input_string, 'AL32UTF8');

    raw_key          RAW(256);

    encrypted_raw    RAW(2048);
    encrypted_string VARCHAR2(2048);
    decrypted_raw    RAW(2048);
    decrypted_string VARCHAR2(2048);

BEGIN
    dbms_output.put_line('> ===== Get Key Bytes =====');
    raw_key := dbms_crypto.randombytes(24);
    dbms_output.put_line('> Key String length: ' ||
        UTL_RAW.LENGTH(raw_key));

    dbms_output.put_line('> Key String: ' ||
        UTL_RAW.RAW_TO_VARCHAR2(raw_key));

    dbms_output.put_line('> Input String: ' || input_string);
    dbms_output.put_line('> ===== BEGIN TEST Encrypt =====');
    encrypted_raw := dbms_crypto.Encrypt(
        src => raw_input,
        typ => DBMS_CRYPTO.DES3_CBC_PKCS5,
        key => raw_key);

    dbms_output.put_line('> Encrypted hex value : ' ||
        rawtohex(UTL_RAW.CAST_TO_RAW(encrypted_raw)));

    decrypted_raw := dbms_crypto.Decrypt(
        src => encrypted_raw,
        typ => DBMS_CRYPTO.DES3_CBC_PKCS5,
        key => raw_key);

    decrypted_string := UTL_I18N.RAW_TO_CHAR(decrypted_raw, 'AL32UTF8');

    dbms_output.put_line('> Decrypted string output : ' ||
        decrypted_string);
    ...

```

```
...
if input_string = decrypted_string THEN
    dbms_output.put_line('> String DES Encryption and Decryption
successful');
END if;
END;
/

> ===== Get Key Bytes =====
> Key String length: 24
Key String: )??Q?? ?]?=5-2/8C
> Input String: CreditCardNumber
> ===== BEGIN TEST Encrypt =====
> Encrypted hex value :
413035444539423746363935373335333037384139363139454346423533354537
    313933383832414144333139333346
> Decrypted string output : CreditCardNumber

> String DES Encryption and Decryption successful
PL/SQL procedure successfully completed.
```

Quiz

The DBMS_CRYPTO package is provided to enable users to:

- a. Generate random encryption keys
- b. Implement application-based column encryption
- c. Implement encryption of network data



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: a, b

Using ENCRYPT and DECRYPT

- ENCRYPT:

```
encrypted_raw := dbms_crypto.Encrypt (
    src => raw_input,
    typ => dbms_crypto.DES3_CBC_PKCS5,
    key => raw_key);
```

- DECRYPT:

```
decrypted_raw := dbms_crypto.Decrypt (
    encrypted_raw,
    dbms_crypto.DES3_CBC_PKCS5,
    raw_key);
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The simplest way to encrypt and decrypt data is by using the ENCRYPT and DECRYPT functions.

Multiple Encryption Passes

You cannot execute multiple passes of encryption. The 3DES algorithm encrypts data multiple times. You cannot call the ENCRYPT function more than once to encrypt the same data. If the user tries to encrypt data more than once, the procedure raises the following error:
ORA-28233 "Source data was previously encrypted"

A full example can be executed with the

/home/oracle/labs/demos/demo_double_encrypt.sql script.

Example:

The code in the slide uses functions to encrypt and decrypt the data. A full example using the ENCRYPT and DECRYPT functions is on the following page from the /home/oracle/labs/demos/demo_encrypt.sql script.

Notice that SRC, KEY, and returned values are RAW data types. The developer is responsible for converting the character or number data types to and from RAW. The UTL_RAW package simplifies this process. The UTL_I18N package performs similar functions with the added feature of allowing for character set conversions.

To encrypt VARCHAR2 data, it should first be converted to the AL32UTF8 character set. To convert VARCHAR2 to RAW, use the UTL_I18N.STRING_TO_RAW function as in the example to perform the following steps:

1. Convert VARCHAR2 in the current database character set to VARCHAR2 in the AL32UTF8 database character.
2. Convert VARCHAR2 in the AL32UTF8 database character set to RAW.

```
-- demo_encrypt.sql
DECLARE
    input_string      VARCHAR2(16)  := 'CreditCardNumber';
    raw_input         RAW(128)    :=
        UTL_I18N.STRING_TO_RAW(input_string, 'AL32UTF8');

    key_string        VARCHAR2(8)   := 'ADGJLZCB';
    raw_key          RAW(128)    :=
        UTL_I18N.STRING_TO_RAW(key_string, 'AL32UTF8');

    encrypted_raw     RAW(2048);
    encrypted_string  VARCHAR2(2048);
    decrypted_raw     RAW(2048);
    decrypted_string  VARCHAR2(2048);

BEGIN
    dbms_output.put_line('> Input String: ' || input_string);
    dbms_output.put_line('> ===== BEGIN TEST Encrypt =====');
    encrypted_raw := dbms_crypto.Encrypt(
        src => raw_input,
        typ => DBMS_CRYPTO.DES_CBC_PKCS5,
        key => raw_key);

    dbms_output.put_line('> Encrypted hex value : ' ||
        rawtohex(UTL_RAW.CAST_TO_RAW(encrypted_raw)));

    decrypted_raw := dbms_crypto.Decrypt(
        src => encrypted_raw,
        typ => DBMS_CRYPTO.DES_CBC_PKCS5,
        key => raw_key);

    decrypted_string := UTL_I18N.RAW_TO_CHAR(decrypted_raw, 'AL32UTF8');

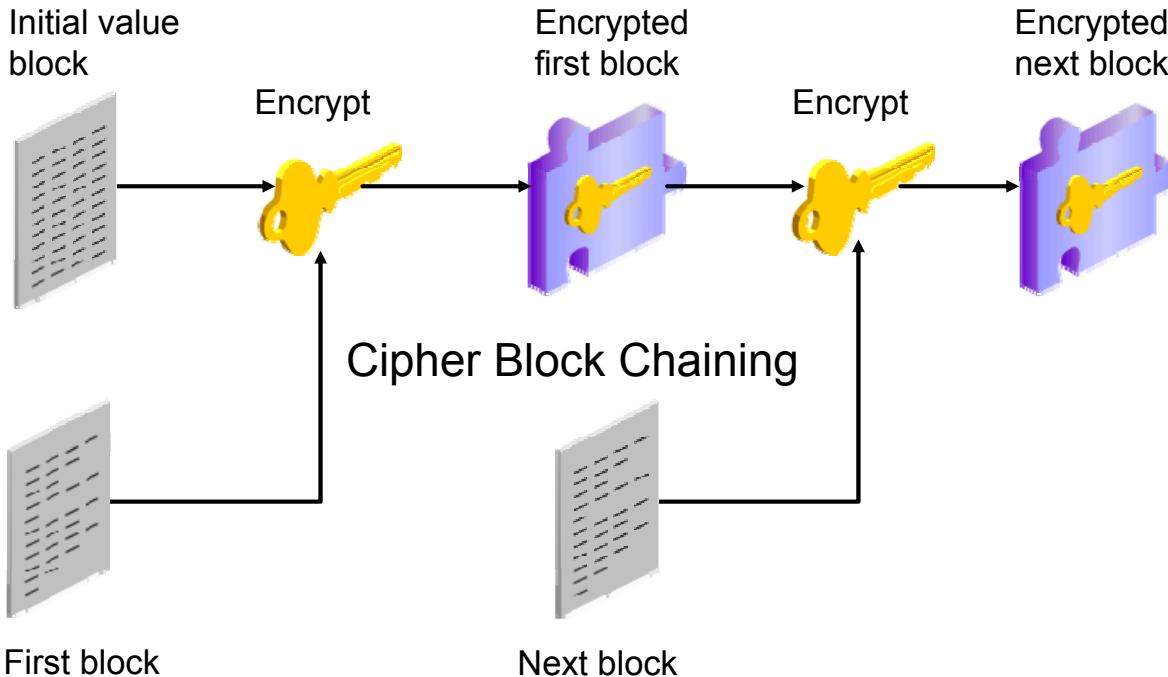
    dbms_output.put_line('> Decrypted string output : ' ||
        decrypted_string);
    ...

```

```
if input_string = decrypted_string THEN
    dbms_output.put_line('> String DES Encryption and Decryption
successful');
END if;
END;
/
SQL> set serveroutput on
SQL> @demo_encrypt.sql
...
> Input String: CreditCardNumber
> ===== BEGIN TEST Encrypt =====
> Encrypted hex value :
38333538373934453630444644383644303242384344431323646384331374644
    324242424537443332344332383433
> Decrypted string output : CreditCardNumber
> String DES Encryption and Decryption successful

PL/SQL procedure successfully completed.
```

Enhanced Security Using Cipher Block Modes



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The DBMS_CRYPTO encryption procedures allow you to choose one of the four cipher block modes:

- **CHAIN_ECB (Electronic Codebook)**: Encrypts each plain text block independently
- **CHAIN_CBC (Cipher Block Chaining)**: Combines a block of plain text with the previous cipher text block in an XOR operation before it is encrypted (shown in the slide)
- **CHAIN_CFB (Cipher-Feedback)**: Enables encrypting units of data smaller than the block size
- **CHAIN_OFB (Output-Feedback)**: Enables running a block cipher as a synchronous stream cipher. This is similar to CFB, except that n bits of the previous output block are moved into the right-most positions of the data queue waiting to be encrypted.

Cipher Block Chaining (CBC) mode is the most common and the strongest of the modes. You can further secure your data by including a 64-byte block of nonsense text with your data as an initial value block. In CBC mode, before a block of plain text is encrypted, it is combined with the previous encrypted block in an XOR operation. CBC mode enhances security because every block depends on its predecessors, which makes the breaking of the code more difficult. The block CBC is 64 bits (8 bytes) long. The disadvantage is that if any block is lost, none of the following blocks can be decrypted.

Hash and Message Authentication Code

- DBMS_CRYPTO includes HASH and Message Authentication Code (MAC) functions.
- Both produce a one-way hash of a LOB or RAW data type.
- Use hash values to verify data integrity.
- MAC uses a secret key.
- Example:

```
encrypted_raw := dbms_crypto.Mac(  
    src => raw_input,  
    typ => DBMS_CRYPTO.HMAC_MD5,  
    key => raw_key);
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The DBMS_CRYPTO package includes two different types of one-way hash functions: the HASH function and the MAC function. Hash functions operate on an arbitrary-length input message and return a fixed-length hash value. One-way hash functions work in one direction only. It is easy to compute a hash value from an input message, but it is extremely difficult to generate an input message that hashes to a particular value. Note that hash values must be at least 128 bits in length to be considered secure.

You can use hash values to verify whether data has been altered. For example, before storing data, you can run DBMS_CRYPTO.HASH against the stored data to create a hash value. When you retrieve the stored data at a later date, you can again run the hash function against it, using the same algorithm. If the second hash value is identical to the first one, the data has not been altered. Hash values are similar to “file fingerprints” and are used to ensure data integrity.

The HASH function included with DBMS_CRYPTO is a one-way hash function that you can use to generate a hash value from either the RAW or LOB data. The DBMS_CRYPTO.MAC function is also a one-way hash function, but with the addition of a secret key. It works in the same way as the DBMS_CRYPTO.HASH function, except that only someone with the key can verify the hash value.

Hash functions provide a reasonable way to store encrypted passwords.

MAC functions can be used to authenticate files between users. They can also be used by a single user to determine whether that user's files have been altered, perhaps by a virus. A user can compute the MAC value of the files and store that value in a table. If the user does not use a MAC function, the virus can compute the new hash value after infection and replace the table entry. A virus cannot do that with a MAC value because the virus does not know the key.

Example

The code in the following example produces a hash value and a MAC value from an input string. You can execute the `/home/oracle/labs/demos/demo_hash.sql` script.

```
-- demo_hash.sql --
DECLARE
    input_string      VARCHAR2(16) := 'CreditCardNumber';
    raw_input         RAW(128) := UTL_I18N.STRING_TO_RAW(input_string, 'AL32UTF8');
    raw_key          RAW(256);
    encrypted_raw    RAW(2048);
    encrypted_string VARCHAR2(2048);
    decrypted_raw    RAW(2048);
    decrypted_string VARCHAR2(2048);
BEGIN
    dbms_output.put_line('> ====== Get Key Bytes ======');

    raw_key := dbms_crypto.randombytes(24);

    dbms_output.put_line('> Key String length: ' || UTL_RAW.LENGTH(raw_key));
    dbms_output.put_line('> Key String: ' || UTL_RAW.CAST_TO_VARCHAR2(raw_key));
    dbms_output.put_line('> Input String: ' || input_string);
    dbms_output.put_line('');
    dbms_output.put_line('> ====== BEGIN TEST Hash ======');

    encrypted_raw := dbms_crypto.Hash(
        src => raw_input,
        typ => DBMS_CRYPTO.HASH_SH1);

    dbms_output.put_line('> Hash value of input string : ' ||
        rawtohex(UTL_RAW.CAST_TO_RAW(encrypted_raw)));
    dbms_output.put_line('> ====== BEGIN TEST Mac ======');
    ...

```

```
...
encrypted_raw := dbms_crypto.Mac(
    src => raw_input,
    typ => DBMS_CRYPTO.HMAC_MD5,
    key => raw_key);

dbms_output.put_line('> Message Authentication Code : ' || 
    rawtohex(UTL_RAW.CAST_TO_RAW(encrypted_raw)));
dbms_output.put_line('> End of Hash and MAC tests');
END;
/
```

```
SQL> connect / as sysdba
Connected.
SQL> set serveroutput on
SQL> @demo_hash.sql
> ====== Get Key Bytes ======
> Key String length: 24
> Key String: ??#f?????*3f5?
> Input String: CreditCardNumber
> ====== BEGIN TEST Hash ======
> Hash value of input string :
4136384633434630394144313531454236353335423343434144304634433042374
4334145373130
> ====== BEGIN TEST Mac ======
> Message Authentication Code :
3731334437364243413334383041323131384544374542424635454434434541
> End of Hash and MAC tests
```

Summary

In this lesson, you should have learned how to use DBMS_CRYPTO to:

- Generate random encryption keys
- Encrypt and decrypt table columns



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Practice: Overview

This practice covers the following topics:

- 18-1: Using DBMS_CRYPTO for Encryption
- 18-2: Checksumming by using the HASH function



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

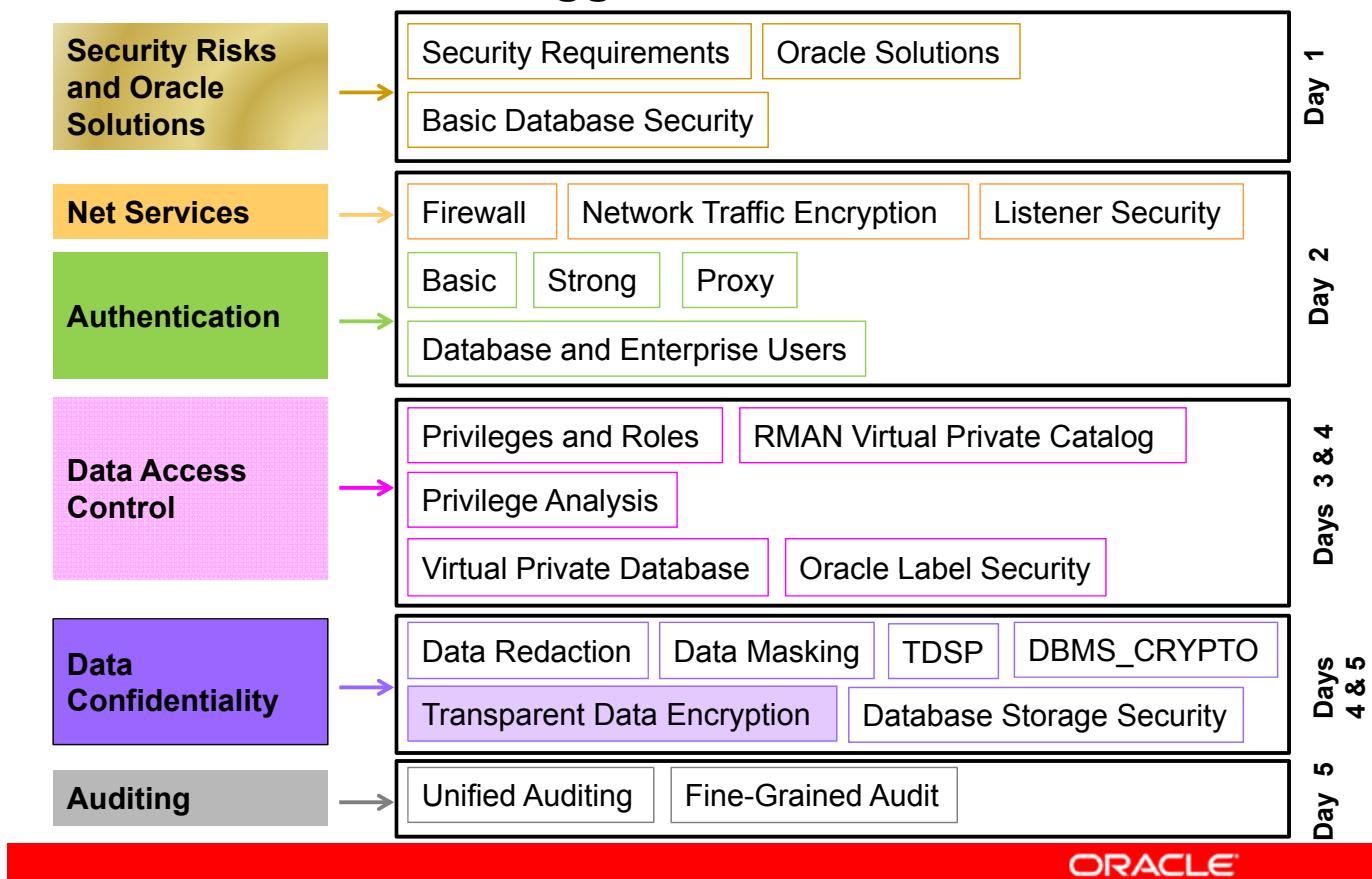
19

Applying Transparent Data Encryption

ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Suggested Schedule



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Objectives

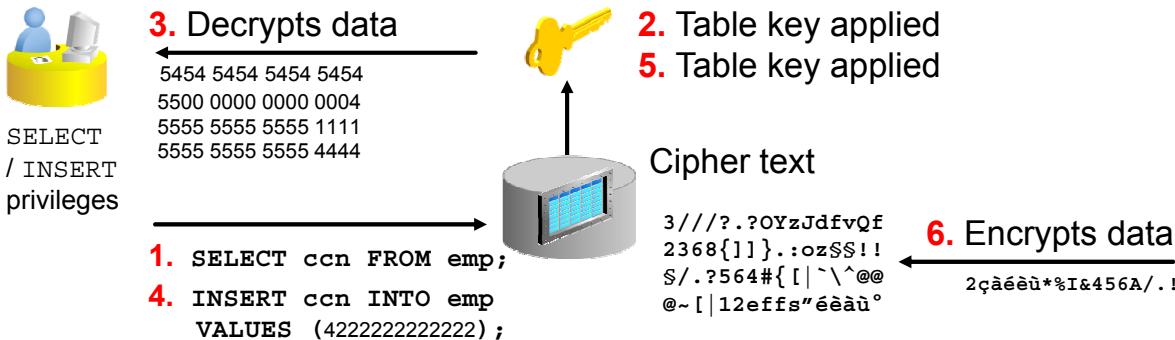
After completing this lesson, you should be able to do the following:

- Implement Transparent Data Encryption
- Configure the keystore
- Set up the database master encryption key
- Encrypt column data
- Encrypt tablespace data



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Transparent Data Encryption



- Encrypts data in:
 - Data files (tablespaces, columns, indexes)
 - Redo log and archive log files
 - Memory (only for column encryption)
 - File backups
- Manages keys automatically
- Does not require changes to the application

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Transparent Data Encryption (TDE) is available with Oracle Advanced Security and provides easy-to-use protection for your data without requiring changes to your applications. TDE allows you to encrypt sensitive data in individual columns or entire tablespaces without having to manage encryption keys. TDE does not affect access controls, which are configured using database roles, secure application roles, system and object privileges, views, Virtual Private Database (VPD), Oracle Database Vault, and Oracle Label Security. Any application or user that previously had access to a table will still have access to an identical encrypted table.

TDE is designed to protect data in storage, but does not replace proper access control.

TDE is transparent to existing applications. Encryption and decryption occurs at different levels depending on whether it is tablespace or column level, but in either case, encrypted values are not displayed and are not handled by the application. For example, with TDE, applications designed to display a 16-digit credit card number do not have to be recoded to handle an encrypted string that may have many more characters.

TDE eliminates the ability of anyone who has direct access to the data files to gain access to the data by circumventing the database access control mechanisms. Even users with access to the data file at the operating system level cannot see the data unencrypted. TDE stores the master key outside the database in an external security module, thereby minimizing the possibility of both personally identifiable information (PII) and encryption keys being compromised. TDE decrypts the data only after database access mechanisms have been satisfied.

Components of TDE

- Key architecture
 - Two-tier architecture: Unified master encryption key stored in an external security module is used to encrypt the table key or tablespace key
 - Low overhead re-key operation: Some security regulations require periodical changes of encryption keys.
- External security module
 - Software keystore
 - Hardware keystore (HSM)
- Algorithm support



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

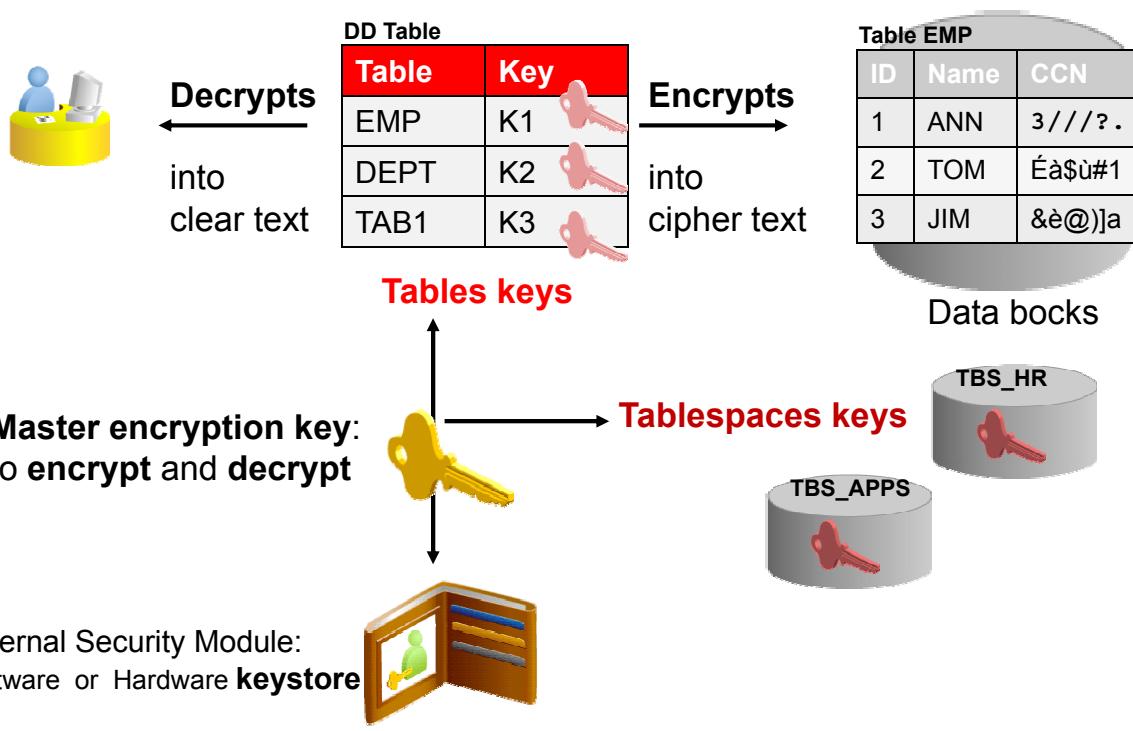
TDE applies the principle of defense in depth in its design. The key architecture is a two-tier system. The master key is stored in an external security module. This is either an Oracle software keystore or a hardware keystore also called as Hardware Security Module (HSM). In previous releases, the term "software keystore" was called a "wallet". This external store is protected by a password, operating system permissions, and encryption. The master encryption key is used to encrypt the table key (for column encryption) and the tablespace key (for tablespace encryption). The table key or tablespace key is then used to encrypt the data.

Some security regulations require a periodic change of encryption keys. This change of keys means that the items that are encrypted be decrypted with the old key and encrypted with the new key. This is also called re-keying.

A major advantage of the two-tier architecture is that the table-level keys can be re-keyed by changing the master key. This automatically causes table-level keys to be re-encrypted using the new master key, but the table-level keys remain unchanged. So the data does not require re-keying. This operation meets the Payment Card Industry requirement for re-keying, with a minimum of overhead.

With TDE, you can specify different encryption algorithms to be used at the table or tablespace level. The available algorithms are 3DES168, AES128, AES192, and AES256. The default is AES192 for column encryption and AES128 for tablespace encryption.

Using TDE



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

TDE enables encryption for sensitive data in columns without requiring users or applications to manage the encryption key. This freedom can be extremely important when addressing, for example, regulatory compliance issues. There is no need to use views to decrypt data because the data is transparently decrypted when a user has passed necessary access control checks. Security administrators have the assurance that the data on disk is encrypted, yet handling encrypted data is transparent to applications.

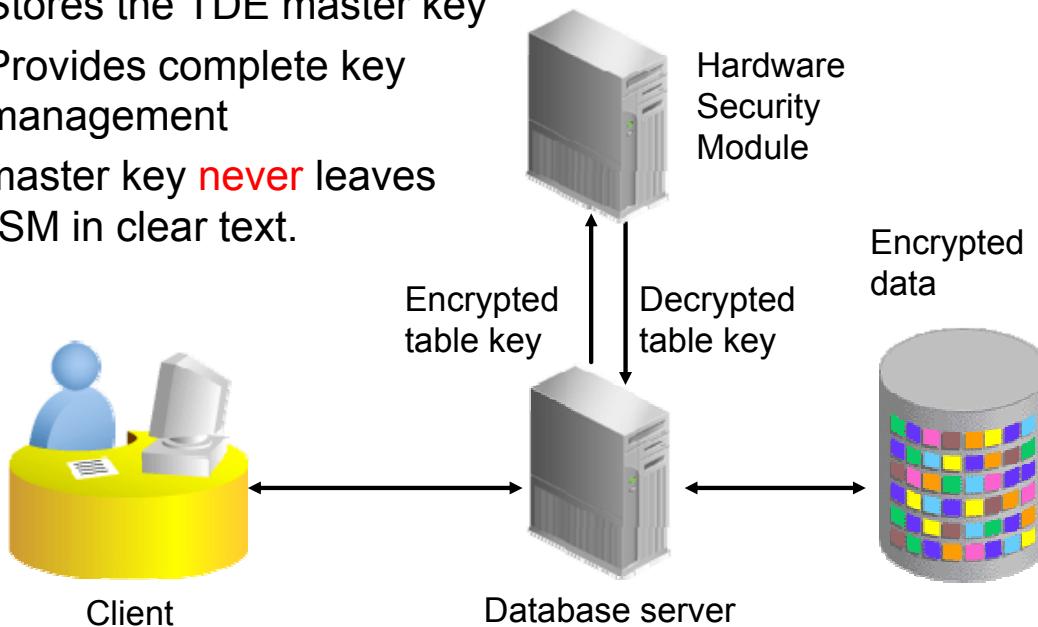
The external security module is implemented through an API that allows a variety of possible key storage solutions. The default external security module is a software keystore. Hardware Security Modules (HSM) from several vendors are also supported for storage of the master keys. TDE support of HSM varies by database version and whether column encryption or tablespace encryption is being used.

Using Hardware Security Modules

HSM:

- Stores the TDE master key
- Provides complete key management

The master key **never** leaves the HSM in clear text.



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

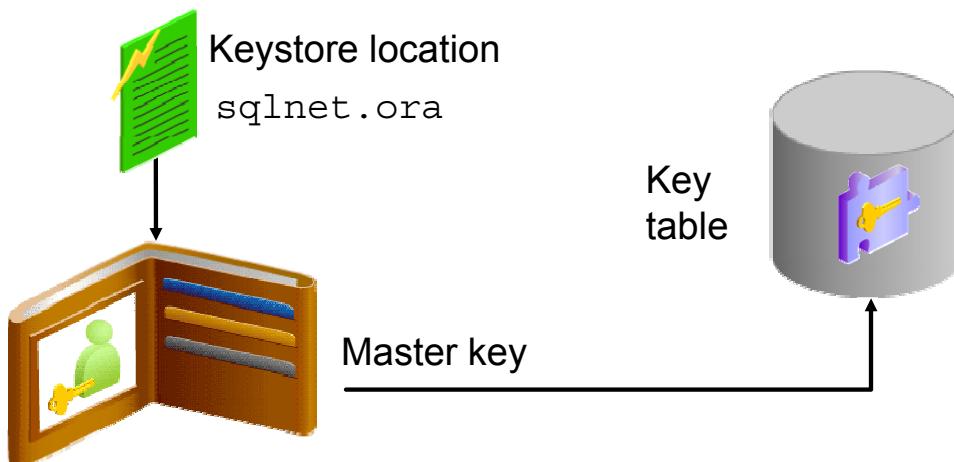
A Hardware Security Module (HSM) is a physical device that provides secure storage for encryption keys. It also provides secure computational space (memory) to perform encryption and decryption operations. HSM is a more secure alternative to the software keystore.

Transparent Data Encryption (TDE) can use an HSM to provide enhanced security for sensitive data. An HSM is used to store the master encryption key used for TDE. The key is secure from unauthorized access attempts because the HSM is a physical device and not an operating system file. All encryption and decryption operations that use the master encryption key are performed inside the HSM. This means that the master encryption key is never exposed in nonsecure memory.

There are several vendors that provide Hardware Security Modules. The vendor must also supply the appropriate libraries.

Defining the Keystore Location

```
ENCRYPTION_WALLET_LOCATION =  
(SOURCE =  
(METHOD = FILE)  
(METHOD_DATA =  
(DIRECTORY =  
/u01/app/oracle/admin/orcl/wallet)))
```



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

TDE creates a key for each table that uses encrypted columns and each encrypted tablespace. The table key is stored in the data dictionary and the tablespace keys are stored in the tablespace data files. Both tablespace and table keys are encrypted with a master key. There is one master key for the database. The master key is stored in a PKCS#12 software keystore or a PKCS#11-based HSM, outside the database. For the database to use TDE, a keystore must exist.

Use the following procedure to create a software keystore and a master key.

1. Create a directory to hold the keystore, which is accessible to the Oracle software owner.
2. Specify the location of the keystore file used to store the encryption master key by adding an entry in the \$ORACLE_HOME/network/admin/sqlnet.ora file as shown in the example of the slide.

When determining which keystore to use, Oracle Database searches for the keystore location in the following places, in this order:

- First, it attempts to use the keystore in the location specified by the parameter `ENCRYPTION_WALLET_LOCATION` in the `sqlnet.ora` file.
 - If the `ENCRYPTION_WALLET_LOCATION` parameter is not set, then it attempts to use the keystore in the location that is specified by the parameter `WALLET_LOCATION`.
 - If the `WALLET_LOCATION` parameter is also not set, then Oracle Database looks for a keystore at the default database location, which is
`$ORACLE_BASE/admin/DB_UNIQUE_NAME/wallet` or
`$ORACLE_HOME/admin/DB_UNIQUE_NAME/wallet`. (`DB_UNIQUE_NAME` is the unique name of the database specified in the initialization parameter 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.
3. Log in to the database instance as a user who is granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
 4. Create the software keystore file.
 5. Open the software keystore file.
 6. Create the master encryption key. It will be stored in the software keystore.

Creating and Opening the Keystore

```
SQL> CONNECT / AS SYSKM
SQL> ADMINISTER KEY MANAGEMENT CREATE KEYSTORE
      '/u01/app/oracle/admin/orcl/wallet'
      IDENTIFIED BY keystore_password;
```

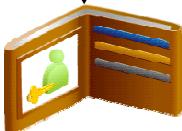


3 types of software keystores

- >Password-based
- Auto-login
- Local auto-login

- To be opened before the keys can be retrieved or used
- Does not need to be explicitly opened
- Cannot be opened on any computer other than the one on which they are created

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



/u01/app/oracle/admin/orcl/wallet/ewallet.p12

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

To create and open the keystore, log in to the database instance as a user who is granted the ADMINISTER KEY MANAGEMENT or SYSKM privilege.

Before column values can be encrypted or encrypted columns can be viewed by a user, the keystore must be created and opened as explained in the slide. The `keystore_password` is the password of the keystore that the security administrator creates. The keystore created is the `ewallet.p12` file stored in the location defined in the statement.

Refer to the *Database Advanced Security Guide 12c Release 1 (12.1)* to get detailed information on the types of keystores you can create. TDE uses an auto login keystore only if it is available at the correct location (`ENCRYPTION_WALLET_LOCATION`, `WALLET_LOCATION`, or the default keystore location), and the SQL statement to open an encrypted keystore has not already been executed.

If the keystore is not opened and the user attempts to access an encrypted column, an error message is generated as shown in the following example:

```
SQL> select * from cust_payment_info;
*
ERROR at line 1:
ORA-28365: wallet is not open
```

Generating the Master Key

Generating the master key:

- Creates a new master key in the keystore
- Backs up the new keystore

```
SQL> ADMINISTER KEY MANAGEMENT SET KEY  
      IDENTIFIED BY keystore_password  
      WITH BACKUP  
      USING 'for_12c' ;
```

Regenerating the master key:

- Backs up the current keystore
- Creates a new master key in the keystore
- Keeps retired master keys



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Once the keystore is created and opened, you can set the master encryption key. The master encryption key is stored encrypted in the keystore file. Use the `WITH BACKUP` clause to backup the keystore. You must use this option for password-based keystores. Optionally, you can use the `USING` clause to add a brief description of the backup. This identifier is appended to the named keystore file.

In a normal operation, you need to regenerate the master key only if it has been compromised. Changing the master periodically may be required by regulation. Change the master key with the command of the slide where `keystore_password` is the keystore password. The master key is generated using a random number generator.

Regenerating the master key does not cause the data to be reencrypted. The master key is used to encrypt table keys, used for column encryption, and tablespace keys. The table keys are used to encrypt column data. Tablespace keys are used to encrypt tablespace blocks. Changing the master key will cause the table and tablespace keys to be re-encrypted, which is a relatively quick operation, but the column data and the tablespace blocks are not reencrypted.

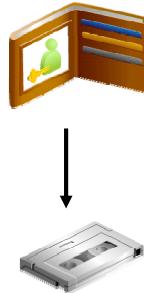
All past master keys are held in the keystore, and the prior keys are available if the old data is recovered from a backup or if the database is recovered to a point in time before the key was regenerated.

Note: If the master key is regenerated, offline tablespaces will be re-keyed the next time they are brought online.

Backing Up the Keystore

Backing up the current keystore:

```
SQL> ADMINISTER KEY MANAGEMENT BACKUP KEYSTORE  
IDENTIFIED BY keystore_password;
```



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The master keys are required to access encrypted data and you must protect these keys with backups. Because master keys reside in a keystore, the keystore should be periodically backed up in a secure location along with the database data files. You must back up a copy of the encryption keystore whenever a new master key is set.

If you lose the keystore that stores the master key, you can restore access to encrypted data by copying the backed-up version of the keystore to the appropriate location. If the restored keystore was archived after the last time the master key was reset, no additional action needs to be taken.

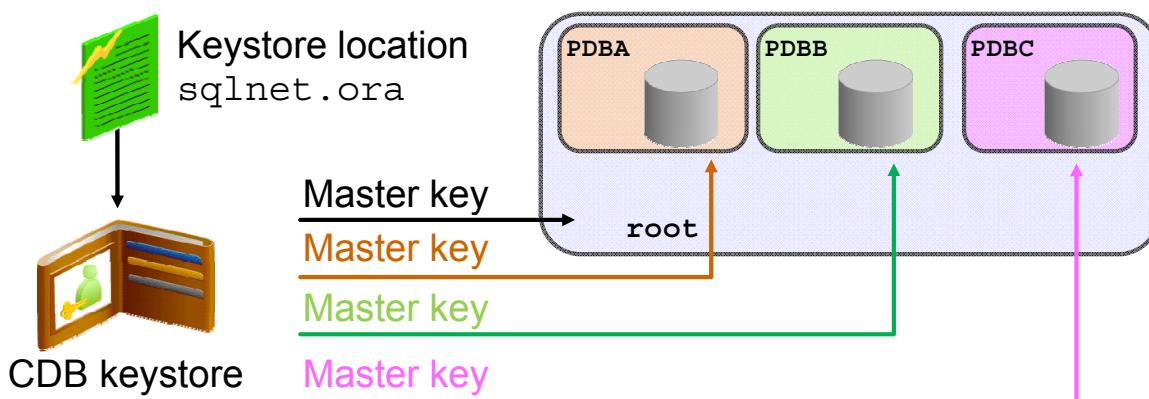
If the restored keystore does not contain the most recent master key, you can recover old data up to the point when the master key was reset by rolling back the state of the database to that point in time. All modifications to encrypted columns after the master key was reset are lost.

There can be two keystores present whenever the keystore is open: the password-based keystore identified with the `p12` extension, and an auto-login `cwallet.sso` keystore. The auto-login keystore is changed every time the keystore is opened, so it is not useful to include it in backups. The password-based keystore holds current and past master keys. It must be included in backups.

There are separate and distinct keystores used for Recovery Manager (RMAN) and Oracle Secure Backup (OSB) encryption.

Managing the Keystore in CDB and PDBs

- There is one master key per PDB to encrypt PDB data.
- The master key must be transported from the source database keystore to the target database keystore when a PDB is moved from one host to another.



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In a multitenant container database (CDB), the root container and each pluggable database (PDB) has its own master key used to encrypt data in the PDB. The master key must be transported from the source database keystore to the target database keystore when a PDB is moved from one host to another.

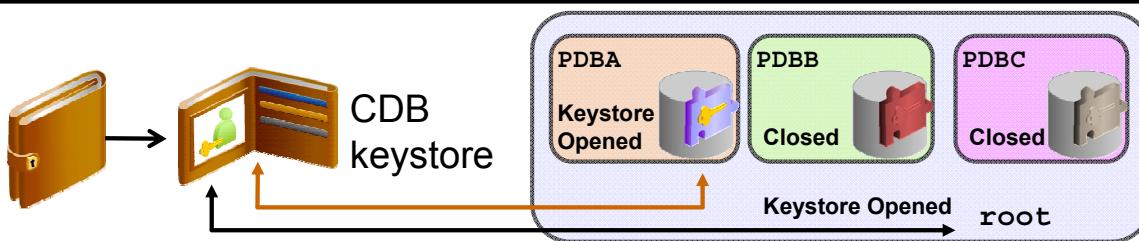
Creating and Opening the Keystore

1. Create the unique keystore in the root.

```
SQL> ADMINISTER KEY MANAGEMENT CREATE KEYSTORE
  2  'u01/app/oracle/product/12.1.0/dbhome_1/wallet'
  3 IDENTIFIED BY keystore_password;
```

2. Open the keystore in the root and then for a specific PDB.

```
SQL> CONNECT john@PDBA AS SYSKM
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE
  2 OPEN IDENTIFIED BY keystore_password
  3 CONTAINER = CURRENT;
```



ORACLE

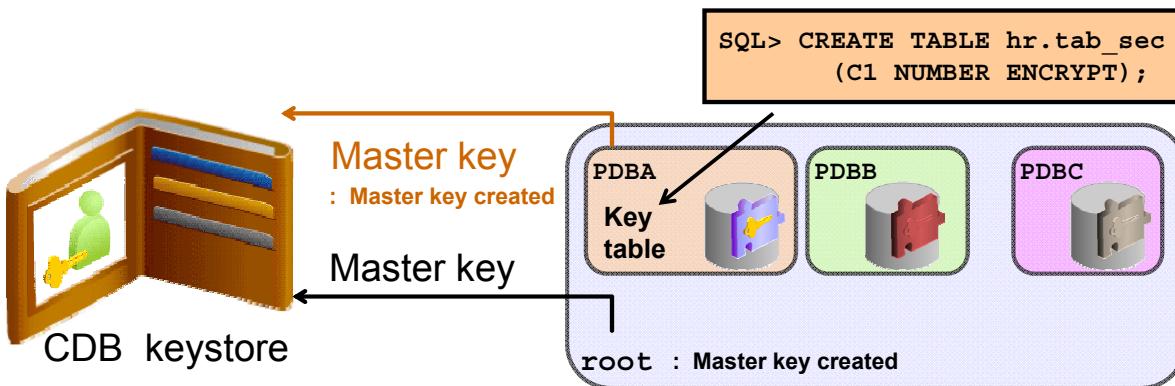
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

1. Create the keystore from the root for the CDB using the first command in the slide.
2. Open the keystore:
 - First for the root: Use CONTAINER=ALL to open the keystore for all PDBs including for the root container or without the CONTAINER clause to open the keystore at least for the root.
 - Then for a specific PDB: Use CONTAINER=CURRENT

Setting Master Encryption Keys

3. Set the master encryption key for a PDB

```
SQL> CONNECT john@PDBA AS SYSKM
SQL> ADMINISTER KEY MANAGEMENT SET KEY
  2 IDENTIFIED BY keystore_password WITH BACKUP
  3 CONTAINER = CURRENT;
```



You can now encrypt data in tablespaces and tables.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

3. Set the master key:

- First for the root: Use `CONTAINER=ALL` to set a master key for each PDB and for the root container or without the `CONTAINER` clause to set a master key at least for the root.
- Then for a specific PDB: Use `CONTAINER=CURRENT`

Quiz

To use Transparent Data Encryption, you must create a separate master key for table and tablespace encryption.

- a. True
- b. False



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: b

Re-Keying Table Keys

- Re-keying the individual table keys is an update operation because all encrypted values are decrypted and reencrypted.
- Usually, re-keying the master is sufficient to achieve Payment Card Industry compliance.
- Re-keying does not impact the availability of your database.

```
SQL> ALTER TABLE card_payment_info REKEY;  
  
SQL> ALTER TABLE employee REKEY USING '3DES168';
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The two examples in the slide generate a new key for the table. The first generates a new key based on the algorithm that was specified when the table columns were encrypted. The second generates a new key and changes the algorithm. Both of these examples cause all the encrypted data in the tables to be decrypted, and updated with a new encrypted value.

Note: There is only one key and one algorithm per table, even if multiple columns are encrypted in the table.

Creating an Encrypted Column

- To create an encrypted column, use the ENCRYPT attribute when the table is created or altered.

```
SQL> CREATE TABLE cust_info_salt
      (name VARCHAR2(11), order_number NUMBER(13),
       credit_card_number VARCHAR2(20)
       ENCRYPT SALT );
```

- Use a specific algorithm, a password, the NOMAC function.

```
SQL> CREATE TABLE cust_payment_info
      (... ccn VARCHAR2(20) ENCRYPT USING 'AES256'
           [IDENTIFIED BY password]
           [NO SALT] ['NOMAC']);
```

- Remove the encryption on columns.

```
SQL> ALTER TABLE cust_payment_info MODIFY ccn DECRYPT;
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In the example in the slide, the SALT keyword used does not allow an index to be created on this column. The default is SALT. SALT is a string that is added to the data before it is encrypted, so that identical values in the column have different encrypted values. It is not possible to create an index on a column that has SALT .

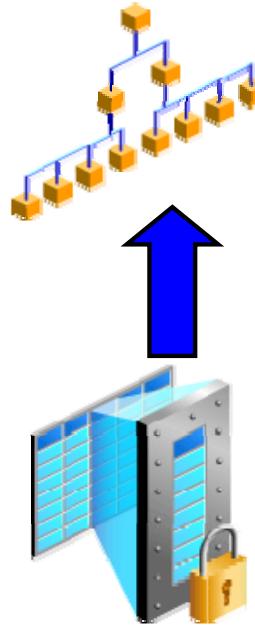
The encrypt clause allows you to specify the encryption algorithm to use. The name of an algorithm implicitly determines the key length. The valid algorithm names are 3DES168, AES128, AES192 (default), AES256.

The IDENTIFIED BY <password> clause is optional. Specifying a password means that the key used to protect the table will be based on that password. The user does not have to remember the password, but can use that same password on another table if necessary—for example, for a partitioned table that needs the same key shared across table partitions.

The NOMAC parameter enables you to skip the integrity check performed by TDE. This saves 20 bytes of disk space per encrypted value.

If a column is encrypted using the ALTER TABLE...MODIFY...ENCRYPT command, the rows are updated. The **unencrypted data will remain in the data blocks** until the space for the original version of the rows is reclaimed. The ALTER TABLE...MOVE command will move only the current encrypted rows to a new location.

Creating an Index on an Encrypted Column



ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

A B-tree index can be created on an encrypted column with NO_SALT. A B-tree may not be created on a column with SALT. Equality lookup operations are supported on the index.

A bitmapped index cannot be created on encrypted columns.

TDE column encryption is not supported on foreign keys. This is because each table has its own encryption key. For this reason, do not use sensitive data items such as credit card number or national identity number as a primary key.

Index range-scan operations are supported for equality lookups because the value is encrypted before the comparison to the stored values. WHERE clauses with BETWEEN...AND or LIKE comparison operators use full-table scans.

Note: Tablespace-level TDE supports all index types, all internal data types, and foreign keys.

TDE Column Encryption Support

- TDE column encryption supports:
 - Most scalar data types
 - Data Guard
 - Physical standby database configuration in 10g Release 2
 - Logical standby database configuration and Streams in 11g
 - Direct path SQL*Loader
- TDE column encryption does not support:
 - Materialized view logs
 - Synchronous change data capture (CDC)
 - Transportable tablespaces
 - Foreign keys
 - External large objects (BFILE data type)



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

TDE column encryption performs the encryption and decryption operations in the SQL layer, so that the data remains encrypted in the database buffer cache, undo, and redo memory, and associated files.

Some Oracle database features, most of them related to data warehouse technologies, bypass the SQL layer for better performance when moving large amounts of data between tables. These features are not supported with TDE column encryption. TDE column encryption is not supported at this time when used with materialized view logs, which keep track of changes to a master table in order to update the materialized view. Furthermore, using TDE with synchronous change data capture (CDC), BFILE data types, and transportable tablespaces is unsupported.

TDE column encryption can be used with most scalar data types. The following data types can be encrypted:

- CHAR
- DATE
- INTERVAL DAY TO SECOND
- INTERVAL YEAR TO MONTH
- NCHAR
- NUMBER

- NVARCHAR2
- RAW
- TIMESTAMP (includes TIMESTAMP WITH TIME ZONE and TIMESTAMP WITH LOCAL TIME ZONE)
- VARCHAR2 (must be less than or equal to 3,932 bytes)
- CLOB (SecureFiles)
- BLOB (SecureFiles)

TDE column encryption supports Oracle Data Guard in the physical standby configuration. To use TDE with Data Guard, both primary and standby databases must be of the same version. You have two choices in terms of security at the standby sites:

- Make a file copy of the encrypted keystore on the primary site and ship it to the standby site. In this case, the keystore in the standby site needs to be opened by a DBA before the databases can process encrypted data in case of a failover.
- Use an auto-login keystore that you create from the password-based keystore on the primary site and ship it to the standby site. To create an auto-login keystore from a password-based keystore , use:

```
mkwallet -s pwd wrl
```

- where pwd is the password for both keystores and wrl is the directory where the new auto-login keystore is to be stored. The file name of this new obfuscated, auto-login keystore is cwallet.sso.

In both cases, each time, the master key on the primary site is changed, the keystores must be shipped to all standby sites.

TDE column encryption supports Data Guard logical standby database configuration. The logs may be mined either on the source or target database; thus, the keystore must be the same for both databases. Every time, the master key is changed, the keystore must be copied from the primary database to the standby database. An error is raised if the DBA attempts to change the master key on the standby database.

If auto-login keystore is not used, the keystore must be opened on the standby database. Keystore open and close commands on the primary database are not replicated on the standby database. A different password can be used to open the keystore on the standby database. The keystore owner can change the password to be used for the copy of the keystore on the standby database.

TDE Column-Level Storage Requirements

- SALT (16 bytes):
 - Is recommended when clear text values are not unique, to make sure that cipher text strings are unique
 - Cannot be added to columns that are indexed
 - Message Authentication Code (MAC) (20 bytes)
 - MAC is a hash value computed over the cipher text to detect tampering or corruption.
 - NOMAC neither computes nor stores this value.
 - Each clear text value is padded to the next full 16 byte.
- Examples:
- 9 bytes + 7 bytes padding = 16 bytes
 - 49 bytes + 15 bytes padding = 64 bytes



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The storage overhead associated with TDE column encryption can be significant. When specified, SALT requires 16 bytes. Specifying NO SALT reduces storage requirement and saves Critical Patch Update (CPU) cycles. Message Authentication Code (MAC), an integrity check associated with each encrypted value, requires an additional 20 bytes. In addition, TDE will pad out encrypted values to a multiple of 16 bytes, so if a credit card number required 9 bytes of storage, encrypting the credit card number would require an additional 7 bytes of storage. In summary, encrypting a single column in a table with SALT will require between 37 and 52 bytes of additional storage per row.

SALT is not needed if the clear text values are unique, and SALT cannot be used with columns that will be indexed. The NOMAC parameter enables you to skip the integrity check performed by TDE. This saves 20 bytes of disk space per encrypted value. If the number of rows and encrypted columns in the table is large, this adds up to a significant amount of disk space. The NOMAC parameter also reduces the performance overhead. The NOMAC parameter is applied to all the columns of a table. If one column uses NOMAC, they all must use the NOMAC option.

In summary, a customer encrypting a single column using both NO SALT and NOMAC parameters can reduce the encryption overhead to between 1 and 15 bytes per row of additional storage, instead of 37 to 52 bytes.

TDE Column Encryption: Restrictions

You cannot use TDE column encryption:

- For bitmapped or domain indexes on encrypted columns
- On columns used in foreign key constraints
- On `SYS` schema objects
- If range scan search through an index is required
- In synchronous change data capture
- For transportable tablespaces

LOB data types

- Internal LOBs can be encrypted.
- External LOBs (`BFILE` data type) cannot be encrypted.

Note: External tables may have encrypted columns with the `ORACLE_DATAPUMP` access driver.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

TDE column encryption cannot be used with the database features described in the slide.

Internal LOB data types (such as `BLOB` and `CLOB`) can be encrypted, but external LOBs (such as binary large file objects [`BFILE` data type]) cannot be encrypted.

Note: External tables can have encrypted columns using the `ORACLE_DATAPUMP` access driver.

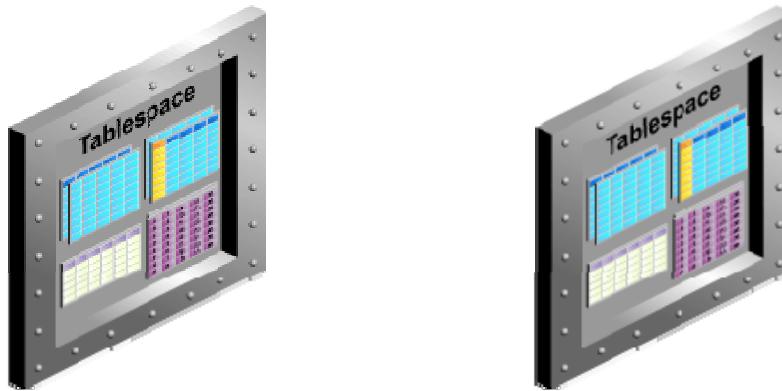
Applications that need to use these unsupported features can use the TDE tablespace encryption.

Note: TDE tablespace encryption supports all data types, except external table and `BFILE`. The `SYSTEM` tablespace cannot be encrypted.

Tablespace Encryption: Advantages

Tablespace encryption provides the following advantages over column-based transparent data encryption:

- Encrypts all data stored in the tablespace, including LOBs
- Allows index range-scans on data in the tablespace
- Supports all data types



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

TDE tablespace encryption is performed in the I/O level on a per-block basis. The only encryption penalty is associated with I/O, so the performance overhead will be seen in the I/O statistics. When there is a large number of columns in a table to be encrypted, tablespace encryption may provide better performance than column encryption.

The SQL access paths are unchanged and all data types are supported. Because the data is not encrypted in memory, there is no difference in the handling of data when it is read off disk. All data types, index types, and even LOBs are supported with tablespace encryption.

Data retrieved from encrypted tablespaces is protected whenever it is written to disk, including temporary tablespaces, undo tablespace, and redo logs. During operations such as JOIN and SORT, data that is selected from an encrypted tablespace is encrypted when written to temporary tablespaces.

Encrypted tablespaces are transportable if the platforms have the same endianess and the same keystore.

There is currently no mechanism to re-key a tablespace.

Tablespace encryption does not require additional storage space.

Creating an Encrypted Tablespace

- Create or open the encryption wallet:
- Create a tablespace with the encryption keywords:

```
SQL> CREATE TABLESPACE encrypt_ts  
      DATAFILE '$ORACLE_HOME/dbs/encrypt.dat' SIZE 100M  
      ENCRYPTION USING '3DES168'  
      DEFAULT STORAGE (ENCRYPT);
```

Restrictions:

- Temporary and undo tablespaces cannot be encrypted.
- BFILE data type and external tables are not encrypted.
- The key for an encrypted tablespace cannot be changed.
- The SYSTEM tablespace cannot be encrypted.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The CREATE TABLESPACE command has an ENCRYPTION clause that sets the encryption properties, and an ENCRYPT storage parameter that causes the encryption to be used. You specify USING 'encrypt_algorithm' to indicate the name of the algorithm to be used. Valid algorithms are 3DES168, AES128, AES192, and AES256. The default is AES128. You can view the properties in the V\$ENCRYPTED_TABLESPACES view.

Because tablespace encryption is performed at the I/O level, many of the restrictions that apply to TDE column encryption do not apply to tablespace encryption.

The following restrictions apply to tablespace encryption:

- Temporary and undo tablespaces cannot be encrypted. But when a data buffer containing data from an encrypted tablespace is written to an undo or temporary tablespace, that data block is encrypted.
- BFILE data type and external tables are not encrypted because they are not stored in tablespaces.
- Transportable tablespaces across different endian platforms are not supported.
- The key for encrypted tablespaces cannot be changed. A workaround is to create a tablespace with the desired properties and move all objects to the new tablespace.

Encrpyted tablespace work on active Data Guard, snapshot standby and also logical standby databases.

SECUREFILE LOB Encryption

LOB encryption:

- Is allowed only for SECUREFILE LOBs
- Encrypts all LOBs in the LOB column
- Can be encrypted only per column
- Allows for the coexistence of SECUREFILE and BASICFILE LOBs

```
SQL> CREATE TABLE test1
      (doc CLOB ENCRYPT USING 'AES128')
      LOB(doc) STORE AS SECUREFILE (CACHE NOLOGGING);
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Oracle Database 11g introduced SecureFiles implementation (of LOBs), which offers intelligent compression and transparent encryption. The encrypted data in SecureFiles is stored in place and is available for random reads and writes. The encryption takes place at the block level. If you add a LOB column to a table, you can specify how it should be created using SECUREFILE or BASICFILE keywords. By default, it is created as SECUREFILE in Oracle Database 12c.

To enable encryption of LOBs, you must create the LOB with the SECUREFILE keyword, with encryption enabled (ENCRYPT) or disabled (DECRYPT, which is the default) on the LOB column. The current TDE syntax is used for extending encryption to LOB data types.

Valid encryption algorithms are 3DES168, AES128, AES192, and AES256. The default is AES192.

Summary

In this lesson, you should have learned how to:

- Set up the database master encryption key
- Encrypt column data
- Encrypt tablespace data



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Practice: Overview

This practice covers the following topics:

- 19-1: Configuring the password-based keystore
- 19-2: Implementing table column encryption
- 19-3: Implementing tablespace encryption



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

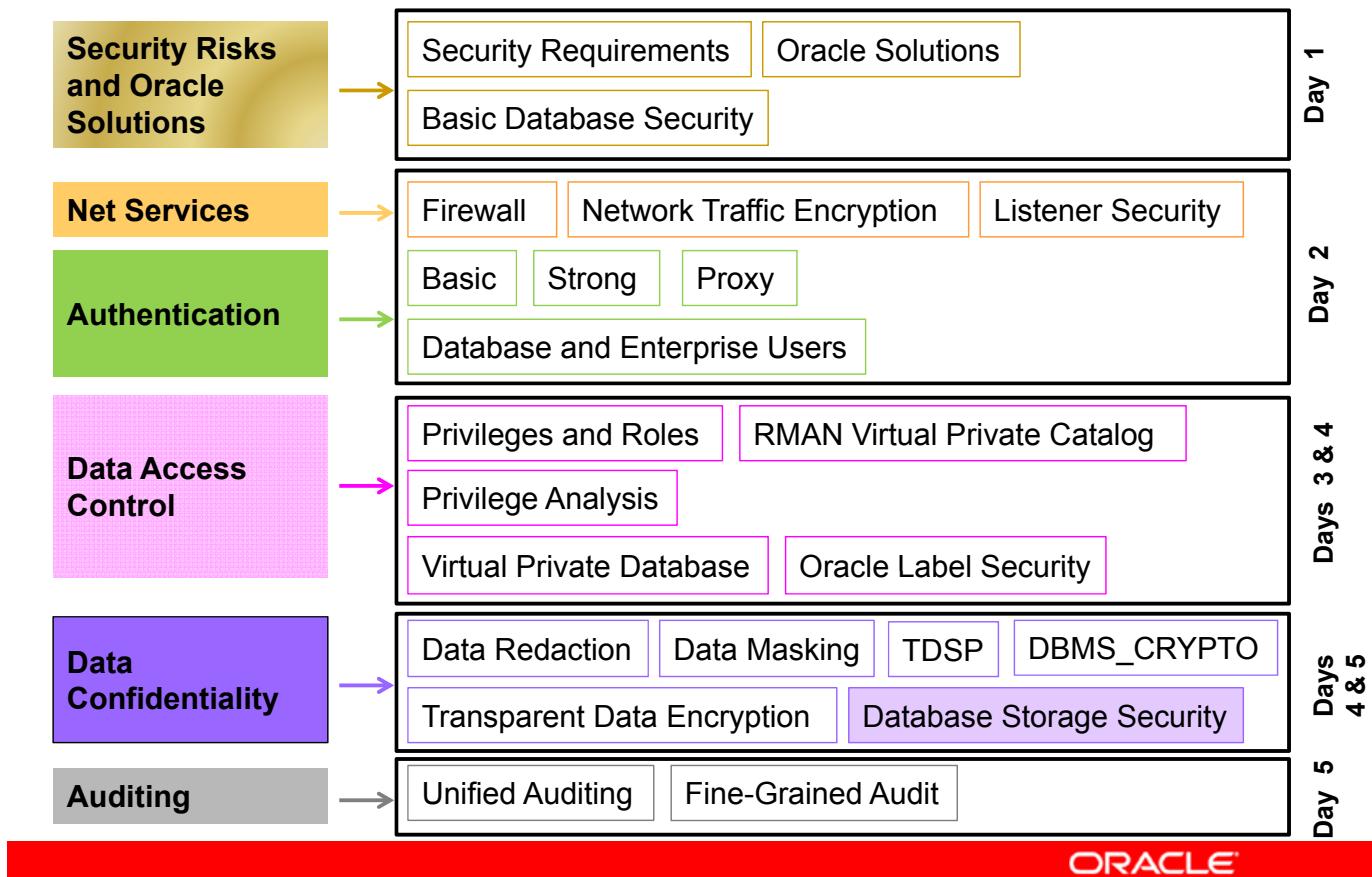
20

Database Storage Encryption

ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Suggested Schedule



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Objectives

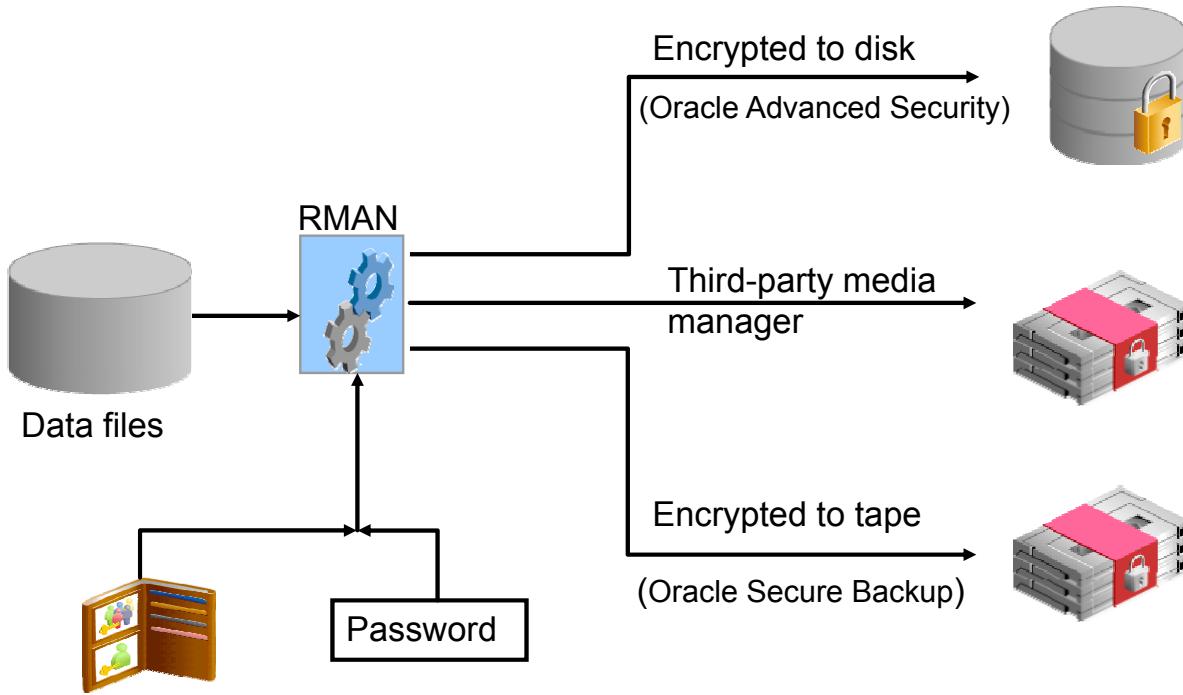
After completing this lesson, you should be able to use the following:

- Recovery Manager (RMAN)—encrypted backups
- Oracle Secure Backup
- Data Pump Export encryption



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

RMAN-Encrypted Backups



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

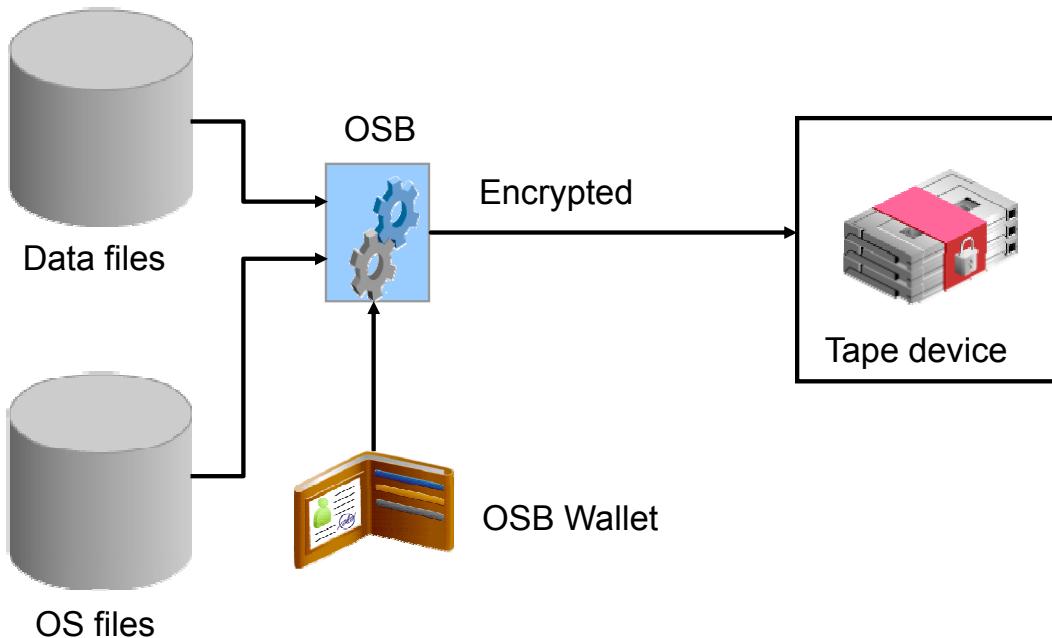
Recovery Manager (RMAN) can create encrypted backups to either tape or disk as long as the required key management infrastructure is available. RMAN encryption can use either a password-based key or a generated key held in the keystore.

The data is encrypted by RMAN before it is transmitted to the disk or tape storage device, and no further encryption is performed.

RMAN backup encryption is available only in the Enterprise Edition of the database, and the COMPATIBLE parameter must be set to 10.2.0 or higher.

Oracle Advanced Security is required for RMAN-encrypted disk backups. Encrypted backups to tape require Oracle Secure Backup to provide the key infrastructure.

Oracle Secure Backup Encryption



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Oracle Secure Backup (OSB) is available in both Standard Edition and Enterprise Edition of Oracle Database 12c. OSB includes the secure communications technology of Oracle Advanced Security in the Enterprise Edition to provide secure communication between hosts (administrative, source, and target) in the OSB domain. OSB encrypts the transmitted data and control messages with a default key of 1,024 bits generated for each session using secure sockets layer (SSL). OSB provides this key from an embedded wallet that is separate from the keystore used by RMAN to encrypt backups. If RMAN encryption is provided, OSB does not encrypt the data again for transmission. But if RMAN encryption is disabled, and the OSB host encryption policy is set to "required," the OSB encryption will be used for the data; if the OSB encryption policy is set to "allowed," in principal, the decision is referred to the next lower level.

You can modify the default security configuration in the following ways:

- Disable SSL for interhost authentication and communication by setting the `securecomms` security policy in Oracle Secure Backup.
- Transmit identity certificates in manual certificate provisioning mode.
- Set the key size for a host to a value from 512 to 4,096 bits, rather than the default of 1,024 bits.

- Disable encryption for backup data in transit by setting the `encryptdataintransit` security policy.

Because Oracle Secure Backup–embedded wallets are used only for interdomain communication, they do not have any direct relationship with the backup data written to tape. Therefore, if wallets are destroyed and re-created, it does not affect the restoration of data from tape.

OSB does not share its wallets with other Oracle products.

Besides maintaining its password-protected wallet, each host in the domain maintains an obfuscated wallet. This version of the wallet does not require a password. The obfuscated wallet is created when the wallet is opened and destroyed when the wallet is closed. This wallet, which is scrambled but not encrypted, enables the OSB software to run without requiring a password during system startup.

The password for the password-protected wallet is generated by OSB and not made available to the user. The password-protected wallet is not normally used after the security credentials for the host have been established because the OSB daemons use the obfuscated wallet.

To reduce the risk of unauthorized access to obfuscated wallets, OSB does not back them up. The obfuscated version of a wallet is named `cwallet.sso`. By default, the wallet is located in `/usr/etc/ob/wallet` on Linux and UNIX and `C:\Program Files\Oracle\Backup\db\wallet` on Windows.

Best practice tip: Back up the OSB-encrypted wallet; do not back up the obfuscated wallet.

Encrypted Backups to Tape

- RMAN backup encryption:
 - Encryption keys transparently managed by the database
 - Backup encryption at the database or tablespace level
 - Encryption algorithms up to 256-bit AES
- Secure transportation over the network:
 - Database backups with RMAN encryption
 - File-system backups with secure sockets layer (SSL)
- RMAN-encrypted backups to tape is available with:
 - Oracle Secure Backup
 - Third-party media managers



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

OSB leverages RMAN backup encryption technology, such as:

- Encryption keys being transparently managed by the database
- Your ability to choose backup encryption at the database or tablespace level (This is in addition to the Transparent Data Encryption (TDE), which you can use inside the Oracle database.)
- Substantial protection through encryption algorithms up to 256-bit AES

During transportation over the network, database backups are secured with RMAN encryption (in which case, no additional SSL is used). If your database backups are not encrypted by RMAN, Oracle Secure Backup uses SSL by default. It also secures your file-system backups over the network by using SSL.

RMAN can create encrypted backups on tape using OSB or a third-party media manager with Oracle Advanced Security.

Creating RMAN-Encrypted Backups

RMAN supports three encryption modes:

- Transparent mode:
 - Uses the Oracle key management infrastructure
 - Requires that you first configure the keystore
- Password mode: Requires the use of the SET ENCRYPTION ON IDENTIFIED BY password ONLY command in your RMAN scripts
- Dual mode: Requires the use of the SET ENCRYPTION ON IDENTIFIED BY password command in your RMAN scripts



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

For improved security, RMAN backups created as backup sets can be encrypted. Image copy backups cannot be encrypted.

Encrypted backups are decrypted automatically during restore and recover operations, as long as the required decryption keys are available, by means of either a user-supplied password or the keystore.

RMAN supports three encryption modes:

- Transparent mode
- Password mode
- Dual mode

Additional information about each mode follows.

Using Transparent-Mode Encryption

1. Identify the keystore location:

```
ENCRYPTION_WALLET_LOCATION= (SOURCE=(METHOD=FILE) (METHOD_DATA=
(DIRECTORY=/opt/oracle/product/11.2.0/dbhome_1)))
```

2. Create the keystore:

```
SQL> CONNECT / AS SYSKM
SQL> ADMINISTER KEY MANAGEMENT CREATE KEYSTORE
      'u01/app/oracle/product/12.1.0/dbhome_1/wallet'
      IDENTIFIED BY keystore_password;
```

3. Open the keystore:

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

4. Set the master key:

```
SQL> ADMINISTER KEY MANAGEMENT SET KEY
      IDENTIFIED BY keystore_password WITH BACKUP;
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Transparent data encryption does not require DBA intervention as long as the required Oracle key management infrastructure is available. Transparent data encryption is best suited for day-to-day backup operations, where backups will be restored on the same database that they were backed up from. Transparent encryption is the default encryption mode.

You must first configure the keystore to use transparent encryption. Refer to the *Oracle Database Advanced Security Guide* for detailed information about the keystore.

Perform the following steps to use transparent mode encryption:

1. By default, an unencrypted keystore (`cwallet.sso`) is created when Oracle Database is installed. An encrypted keystore (`ewallet.p12`) is recommended for use with backup set encryption. Place an entry in the `sqlnet.ora` file as shown in the slide.
2. Create the keystore.
3. Before you can use backup set encryption, you need to make sure that the keystore is opened by your instance. The password specified with the `ALTER SYSTEM` command is the same password you specified when you created the keystore in step 1.
4. Set the master key from within your instance. When the keystore is opened, you need to set the master key.

Using Transparent-Mode Encryption

5. Configure RMAN encryption level (database, tablespace, or database excluding tablespaces):

```
RMAN> CONFIGURE ENCRYPTION FOR DATABASE ON
```

```
RMAN> CONFIGURE ENCRYPTION FOR TABLESPACE  
<tablespace_name> ON
```

6. Set encryption algorithm, if needed:

```
RMAN> SET ENCRYPTION ALGORITHM 'algorithm name'
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

5. Configure the RMAN encryption level. The `CONFIGURE ENCRYPTION` command is used to specify encryption settings for the database or tablespaces within the database, which apply unless overridden using the `SET` command. Options specified for an individual tablespace take precedence over options specified for the whole database.
6. Set an encryption algorithm, if needed. Query `V$RMAN_ENCRYPTION_ALGORITHMS` to obtain a list of encryption algorithms supported by RMAN. The default encryption algorithm is 128-bit AES.

Using Password-Mode Encryption

Enable password mode encryption in your RMAN session:

```
SET ENCRYPTION ON IDENTIFIED BY <password> ONLY
```



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

When you use password encryption, you must provide a password when you create and restore encrypted backups. When you restore the password-encrypted backup, you must supply the same password that was used to create the backup. Password encryption is most appropriate for backups that will be restored at remote locations, but which must remain secure in transit.

Use the `SET ENCRYPTION ON IDENTIFIED BY password ONLY` command in your RMAN scripts to enable password encryption. Password encryption cannot be persistently configured.

The Enterprise Manager interface will place the proper command in the RMAN backup scripts that it generates.

Note: For security reasons, it is not possible to permanently modify your existing backup environment so that RMAN backups are encrypted by using password mode. You can enable only password-encrypted backups for the duration of an RMAN session.

Using Dual-Mode Encryption

- Dual-mode encrypted backups can be restored transparently or by specifying a password.
- Enable password mode encryption in your RMAN session:

```
SET ENCRYPTION ON IDENTIFIED BY 'password'
```



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Dual-mode encrypted backups can be restored transparently or by specifying a password. Dual-mode encrypted backups are useful when you create backups that are normally restored using the keystore, but which occasionally need to be restored where the keystore is not available.

To create dual-mode encrypted backup sets, specify the `SET ENCRYPTION ON IDENTIFIED BY password` command in your RMAN scripts.

Quiz

Which RMAN encryption techniques does not require enabling in an RMAN session?

- a. Dual-mode encryption
- b. Password-mode encryption
- c. Transparent-mode encryption



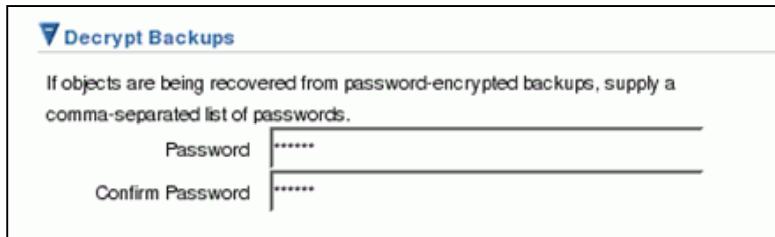
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: c

Restoring Encrypted Backups

- Before restoration, set the RMAN session to decrypt backups.
- Specify all required passwords with the SET DECRYPTION command when restoring from a set of backups that were created with different passwords.

```
SET DECRYPTION IDENTIFIED BY '<password_1>
{, '<password_2>', ..., '<password_n>' }
```



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Use the SET DECRYPTION command to specify one or more decryption passwords to be used when reading dual-mode or password-encrypted backups. When RMAN reads encrypted backup pieces, it tries each password in the list until it finds the correct one to decrypt that backup piece. An error is signaled if none of the specified keys are correct.

If you lose the password for a password-encrypted backup, you cannot restore that backup.

Because the Oracle key management infrastructure archives all previous master keys in the keystore, changing or resetting the current database master key does not affect your ability to restore encrypted backups performed using an older master key. You may reset the database master key at any time, but RMAN will always be able to restore all encrypted backups that were ever created by this database.

If you lose the keystore containing the key for a transparent encrypted backup, you cannot restore that backup. Because the keystore contains all past backup encryption keys, a restored keystore can be used to restore past encrypted backups up to the backup time of the keystore. Encrypted backups made after the keystore backup will be lost.

Best Practice Tip: Back up the Oracle keystore frequently.

RMAN-Encrypted Backups: Considerations

- Image copy backups cannot be encrypted.
- V\$RMAN_ENCRYPTION_ALGORITHMS contains the list of possible encryption algorithms.

```
RMAN> CONFIGURE ENCRYPTION ALGORITHM 'algorithmname'
```

```
RMAN> SET ENCRYPTION ALGORITHM 'algorithmname'
```

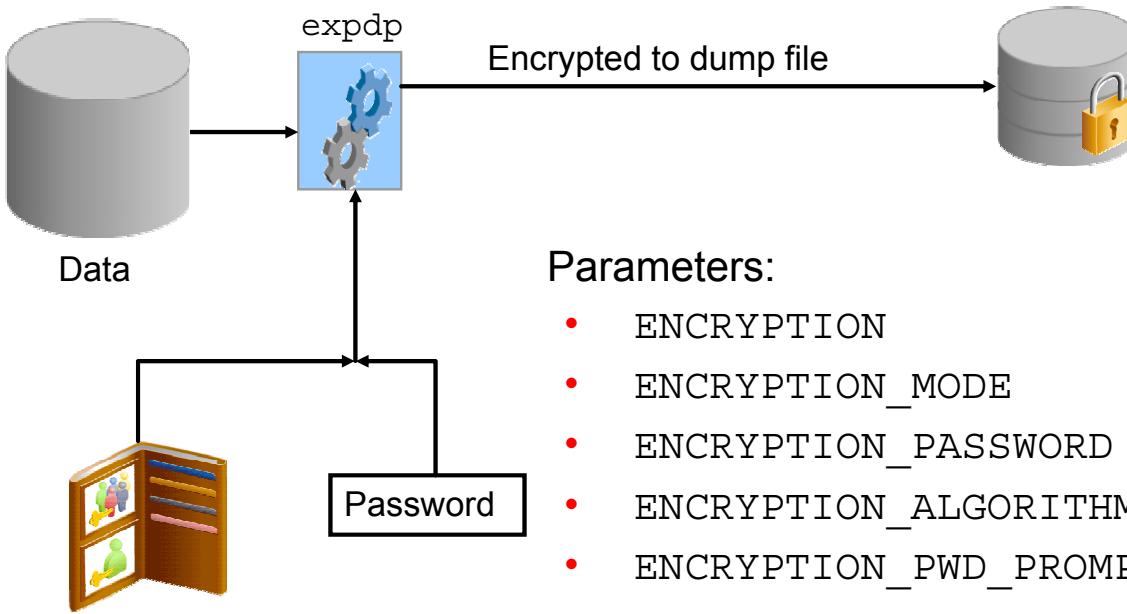
- One new encryption key is used for each new encrypted backup.
- You can increase disk performance by using multiple channels.
- You can change the master key any time without affecting your transparent encrypted backups.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

- Any RMAN backups created as backup sets can be encrypted. However, image copy backups cannot be encrypted.
- The V\$RMAN_ENCRYPTION_ALGORITHMS view contains a list of encryption algorithms supported by RMAN. If no encryption algorithm is specified, the default encryption algorithm is 128-bit AES. You can change the algorithm by using the commands shown in the slide.
- The Oracle Database server uses a new encryption key for every encrypted backup. The backup encryption key is then encrypted with either the password or the database master key, or both, depending on the chosen encryption mode. Individual backup encryption keys or passwords are never stored in clear text.
- Encryption can have a negative effect upon disk backup performance. Because encrypted backups use more CPU resource than nonencrypted backups, you can improve the performance of encrypted backups to disks by using more RMAN channels.

Data Pump Encryption



Note: TDE is supported only with Data Pump Export and Import.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Every file that could contain sensitive data should be protected in some way; the dump file produced by Data Pump Export is no exception. In Oracle Database 12c, Data Pump Export can encrypt the dump file. Data Pump file encryption requires Oracle Advanced Security.

The `expdp` process receives the data unencrypted from the database even if the data is encrypted in the database with TDE.

Note: `expdp` cannot decrypt data that has been encrypted with application encryption such as `DBMS_CRYPTO` procedures.

Data may be exported across network connections. If the `expdp` process connects to the database using a service name such as `hr/********@HR_DB`, the data may be encrypted if Oracle Advanced Security network encryption is specified between the client (where `expdp` is executing) and the server. The `expdp` process may also connect using a database link specified with the `NETWORK_LINK` parameter. The data will be sent across this link in clear text unless the database link has been configured to use network encryption.

The `ENCRYPTION` parameter determines the scope of the encryption—that is, which data elements are encrypted. The `ENCRYPTION_MODE` parameter determines the type of encryption used—that is, the type of key used (keystore or password). The `ENCRYPTION_PASSWORD` interacts with both the other parameters. The `ENCRYPTION_PWD_PROMPT` parameter determines whether Data Pump should prompt you for the encryption password. Default is `NO`.

ENCRYPTION Parameter

The ENCRYPTION parameter determines which elements of the dump file are encrypted:

- ENCRYPTED_COLUMNS_ONLY
- DATA_ONLY
- METADATA_ONLY
- ALL
- NONE (default)



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The ENCRYPTION parameter determines which elements of the dump file are encrypted; settings are shown in the slide.

The ENCRYPTED_COLUMNS_ONLY setting causes only columns that have been declared encrypted in the database to be encrypted in the dump file; all other data is in clear text.

The DATA_ONLY setting causes all the data in the dump file to be encrypted, but not the metadata, such as the data definition language (DDL) required to re-create the objects.

The METADATA_ONLY setting encrypts the metadata but not the data.

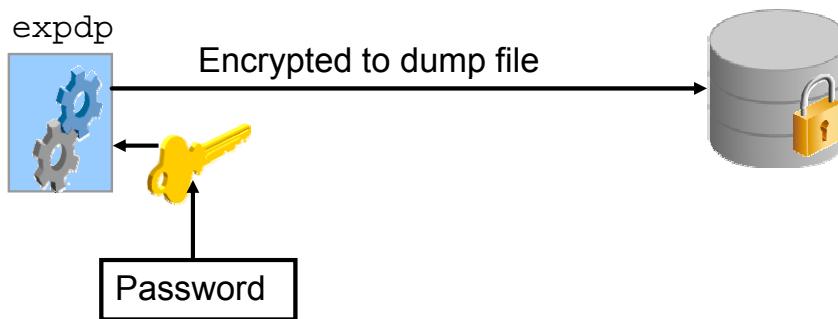
The ALL setting causes the entire dump file to be encrypted. If the data being exported includes SecureFiles, you must use the ALL setting to get encryption security for these objects.

The NONE setting is the default. If ENCRYPTION_PASSWORD is set and ENCRYPTION is not set, ENCRYPTION defaults to ALL.

ENCRYPTION_PASSWORD Parameter

ENCRYPTION_PASSWORD :

- Sets the password for the dump file
- Is required for ENCRYPTION_MODE=PASSWORD
- Is *not* related to any other password
- Is used to generate a key value for encrypting elements of the dump file



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The ENCRYPTION_PASSWORD parameter may be used by itself in the command line or the parameter file. ENCRYPTION_PASSWORD specifies a key for reencrypting encrypted table columns so that they are not written as clear text in the dump file set. If the export operation involves encrypted table columns, but an encryption password is not supplied, the encrypted columns will be written to the dump file set as clear text and a warning will be issued.

There is no connection or dependency between the key specified with the Data Pump ENCRYPTION_PASSWORD parameter and the key specified with the ENCRYPT keyword when the table with encrypted columns was initially created. For example, suppose a table is created as follows, with an encrypted column whose key is e3r:

```
CREATE TABLE emp (salary NUMBER(8,2) ENCRYPT IDENTIFIED BY "e3r");
```

When you export the EMP table, you can supply any arbitrary value for ENCRYPTION_PASSWORD. It does not have to be e3r.

Passwords should never be used in a command line.

Best Practice Tip: Place the ENCRYPTION_PASSWORD parameter in a parameter file.

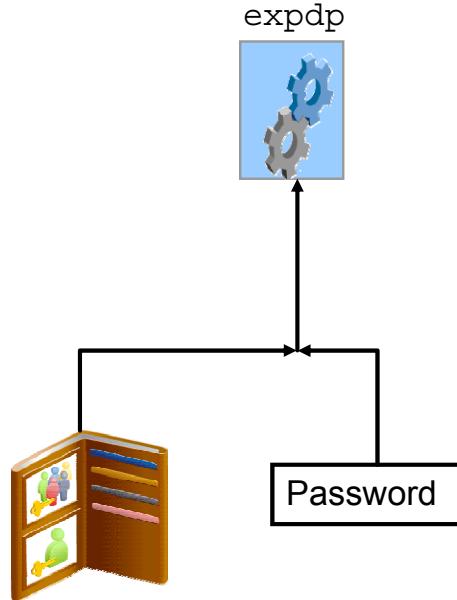
For network exports, the ENCRYPTION_PASSWORD parameter is not supported with user-defined external tables that have encrypted columns. In such cases, the table will be skipped and an error message will be displayed, but the job will continue.

Concurrent use of the ENCRYPTION_PWD_PROMPT and ENCRYPTION_PASSWORD parameters is prohibited.

ENCRYPTION_MODE Parameter

ENCRYPTION_MODE

- TRANSPARENT
- PASSWORD
- DUAL



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The ENCRYPTION_MODE parameter sets the method of obtaining the key for encrypting the dump file. The ENCRYPTION or ENCRYPTION_PASSWORD parameter must also be set when specifying the ENCRYPTION_MODE parameter.

If the encryption keystore is configured and TRANSPARENT is specified, the dump file is encrypted with no intervention by the DBA required. The ENCRYPTION_PASSWORD parameter is not needed and `expdp` will return an error if ENCRYPTION_PASSWORD is specified. A dump file exported in TRANSPARENT mode may be imported transparently if the encryption keystore is available. These dump files should be imported to the same database that they exported from.

When PASSWORD mode is specified, the password is not stored, but must be specified on import. Dump files created in PASSWORD mode are best suited for cases where the file will be imported offsite where the encryption keystore is not available. The ENCRYPTION_PASSWORD must be specified when using this mode. To import the dump file, the same password must be specified, and the target table must have the same encryption attributes as the source table (the same columns must be declared as ENCRYPT or NO ENCRYPT).

DUAL mode allows the dump file to be imported transparently where the encryption keystore is available or with a password where the keystore is not available.

Encrypting Dump Files

Data Pump Export can encrypt all or part of a dump file.

- Encrypt only data, with transparent mode:

```
$ expdp hr TABLES=employees  
    DIRECTORY=data_pump_dir DUMPFILE=hr_emp.dmp  
    ENCRYPTION_MODE = TRANSPARENT  
    ENCRYPTION = DATA_ONLY
```

- Encrypt the entire file, with password mode:

```
$ expdp oe TABLES=cust_payment_info  
    DIRECTORY=data_pump_dir DUMPFILE=cust_pay.dmp  
    ENCRYPTION = ALL  
    ENCRYPTION_MODE = PASSWORD  
    ENCRYPTION_PASSWORD = g&t1L47#
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Transparent Data Encryption (TDE) allows you to protect your database data files and image backups by encrypting the data of sensitive columns. Data Pump Export allows you to export that data into a dump file or an external table that is created in XML format. By default, the data in the dump file is in clear text.

In the example, you can encrypt only the data, or you can encrypt the entire dump file. The first example uses transparent mode and the second uses password mode to generate the key.

When you want to encrypt in the dump file, only the columns that are encrypted in the database, use ENCRYPTION=ENCRYPTED_COLUMNS_ONLY. The ENCRYPTION_PASSWORD must be specified. Therefore, ENCRYPTION_MODE must be PASSWORD.

The example shows the encryption password on the command line. Passwords should never be placed on the command line. Use PARFILE with expdp or impdp to specify ENCRYPTION_PASSWORD.

Summary

In this lesson, you should have learned how to use:

- RMAN-encrypted backups
- Oracle Secure Backup
- Data Pump Export encryption



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Practice: Overview

This practice covers the following topics:

- 20-1: Creating and recovering backups using transparent-mode, dual-mode, password-mode backups
- 20-2: Exporting encrypted tables
- 20-3: Importing encrypted tables



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Module

Auditing

A solid red horizontal bar spanning most of the page width, positioned below the module title.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

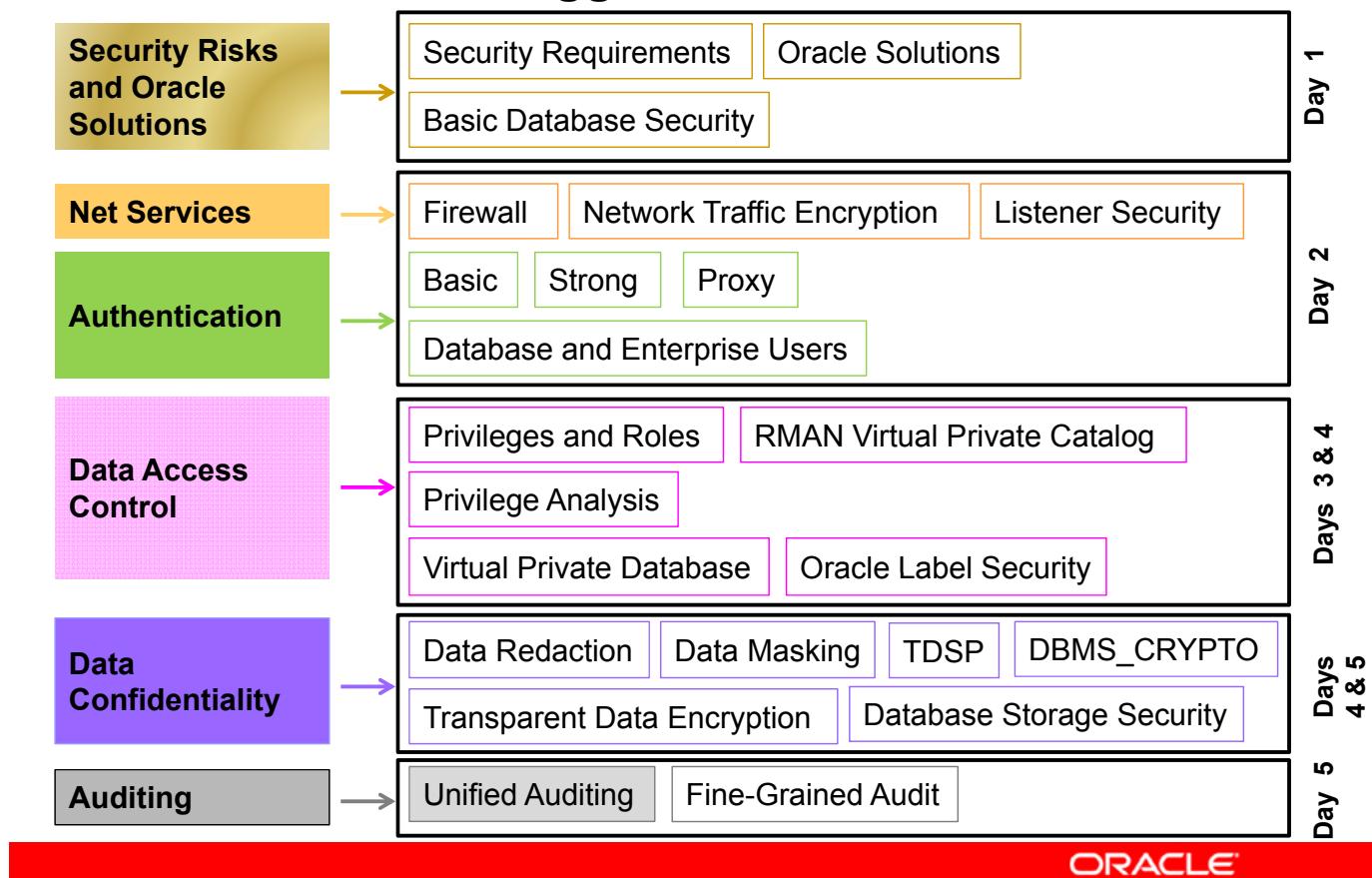
21

Using Unified Auditing

ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Suggested Schedule



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Understand the implementation and benefits of using the unified audit trail
- Enable unified auditing
- Configure the unified auditing
- Create and enable audit policies
- Add application context data to audit records
- View data in the unified audit trail
- Clean up the unified audit trail



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Note

For a complete understanding of Oracle Unified Auditing new feature and usage, refer to the following guides in the Oracle documentation:

- *Oracle Database Security Guide 12c Release 1 (12.1)*

Refer to other sources of information:

- *Oracle Database 12c New Features Demo Series* demonstrations under Oracle Learning Library:
 - *Unified Auditing*
- *Oracle By Example (OBE)*:
 - *Implementing Unified Auditing*

Types of Auditing

Monitor activity:

Type of Auditing	Activity Audited
Mandatory auditing	<ul style="list-style-type: none"> • STARTUP, SHUTDOWN • SYSDBA, SYSOPER, SYSASM connections
Standard database auditing	AUDIT_TRAIL parameter
Privileged users auditing	AUDIT_SYS_OPERATIONS = TRUE
Fine-grained auditing	DBMS_FGA.ADD_POLICY
Oracle Database Vault auditing	Any violation against any component: <ul style="list-style-type: none"> • Realm, Command Rule, Secure Application Role • Rule, Factor



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Compliance, privacy, and increasingly sophisticated security threats have resulted in an almost mandatory requirement for turning on native auditing. This is especially true for monitoring privileged users, but also for actions that may be out of the normal operations that one would expect to see in a production environment. In fact, privileged users have increasingly been the target of external hackers. As a result, auditing privileged user activity and application changes such as DDL operations are especially important. Database link creation, create table, and truncate table are just a few of the DDL operations that should be audited. Starting with Oracle Database 11g, auditing for these types of operations is configured by default. Oracle Database 11g provides three different types of auditing. The administrator can audit all actions that take place within the database. It is important to remember that auditing should be focused so that only events that are of interest are captured. Properly focused auditing has minimal impact on system performance and provide highly relevant data for reporting and alerting purposes.

- Standard database auditing captures several pieces of information about an audited event, including that the event occurred, when it occurred, the user who caused the audited event, and which client machine was being used when the event happened.
- Audit privileged user operations with the AUDIT_SYS_OPERATIONS parameter.
- Independently of standard database auditing, use fine-grained auditing to audit SQL data manipulation language statements occurring only when specific columns are impacted under certain conditions.

Oracle Database 12c Auditing

- **Simplicity:**
 - Audit options grouped into a simple audit policy
 - Simpler action-based audit configuration
 - Condition-based audit configuration
 - Users exempted from being audited
- **Consolidation:** One single unified audit trail
- **Security:**
 - Read-only audit trail table
 - Any operation related to audit configuration audited
 - Any SYS user actions audited
 - Separation of audit administration duties with audit roles
- **Performance:**
 - Negligible overhead

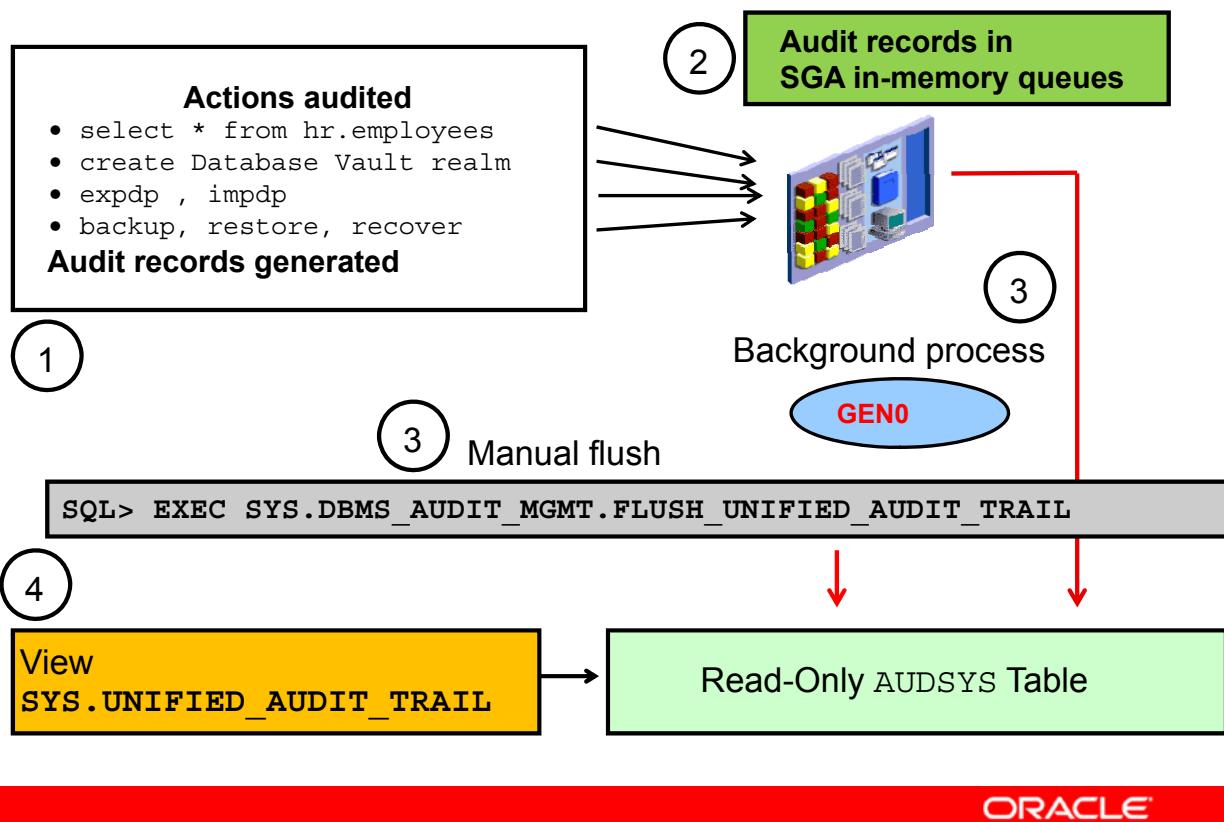


Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The unified auditing facility needs to address the following challenges:

- **Simplicity:**
 - Grouping audit options into a simple audit policy
 - Allowing simpler action-based audit configurations
 - Setting condition-based audit configurations
 - Exempting users from being audited
- **Consolidation:** Merging all audit trails into a single unified audit trail table
- **Security:**
 - Relying on a read-only audit trail table
 - Auditing any operation related to audit configuration
 - Auditing any SYS user auditable action
 - Separating audit administration duties with audit administration roles, AUDIT_ADMIN and AUDIT_VIEWER
- **Performance:**
 - Oracle Database 12c auditing provides the ability to use System Global Area (SGA) queues for accumulating audit records, enabling auditing to be turned on with negligible overhead.

Security and Performance: Audit Architecture



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

How does the new audit architecture work?

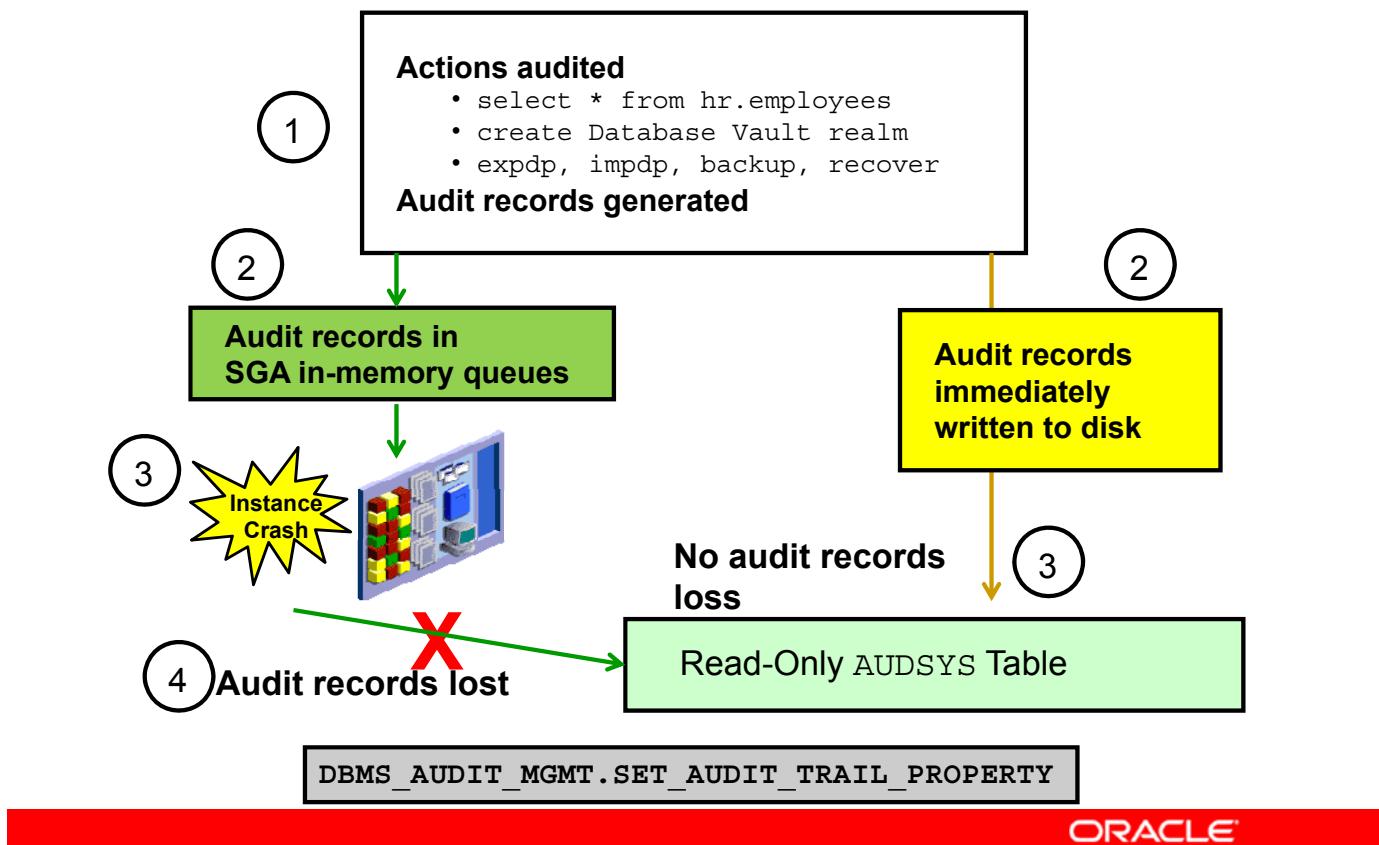
Named queues inside the SGA have a corresponding persistent storage in tables owned by AUDSYS.

When the SGA queue overflows, a background process flushes the contents to the table. Each client has two SGA queues so that a client can continue to write to the second queue while the first queue is being persisted to the table.

When you want to see the audit trail records immediately in the `UNIFIED_AUDIT_TRAIL` view, use the following procedure to explicitly flush the queues to disk:

```
SQL> EXEC SYS.DBMS_AUDIT_MGMT.FLUSH_UNIFIED_AUDIT_TRAIL
```

Tolerance Level for Loss of Audit Records



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The in-memory queuing infrastructure means that the audit records are not persisted immediately but queued, thus achieving the performance benefit. However, in cases like instance crash, *queued* audit records can be lost. Hence, the audit trail must be configurable to indicate the tolerance level. The following two modes are based on different levels of loss tolerance:

- Immediate-Write mode: The audit records are written immediately. This comes with a performance cost.

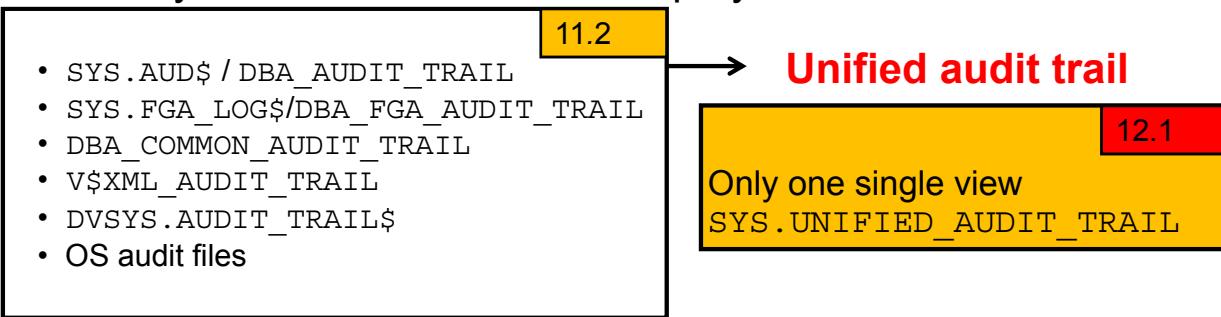
```
SQL> EXEC DBMS_AUDIT_MGMT.SET_AUDIT_TRAIL_PROPERTY(-
  2      DBMS_AUDIT_MGMT.AUDIT_TRAIL_UNIFIED, -
  3      DBMS_AUDIT_MGMT.AUDIT_TRAIL_WRITE_MODE, -
  4      DBMS_AUDIT_MGMT.AUDIT_TRAIL_IMMEDIATE_WRITE);
```

- Queued-Write mode: The content of the SGA queues is periodically dumped to disk. This is the default mode for the new audit engine.

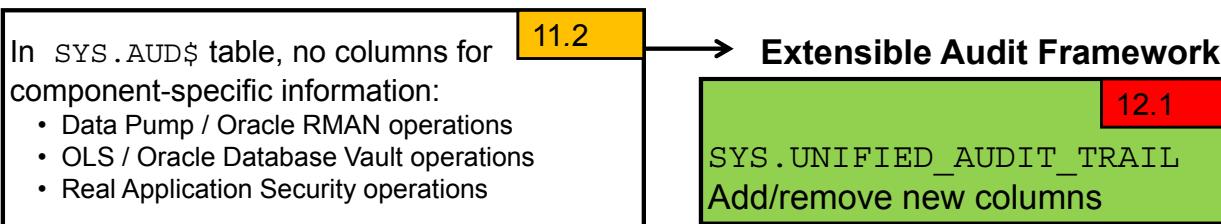
```
SQL> EXEC DBMS_AUDIT_MGMT.SET_AUDIT_TRAIL_PROPERTY(-
  2      DBMS_AUDIT_MGMT.AUDIT_TRAIL_UNIFIED, -
  3      DBMS_AUDIT_MGMT.AUDIT_TRAIL_WRITE_MODE, -
  4      DBMS_AUDIT_MGMT.AUDIT_TRAIL_QUEUED_WRITE);
```

Consolidation: Unique Audit Trail

Too many audit trails to be looked up by auditors:



No easy extension to add an audit column to the audit trail:



Management and security of audit trail improved with a single audit trail.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

- In Oracle Database 11g, the Oracle database has too many audit trails where the audited data must be looked up by auditors.
The first major challenge is to consolidate all audit trails in a unified audit trail.
- The existing audit trail tables do not easily extend to add or remove new audit columns for newer RDBMS components, like Data Pump, Oracle Label Security (OLS), or Oracle Database Vault, to supply additional component-specific information.
The second challenge is therefore to allow components to extend the basic audit information via the extensible audit framework to supply their own additional audit information in the unified audit trail. For example, in Oracle Database 11g, OLS actually stores its audit data in the SYS.AUD\$ table, but there are no specific columns related to OLS specific information.
- The management and security of audit trail becomes easier with a single audit trail.

Basic Audit Versus Extended Audit Information

Basic Audit Information record

Database session:

- User name, Database Client
- Terminal, IP Address
- Instance number, DBID

Database operation:

- Action executed
- SCN
- Object accessed, SQL statement

Basic Audit Information

BAI in view

SYS.UNIFIED_AUDIT_TRAIL

Extended Audit Information columns

For component-specific information:

- FGA: **FGA_POLICY_NAME**
- Data Pump operations: **DP_xxx**
- RMAN operations: **RMAN_xxx**
- OLS operations: **OLS_xxx**
- DV violations/changes: **DV_xxx**
- RAS operations: **XS_xxx**

Extended Audit Information

EAI in view

SYS.UNIFIED_AUDIT_TRAIL

New columns

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Basic Audit Information

In Oracle Database 11g, the attributes combining Database Session information and Database Operation information constitute the basic format of an audit record in Oracle Database. This basic format is called Basic Audit Information or BAI for short.

The SQL AUDIT command configures the audit for SQL operations and connections.

Extended Audit Information

Apart from these attributes, the different RDBMS components could have their own actions, like the Data Pump “Direct Path Load” actions or the OLS “Apply Policy” or the Oracle Database Vault “Create Command Rule” which are PL/SQL procedures. Any database component, like FGA, Data Pump, Oracle RMAN, OLS, Oracle Database Vault (DV), or Real Application Security (RAS), can supply additional component-specific information and store it in the unified audit trail view. This additional component-specific information is called Extended Audit Information, or EAI for short.

Though the audit trail is unified, the audit records appear separately for each type of auditing. For example, if the audit is configured in both standard auditing and Oracle Database Vault, separate audit records corresponding to each of these components appear in the audit trail.

Extended Audit Information

EAI in SYS.UNIFIED_AUDIT_TRAIL view: new columns

FGA event

- FGA_POLICY_NAME
- DP_TEXT_PARAMETERS1
- DP_BOOLEAN_PARAMETERS1

Data Pump Export / Import

RMAN backup / recovery

- RMAN_OPERATION
- RMAN_OBJECT_TYPE
- RMAN_DEVICE_TYPE
- RMAN_xxx

OLS operations

- OLS_POLICY_NAME
- OLS_xxx
- DV_xxx
- XS_xxx

Database Vault violations or configuration changes

Real Application Security

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

EAI in the UNIFIED_AUDIT_TRAIL New Columns

A database component can be audited like:

- FGA component, which is registered with the framework. The EAI is stored in the FGA_POLICY_NAME column.
- Data Pump when export or import operations are executed. The EAI is stored in two new columns: DP_TEXT_PARAMETERS1 and DP_BOOLEAN_PARAMETERS1.
- Oracle RMAN when backup or restore or recovery operations are executed. The EAI is stored in several new columns: RMAN_OPERATION would contain either 'Backup' or 'Restore' or 'Recover'.
- OLS when applying an OLS policy on a table, for example. The EAI is stored in the OLS_xxx columns.
- Oracle Database Vault when configuration changes or violations occur against Oracle Database Vault components, such as realms, command rules, and rulesets factors. The EAI is stored in the DV_xxx columns.
- RAS operations. The EAI is stored in the XS_xxx columns.

Data Pump Audit Policy

1. Create the audit policy for the component and actions.

```
SQL> CREATE AUDIT POLICY dp_pol1 ACTIONS
      2          COMPONENT=datapump export;
```

2. Enable the audit policy.

```
SQL> audit policy DP_POL1;
```

3. Perform an export operation.

```
$ expdp hr/oracle_4U dumpfile=HR_tables tables=EMPLOYEES,JOBS
      directory=DATA_PUMP_DIR
```

4. Flush the audit data to disk.

```
SQL> EXEC SYS.DBMS_AUDIT_MGMT.FLUSH_UNIFIED_AUDIT_TRAIL
```

5. View the Data Pump EAI.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Data Pump EAI in UNIFIED_AUDIT_TRAIL New Columns

If you want the Data Pump Export or Import operations to be audited, you perform these steps:

1. Create an audit policy for export or import or for both. An audit policy is a named group of audit settings that enable you to audit a particular aspect of user behavior in the database.
2. Next, enable the audit policy that was created to audit export operations.
3. Perform an export operation. At each export operation, the EAI is then stored in two new columns of the UNIFIED_AUDIT_TRAIL view: DP_TEXT_PARAMETERS1 and DP_BOOLEAN_PARAMETERS1.
4. Flush the audit information to disk by using the DBMS_AUDIT_MGMT.FLUSH_UNIFIED_AUDIT_TRAIL procedure.
5. Display the Data Pump audit information in the UNIFIED_AUDIT_TRAIL view.

Oracle RMAN Audit Information

1. Perform an Oracle RMAN operation.

```
RMAN> backup tablespace users;
RMAN> restore tablespace users;
RMAN> recover tablespace users;
```

2. Flush the audit data to disk.

```
SQL> EXEC SYS.DBMS_AUDIT_MGMT.FLUSH_UNIFIED_AUDIT_TRAIL
```

3. View the Oracle RMAN EAI.

```
SQL> select DBUSERNAME, RMAN_OPERATION from UNIFIED_AUDIT_TRAIL
  2  where RMAN_OPERATION is not null;

DBUSERNAME                      RMAN_OPERATION
-----                         -----
SYSBACKUP                        Backup
SYSBACKUP                        Restore
SYSBACKUP                        Recover
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

RMAN EAI in the UNIFIED_AUDIT_TRAIL New Columns

Unlike other Oracle Database components, you do not create a unified audit policy for Oracle RMAN events. The UNIFIED_AUDIT_TRAIL view automatically captures commonly audited Oracle RMAN events.

Unified Audit Implementation

→ Mixed auditing mode

Allow smooth migration of existing databases to use the unified auditing features.

→ Unified auditing mode

Before populating the UNIFIED_AUDIT_TRAIL view:

- Enable unified auditing.
 1. Shut down all database instances, then the listener of the ORACLE_HOME.
 2. cd \$ORACLE_HOME/rdbms/lib
make -f ins_rdbms.mk **uniaud_on** ioracle
ORACLE_HOME=\$ORACLE_HOME
 3. Start the listener, then all database instances.
- Define a tablespace for the read-only audit table.

The AUDIT_xxx instance parameters are no longer observed.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Mixed Auditing Mode

Mixed auditing mode allows both the new audit engine and the traditional audit engine to work simultaneously.

- On upgrade, none of the existing audit settings are affected and the database continues to support the traditional audit facilities. It is recommended you migrate to the unified audit facility to take advantage of the new audit architecture and performance improvements. Before you decide to switch to the unified auditing mode, you can use the mixed mode by creating a policy with CREATE AUDIT POLICY command and then enabling it with AUDIT command. If you do not wish to create a new policy, you can simply enable one of the predefined policies - ORA_SECURECONFIG or ORA_ACCOUNT_MGMT or ORA_DATABASE_PARAMETER. Either of this puts the database in mixed auditing mode. The old audit syntax continues to work and the old audit destinations continues to be written to. However, new audit policies can be created and audit data also writes to the new unified audit trail.
- When a database is created, mixed auditing mode is used by default through the predefined enabled policy ORA_SECURECONFIG. But unified auditing mode is not yet enabled.

Unified Auditing Mode

To enable unified auditing, relink Oracle with the `uniaud_on` option after shutting down all database instances and the listener in the `ORACLE_HOME`. By default, the new unified audit data is written to the `SYSAUX` tablespace. If you do not want to use the `SYSAUX` tablespace, create a new tablespace to host the new read-only audit trail table, which is owned by the new `AUDSYS` schema. Note that the `AUDSYS` user is locked by default. After enabling unified audit mode, none of the old audit instance parameters are honored: `AUDIT_TRAIL`, `AUDIT_FILE_DEST`, `AUDIT_SYS_OPERATIONS`, `AUDIT_SYSLOG_LEVEL`. There is no possibility to write audit records to OS or syslog.

For a Windows environment, you rename
the `ORACLE_HOME%/bin/orauniaud12.dll dbl` file to
`%ORACLE_HOME%/bin/orauniaud12.dll`.

Quiz

To audit RMAN backup, restore, and recovery operations, you must create new audit policies.

- a. True
- b. False



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: b

Unlike other Oracle Database components, you do not create a unified audit policy for Oracle RMAN events. The `UNIFIED_AUDIT_TRAIL` view automatically captures commonly audited Oracle RMAN events.

Quiz

Select the schema owner of the unified audit trail.

- a. SYS
- b. AUDSYS
- c. AUDIT_ADMIN
- d. LBACSYS
- e. DVSYS

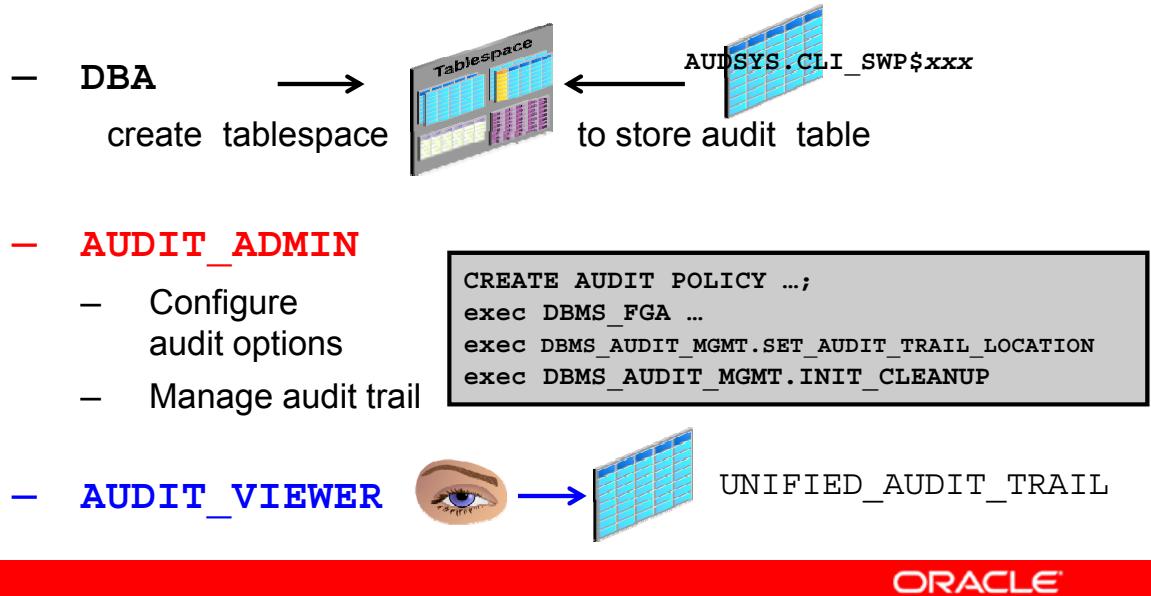


Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: b

Security: Roles

- Read-only audit trail table in AUDSYS schema
- Any audit-related activity mandatorily audited
- Roles to administer auditing and view audit data:



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

ORACLE

Addressing Oracle Database 12.1 Security Challenges

Compliance regulations and laws require businesses to secure audit data by auditing users, activities, and associated data.

Read-Only Audit Trail Table

The audit trail table owned by the AUDSYS schema is in read-only mode, and can be updated only during recovery or upgrade operations by using the SYSDBA privilege.

Audit Configuration Operations

Any audit-related activity, such as creation, modification, deletion, enabling of audit policies, and execution of DBMS_AUDIT_MGMT and DBMS_FGA packages, are audited by mandate.

Roles

The SYSDBA privilege is no longer required to configure audit settings and management of the audit trail. To enable separation of duties, the AUDIT_ADMIN role is required to perform these functions with the AUDIT_SYSTEM and AUDIT_ANY system privileges, the SELECT privilege on various audit views and the EXECUTE privilege on DBMS_AUDIT_MGMT and DBMS_FGA packages.

The AUDIT_VIEWER role allows for viewing and analyzing the audit data.

Security: SYS Auditing

Non Unified Auditing

```
AUDIT_SYS_OPERATIONS=true
AUDIT_FILE_DEST=directory
```

SYSDBA, SYSOPER, SYSASM, SYSBACKUP,
SYSKM, SYSDG:
Statements recorded:
STARTUP, SHUTDOWN, ALTER DATABASE
+
SELECT, DML, DDL

OS audit directory

Unified Auditing

```
AUDIT_SYS_OPERATIONS=true
AUDIT_FILE_DEST=directory
```

SYSDBA, SYSOPER, SYSASM, SYSBACKUP,
SYSKM, SYSDG:
Statements recorded:
STARTUP, SHUTDOWN, ALTER DATABASE ...
+
Enabled actions in audit policies

SYS.UNIFIED_AUDIT_TRAIL

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

When **SYS** is connected as **SYSDBA**, **SYSOPER**, **SYSBACKUP**, **SYSASM**, **SYSKM**, or **SYSDG**, it is subjected to:

- All top-level commands, such as **STARTUP**, **SHUTDOWN**, **ALTER DATABASE**, and **ALTER SYSTEM**, until the database opens.
- System-wide audit policies configured by the audit administrator. When the database opens, Oracle Database audits these users by using the audit configurations in the system—not only the ones that were applied in the **AUDIT** command for **SYS**. It audits all users when the **AUDIT** statement does not define a list of users for which the policy applies.
- Audit when a table or a PL/SQL procedure is configured for auditing.

The audit administrator can configure audit policies that apply to the **SYS** user. This configuration is covered later in the lesson.

The mandatory and **SYS** auditing records are written to **\$ORACLE_BASE/audit/<dbname>** when the database is not yet open. These audit records can be loaded into the database using the **DBMS_AUDIT_MGMT.LOAD_UNIFIED_AUDIT_FILES** procedure when the database is open. This procedure deletes the external audit files after they are successfully loaded in the database.

Simplicity: Audit Policy

- No more AUDIT_xxx instance parameters
- To start auditing, perform the following steps:
 1. Create audit policies used to group audit options:
 - **CREATE AUDIT POLICY** command:
 - Audit options: System-wide or object-specific or role
 - Optional **WHEN** condition **EVALUATE PER STATEMENT**
 - Optional event-handler alert
 - **CONTAINER** = CURRENT | ALL
 2. Enable/disable audit policies: AUDIT and NOAUDIT
 - Define users audited: all by default
 - User list: BY username
 - User exception list: EXCEPT username
- View audited data in **UNIFIED_AUDIT_TRAIL** view

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In a unified audit database, the following actions are audited by mandate: CREATE AUDIT POLICY, ALTER AUDIT POLICY, DROP AUDIT POLICY, AUDIT, NOAUDIT, EXECUTE of DBMS_FGA, and EXECUTE of DBMS_AUDIT_MGMT.

Auditing actions based on system or object privileges or roles are performed, provided that you create and enable audit policies. What are audit policies?

- A named group of audit options that enable you to audit a particular aspect of user behavior in the database by using system, object, or role privileges
- Created with the **CREATE AUDIT POLICY** command requiring the **AUDIT_ADMIN** role
- Enforced conditionally and evaluated during the statement execution or once during the session or at the instance level
- Supplied with an event-handler procedure alerting that the audit occurred
- Applied in a multitenant container database (CDB) either in the root or a pluggable database (PDB)
- Not active until explicitly enabled with the **AUDIT** command
Be aware that existing standard non-policy audit options configured with the **AUDIT** and **NOAUDIT** commands are not effective.
- Applied with the exception of certain user lists.

Step 1: Creating the Audit Policy

Create audit policies based on **system-wide** audit options.

- System privilege

```
SQL> CREATE AUDIT POLICY audit_syspriv_pol1
  2      PRIVILEGES SELECT ANY TABLE, CREATE LIBRARY;
```

- Actions

```
SQL> CREATE AUDIT POLICY audit_actions_pol2
  2      ACTIONS AUDIT, ALTER TRIGGER;
```

- Role

```
SQL> CREATE AUDIT POLICY audit_role_pol3
  2      ROLES mgr_role;
```

- System privilege, actions, and roles

```
SQL> CREATE AUDIT POLICY audit_mixed_pol4
  2      PRIVILEGES DROP ANY TABLE
  3      ACTIONS     CREATE TABLE, DROP TABLE, TRUNCATE TABLE
  4      ROLES       emp_role;
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Audit Policy Creation: System-Wide Options

To create an audit policy, you must first define system-wide or object-specific audit options.

The system-wide options can be of three types:

- The **privilege audit option** audits all events, which exercise the specified system privilege, as in the first example on the slide.
- The **action audit option** indicates which RDBMS action should be audited, such as the ALTER TRIGGER action in the second example.
- The **role audit option** audits the use of all system or object privileges granted directly to the role MGR_ROLE, as in the third example.

You can configure privilege, action, and role audit options in the same audit policy, as shown in the fourth example.

Find the list of auditable system-wide options in the SYS.AUDITABLE_SYSTEM_ACTIONS table.

Creating the Audit Policy: Object-Specific Actions

Create audit policies based on **object-specific** options.

```
SQL> CREATE AUDIT POLICY audit_objpriv_pol5
  2       ACTIONS SELECT, UPDATE,
  3             LOCK ON hr.employees;
```

```
SQL> CREATE AUDIT POLICY audit_objpriv_pol6
  2       ACTIONS ALL;
```

```
SQL> CREATE AUDIT POLICY audit_objpriv_pol7
  2       ACTIONS EXECUTE,
  3             GRANT ON hr.raise_salary_proc;
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Object-specific options are the second type of audit option. These options are actions that are specific to objects in the database. The object-level action audit options are listed in the `SYS.AUDITABLE_OBJECT_ACTIONS` table.

The first example on the slide creates an audit policy to audit any select and update action on any object, and any lock on the `HR.EMPLOYEES` table.

The second example creates an audit policy to audit any object-specific action on any object.

The third example creates an audit policy to audit any execute action on any procedural object, and any grant on the `HR.RAISE_SALARY_PROC` procedure.

The object-level audit options are dynamic. That is, changes in these options become applicable for current and subsequent user sessions.

Creating the Audit Policy: Condition

- Condition and evaluation PER SESSION

```
SQL> CREATE AUDIT POLICY audit_mixed_pol5
  2      ACTIONS RENAME ON hr.employees,
  3          ALTER ON hr.jobs,
  4 WHEN 'SYS_CONTEXT (''USERENV'', ''SESSION_USER'')='JIM'
  5 EVALUATE PER SESSION;
```

- Condition and evaluation PER STATEMENT

```
SQL> CREATE AUDIT POLICY audit_objpriv_pol6
  2      ACTIONS ALTER ON OE.ORDERS
  3 WHEN
  4 'SYS_CONTEXT(''USERENV'', ''CLIENT_IDENTIFIER'')='OE'
  5 EVALUATE PER STATEMENT;
```

- Condition and evaluation PER INSTANCE

```
SQL> CREATE AUDIT POLICY audit_objpriv_pol7
  2      ROLES dba
  3 WHEN SYS_CONTEXT(''USERENV'', ''INSTANCE_NAME'')='sales'
  4 EVALUATE PER INSTANCE;
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Audit Policy Creation: Condition and Evaluation

The audit policies can evaluate the condition per statement, once per session or once per instance.

The first example on the slide creates an audit policy to audit any rename action on the HR.EMPLOYEES table, and any alter action on the HR.JOB\$ table, provided that the user executing the audited statement is JIM. The condition is evaluated only once in the session.

The second example creates an audit policy to audit any alter action on the OE.ORDERS table, provided that the user executing the audited statement is the schema owner OE. The condition is evaluated each time an ALTER statement is executed on the OE.ORDERS table.

The third example creates an audit policy to audit any privilege granted to the DBA role, provided that the instance name is "sales". The condition is evaluated only once during the database instance lifetime. After Oracle Database evaluates the condition, it caches and reuses the result for the remainder of the instance lifetime.

Step 2: Enabling / Disabling the Audit Policy

Enable audit policies:

- Apply to all users.

```
SQL> AUDIT POLICY audit_syspriv_pol1;
```

- Apply only to some users.

```
SQL> AUDIT POLICY audit_pol2 BY scott, oe;
SQL> AUDIT POLICY audit_pol3 BY sys;
```

- Exclude some users.

```
SQL> AUDIT POLICY audit_pol4 EXCEPT jim, george;
```

- Audit the recording based on failed or succeeded actions.

```
SQL> AUDIT POLICY audit_syspriv_pol1 WHENEVER SUCCESSFUL ;
SQL> AUDIT POLICY audit_objpriv_pol2 WHENEVER NOT SUCCESSFUL ;
```

```
SQL> AUDIT POLICY auditpol5 BY joe WHENEVER SUCCESSFUL ;
```

Disable audit policies by using the NOAUDIT command.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Audit Policy Enabling

The next step is to enable the audit policy by using the AUDIT command.

All users are audited by default, except if you define a list of those audited.

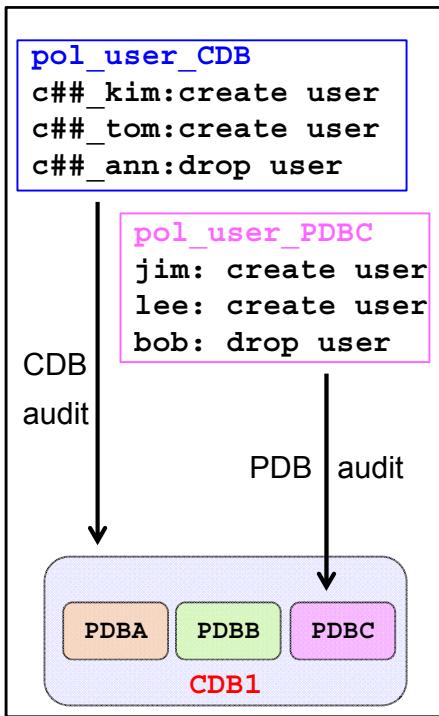
- Apply the audit policy to one or more users by using the BY clause.
The audit Administrator audits SYS-user-specific actions the same way as other user actions.
- Exclude some users by using the EXCEPT clause.
You cannot have a BY list and an EXCEPT list in the same policy enablement statement

Audit records are generated whether the user's actions failed or succeeded. If you want to audit actions only when the user's actions failed, use the WHENEVER NOT SUCCESSFUL clause. If you want to audit actions only when the user's actions succeeded, use the WHENEVER SUCCESSFUL clause. When you omit the WHENEVER clause, the statement is audited whether the action is successful or not, and the RETURN_CODE column displays whether the action succeeded or not.

Audit Policy Disabling

To disable an audit policy, use the NOAUDIT command.

Auditing Actions in a CDB and PDBs



Choose CDB or PDB level:

1. Connect to the root or a PDB.
2. Create audit policies:
 - Audit options: Systemwide or object-specific or role
 - Optional **WHEN** condition **EVALUATE PER STATEMENT**
 - **CONTAINER** = CURRENT | ALL
2. Enable/disable audit policies:
AUDIT and **NOAUDIT**
 - Define users audited: all by default

ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Auditing actions based on system or object privileges or roles are performed, provided that you create and enable audit policies. The policies can be created either at the CDB level for the whole CDB and or at a PDB level for a single PDB. What are audit policies?

- A named group of audit options that enable you to audit a particular aspect of user behavior in the database by using system, object, or role privileges
- Created with the `CREATE AUDIT POLICY` command requiring the `AUDIT_ADMIN` role
- Enforced conditionally and evaluated during the statement execution or once during the session or at the instance level
- Applied either in the root or a PDB: This provides the ability to create audit policies used by all PDBs and audit policies used exclusively for each PDB. An audit configuration that is not enforced across all PDBs means it applies only within a PDB and is not visible outside it.
- In the example of the slide, the `pol_user_CDB` created from the root will audit any `CREATE` or `DROP USER` performed by specific users (`C##_KIM`, `C##_TOM`, and `C##_ANN`) and the `pol_user_PDBC` created from the PDB will audit any `CREATE` or `DROP USER` performed by specific users (`JIM`, `LEE`, and `BOB`)
- Not active until explicitly enabled with the `AUDIT` command
- Applied with the exception of certain user lists.

Viewing the Audit Policy

- View audit policy.

```
SQL> SELECT POLICY_NAME, AUDIT_OPTION, CONDITION_EVAL_OPT
  2  FROM  AUDIT_UNIFIED_POLICIES;

POLICY_NAME          AUDIT_OPTION      CONDITION_EVAL_OPT
-----              -----
POL1                 DELETE            INSTANCE
POL2                 TRUNCATE TABLE    NONE
POL3                 RENAME            SESSION
POL4                 ALL ACTIONS      STATEMENT
```

- View the enabled audit policy.

```
SQL> SELECT POLICY_NAME, ENABLED_OPT, USER_NAME, SUCCESS, FAILURE
  2  FROM  AUDIT_UNIFIED_ENABLED_POLICIES;

POLICY_NAME          ENABLED_          USER_NAME     SUC FAI
-----              -----
POL3                 BY                PM           NO   YES
POL2                 EXCEPT          SYSTEM        NO   YES
POL4                 BY                SYS          YES  YES
POL6                 BY                ALL USERS   YES  NO
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Verify the list of audit policies created in the AUDIT_UNIFIED_POLICIES view.

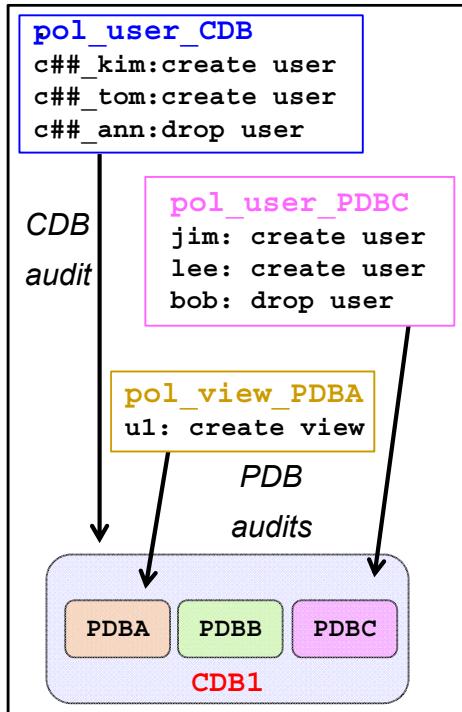
- The CONDITION_EVAL_OPT column defines when the condition, if any, is evaluated.

Verify the list of enabled audit policies in the AUDIT_UNIFIED_ENABLED_POLICIES view.

- The ENABLED_OPT column defines if a list of audited users is defined with the BY value or an exception list of excluded users is defined with the EXCEPT value. The audited or excluded users are listed in the USER_NAME column.
- The SUCCESS and FAILURE columns define if the policy generates audit records only when the user's actions succeed or fail.

Viewing the Audit Records

CDB_UNIFIED_AUDIT_TRAIL



Audit records are generated in each container with a distinct DBID.

- When connected to **PDBA**, only **PDBA** rows in **UNIFIED_AUDIT_TRAIL**
- When connected to **PDBC**, only **PDBC** rows in **UNIFIED_AUDIT_TRAIL**
- When connected to the **root**, only **root** rows in **UNIFIED_AUDIT_TRAIL**
- Query CDB UNIFIED AUDIT TRAIL**

```

SQL> SELECT con_id, ACTION_NAME
  2  FROM CDB_UNIFIED_AUDIT_TRAIL;

  CON_ID ACTION_NAME
  -----
  1  ALTER USER
  3  CREATE VIEW
  4  CREATE TABLE
  
```

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In a non-CDB, the **UNIFIED_AUDIT_TRAIL** view displays all audit rows. In a CDB, the **UNIFIED_AUDIT_TRAIL** view displays all audit rows of the container you are connected to.

For example, if an audit policy has been created and enabled in **PDBA**, and an audit policy has been created and enabled in **PDBC**, audit records are generated in both PDBs with a distinct DBID.

- When connected to **PDB1**, the **UNIFIED_AUDIT_TRAIL** view displays only **PDB1** rows.
- When connected to **PDB2**, the **UNIFIED_AUDIT_TRAIL** view displays only **PDB2** rows.
- When connected to the root, the **UNIFIED_AUDIT_TRAIL** view displays only root rows.

You can query the **CDB_UNIFIED_AUDIT_TRAIL** view - this is a consolidated view of all containers, to retrieve all audit rows of the CDB.

Using Predefined Audit Policies

```
SQL> select POLICY_NAME, AUDIT_OPTION
  2  from  AUDIT_UNIFIED_POLICIES
  3 where policy_name in
  4          ('ORA_ACCOUNT_MGMT',
  5           'ORA_DATABASE_PARAMETER',
  6           'ORA_SECURECONFIG')
  7 order by 1,2 ;

POLICY_NAME                      AUDIT_OPTION
-----                         -----
ORA_ACCOUNT_MGMT                  ALTER ROLE
ORA_ACCOUNT_MGMT                  ALTER USER ...
ORA_DATABASE_PARAMETER             ALTER DATABASE
ORA_DATABASE_PARAMETER             ALTER SYSTEM
ORA_DATABASE_PARAMETER             CREATE SPFILE
ORA_SECURECONFIG                   ADMINISTER KEY MANAGEMENT ...
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

There are three main predefined audit policies for operations, which are commonly audited and recommended under Oracle's Audit Best Practices.

- The `ORA_ACCOUNT_MGMT` audit policy audits events for the user account and privilege management.
- The `ORA_DATABASE_PARAMETER` audit policy audits the changes made to the database parameters settings.
- The `ORA_SECURECONFIG` audit policy audits all Oracle Database 11g secure configuration audit options.

The users must enable or disable such precreated audit policies, based on their requirements. Only the `ORA_SECURECONFIG` is enabled by default.

```
SQL> select POLICY_NAME from AUDIT_UNIFIED_enabled_policies
  2  where policy_name in ('ORA_ACCOUNT_MGMT',
  3           'ORA_DATABASE_PARAMETER', 'ORA_SECURECONFIG') ;
```

```
POLICY_NAME
-----
ORA_SECURECONFIG
```

Including Application Context Data

Context Namespace	Attribute / Value
CONTEXT_HR	MyJob – Developer MyLoc – Paris
CONTEXT_SALES	MyRole – Manager

Use the AUDIT CONTEXT ATTRIBUTES command.

```
SQL> AUDIT CONTEXT NAMESPACE context_HR
  2       ATTRIBUTES MyJob, MyLoc
  3   CONTEXT NAMESPACE context_SALES
  4       ATTRIBUTES MyRole BY jim;
```

```
SQL> select APPLICATION_CONTEXTS from UNIFIED_AUDIT_TRAIL
  2 where APPLICATION_CONTEXTS is not null;
APPLICATION_CONTEXTS
-----
(context_HR,MyJob=Manager); (context_HR,MyLoc=Paris)
```

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Capturing Application Context Information in the Unified Audit Trail

Applications set various contexts before executing actions on the database server, storing information like Module Name and Client_ID. These values are typically stored in application contexts. An application context is an attribute name and attribute value pairs in a namespace, and each pair can be appended in audit data.

The audit administrator can configure application contexts and their attributes that must be captured in audit data by using the AUDIT CONTEXT NAMESPACE ATTRIBUTE command.

To display the audited application context data, select from the UNIFIED_AUDIT_TRAIL view.

To list the application contexts audited, use the AUDIT_UNIFIED_CONTEXTS view:

```
SQL> select * from audit_unified_contexts;
  NAMESPACE          ATTRIBUTE          USER_NAME
  -----
  CONTEXT_HR          MYJOB             ALL_USERS
  CONTEXT_HR          MYJOB             ALL_USERS
  CONTEXT_SALES       MYROLE            JIM
```

To disable an application context audit setting, use the NOAUDIT command.

Dropping the Audit Policy

1. Disable the audit policy.

```
SQL> CONNECT sec_cdb
```

or

```
SQL> CONNECT sec_pdbc@PDBC
```

```
SQL> NOAUDIT POLICY CDB_pol1;
```

```
SQL> NOAUDIT POLICY PDBC_pol1;
```

2. Drop the audit policy.

```
SQL> DROP AUDIT POLICY  
2  
      CDB_pol1;
```

```
SQL> DROP AUDIT POLICY  
2  
      PDBC_pol1;
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The connected users are granted the `AUDIT_ADMIN` role to drop a unified audit policy. They must disable the audit policy before removing it. If a unified system-related audit policy is already enabled for a session, the effect of dropping a system-related audit policy is not seen by this existing session. Until that time, the unified system-related audit policy's settings remain in effect. However, the effect is immediate for object-related unified audit policies.

In a CDB, you disable and drop a CDB audit policy only from root and a local audit policy only from the PDB to which it applies.

Audit Cleanup

Clean up records in SYS.UNIFIED_AUDIT_TRAIL:

- Schedule an automatic purge job to purge a PDB's records

```
DBMS_AUDIT_MGMT.CREATE_PURGE_JOB
(AUDIT_TRAIL_TYPE=> DBMS_AUDIT_MGMT.AUDIT_TRAIL_UNIFIED,
AUDIT_TRAIL_PURGE_INTERVAL => 12,
AUDIT_TRAIL_PURGE_NAME => 'Audit_Trail_PDB',
USE_LAST_ARCH_TIMESTAMP => TRUE,
CONTAINER => DBMS_AUDIT_MGMT.CONTAINER_CURRENT);
```

- Manually purge the ALL audit records of the CDB

```
DBMS_AUDIT_MGMT.CLEAN_AUDIT_TRAIL(
AUDIT_TRAIL_TYPE => DBMS_AUDIT_MGMT.AUDIT_TRAIL_UNIFIED,
CONTAINER        => DBMS_AUDIT_MGMT.CONTAINER_ALL)
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

To perform the audit trail purge tasks, you use the DBMS_AUDIT_MGMT package. You must have the AUDIT_ADMIN role to use the package. By mandate, Oracle Database audits all executions of the DBMS_AUDIT_MGMT package procedures.

The DBMS_AUDIT_MGMT.CREATE_PURGE_JOB procedure has a new attribute CONTAINER. Setting CONTAINER to CONTAINER_CURRENT deletes audit trail records for the current PDB. Setting CONTAINER to CONTAINER_ALL deletes audit trail records for all PDBs creating a job in the root and the invocation of this job will invoke cleanup in all the PDBs.

Setting USE_LAST_ARCH_TIMESTAMP to TRUE indicates that only audit records created before the last archive time stamp should be deleted. A value of FALSE indicates that all audit records should be deleted. The default value is TRUE. Oracle recommends using this value, as this helps guard against inadvertent deletion of records.

You can automate the cleanup process by creating and scheduling a cleanup purge job, or you can manually run a cleanup purge job. If you manually run cleanup purge jobs, use the DBMS_AUDIT_MGMT.CLEAN_AUDIT_TRAIL procedure with a new type value of DBMS_AUDIT_MGMT.AUDIT_TRAIL_UNIFIED.

Quiz

Select the view that you use to view the enabled audit policies.

- a. DBA_UNIFIED_AUDIT_POLICIES
- b. UNIFIED_AUDIT_ENABLED_POLICIES
- c. UNIFIED_AUDIT_POLICIES
- d. AUDIT_UNIFIED_ENABLED_POLICIES
- e. UNIFIED_AUDIT_TRAIL



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: d

Summary

In this lesson, you should have learned how to:

- Understand the implementation and benefits of using the unified audit trail
- Enable unified auditing
- Configure the unified auditing
- Create and enable audit policies
- Add application context data to audit records
- View data in the unified audit trail
- Clean up the unified audit trail



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Practice: Overview

This practice covers the following topics:

- 21-1: Installing unified auditing option
- 21-2: Creating and enabling auditing policies
- 21-3: Cleaning up audit policies and data
- 21-4: Auditing SYS user
- 21-5: Auditing Data Pump operations
- 21-6: Auditing RMAN operations
- 21-7: Auditing Database Vault violations



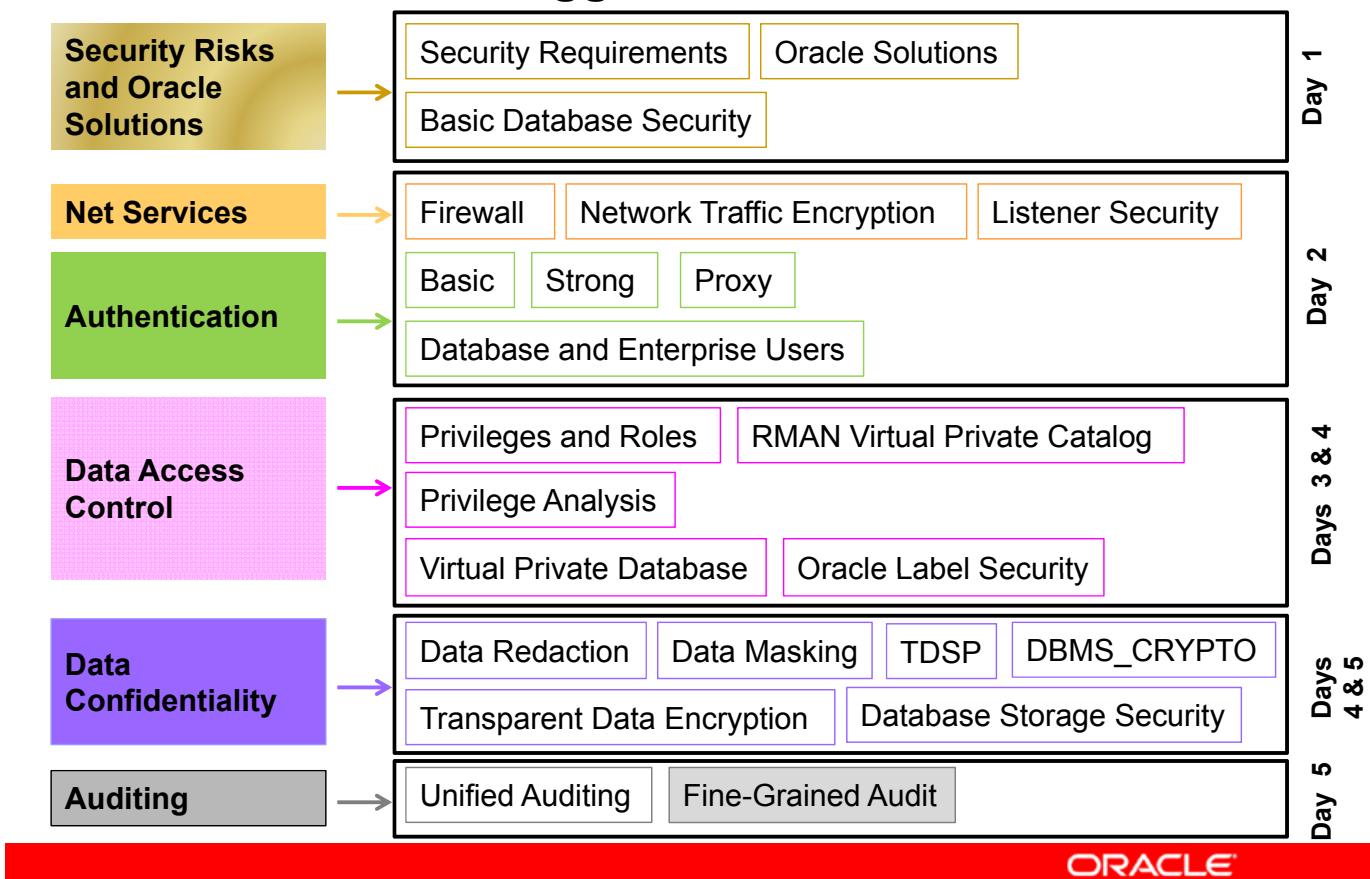
Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Using Fine-Grained Audit

ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Suggested Schedule



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Implement fine-grained auditing (FGA)
- Maintain FGA policies
- Implement an FGA audit event handler
- Read FGA audit events from the FGA audit trail

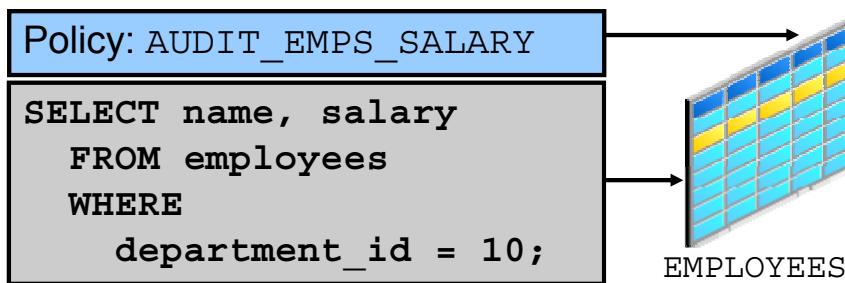


Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Fine-Grained Auditing (FGA)

Policies:

- Monitor data access based on content
- Audit SELECT, INSERT, UPDATE, or DELETE
- Are created on tables or views
- May fire an event handler procedure
- Are administered with the DBMS_FGA package



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Database auditing can record that an operation has occurred, and in extended mode, it records the statement that caused the operation. FGA extends that capability, allowing auditing to be more narrowly focused. FGA captures audit records when certain columns are accessed or when specified conditions are met.

FGA auditing is available only with the Enterprise Edition of Oracle Database 11g.

FGA audit options can be focused on individual columns within a table or view, and can even be specified so that audit records are captured only if certain administrator-defined conditions are met.

The administrator uses the DBMS_FGA PL/SQL package to create an audit policy on the target table or view. If any of the rows returned from a query block matches the audited column and the audit condition, an audit event causes an audit record to be created and stored in the FGA audit trail (FGA_LOG\$). Optionally, the audit event can also execute a procedure. FGA automatically focuses auditing at the statement level. So a SELECT statement that returns thousands of rows generates only one audit record.

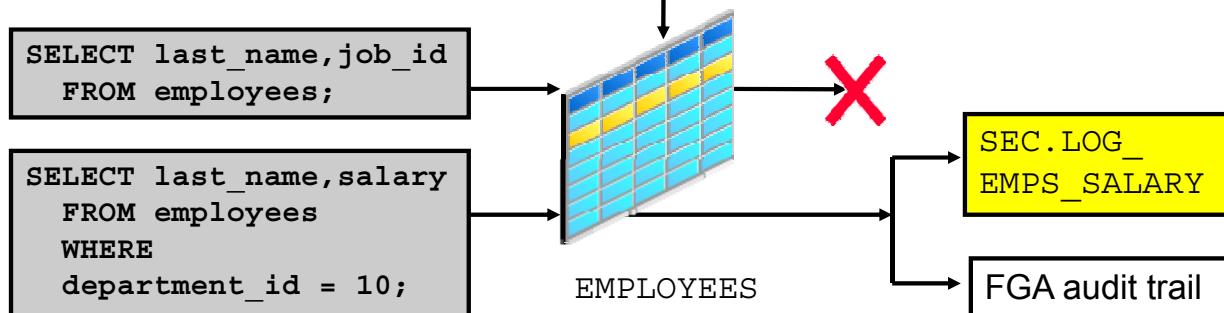
Note: A MERGE statement is not considered to be a single auditable statement. It can perform an INSERT, UPDATE, and sometimes DELETE. So the FGA policies for each of these could fire for one MERGE statement.

FGA Policy

Defines:

- Audit criteria
- Audit action

```
dbms_fga.add_policy (
    object_schema  => 'hr',
    object_name    => 'employees',
    policy_name    => 'audit_emps_salary',
    audit_condition=> 'department_id=10',
    audit_column   => 'salary',
    handler_schema=> 'sec',
    handler_module=> 'log_emps_salary',
    enable         => TRUE,
    statement_types=> 'select' );
```



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The slide illustrates an example where the SEC user uses the DBMS_FGA.ADD_POLICY procedure to create an FGA policy on the HR.EMPLOYEES table. After an FGA policy has been enabled, any SQL statement that matches the audit condition and audit column will generate an audit record.

Note: The SEC user must be granted EXECUTE on the DBMS_FGA package.

The procedure accepts the following arguments:

- **Policy Name:** Each FGA policy is assigned a name when you create it. The example in the slide names the policy AUDIT_EMPS_SALARY, by using the following argument:
policy_name => 'audit_emps_salary'
- **Object:** The object is the table or view that is being audited. It is passed as two arguments:
 - The schema that contains the object
 - The name of the object

In the example in the slide, the HR.EMPLOYEES table is audited by using the following arguments:

```
object_schema => 'hr'
object_name  => 'employees'
```

- **Statement Type:** Which type of statements should be audited? Any combination of SELECT, INSERT, UPDATE, DELETE, or INDEX may be specified as a single comma-delimited string for the statement_types argument—for example, 'SELECT, INSERT, DELETE'. The default is all except INDEX.
- **Audit Condition:** The audit condition is a SQL predicate that defines when the audit event should fire. In the example in the slide, whenever an employee record belonging to department 10 is accessed, an audit event fires due to the audit_condition argument:

```
audit_condition => 'department_id = 10'
```

- **Audit Column:** The audit column defines the data that is being audited. An audit event occurs only if this column is included in the SELECT statement. In the example in the slide, the SALARY column is audited by using the following argument:

```
audit_column => 'salary'
```

This argument is optional. If it is not specified, only the AUDIT_CONDITION argument determines whether an audit event occurs.

- **Handler:** An optional event handler is a PL/SQL procedure that defines any additional actions that should be taken during auditing. For example, the event handler may send an alert page to the administrator. When the audit event occurs, an audit event entry is inserted into the FGA audit trail. If an audit event handler is defined, the audit event handler is executed.

The audit event entry includes the FGA policy that caused the event, the user executing the SQL statement, and the SQL statement and its bind variables.

The event handler is passed as two arguments:

- The schema that contains the PL/SQL program unit
- The name of the PL/SQL program unit

The example in the slide executes the SEC.LOG_EMPS_SALARY procedure by using the following arguments:

```
handler_schema => 'SEC'  
handler_module => 'log_emps_salary'
```

- **Status:** The status indicates whether the FGA policy is enabled. In the example in the slide, the following argument enables the policy:

```
enable => TRUE
```

The FGA policy can generate audit records when the SQL statement does not access the columns or rows targeted. This is called a false positive. This can occur because the evaluation takes place at parse time rather than when the rows are returned. False positives can occur but not false negatives.

Triggering Audit Events

- The following SQL statements cause an audit event:

```
SQL> SELECT count(*)
      FROM hr.employees
     WHERE department_id = 10
       AND salary > &v_salary;
```

```
SQL> SELECT salary
      FROM hr.employees;
```

- The following statement does not cause an audit event:

```
SQL> SELECT last_name
      FROM hr.employees
     WHERE department_id = 10;
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

In general, the audited columns and simple user-defined SQL predicates determine the FGA policy. During parsing, whenever policy conditions are met for a statement, the statement is audited, and if there is an event handler, it is also fired.

The audit function is executed as an autonomous transaction. Each audit policy is applied individually. That is, as long as the rows being returned or changed match any of the audit conditions defined on the table, an audit record is generated, but there is only one record generated for each policy per SQL statement.

Examples:

The first two examples in the slide cause an audit event because they access the `SALARY` column and rows with `DEPARTMENT_ID = 10`. The database server is aware that the second example accesses the rows in department 10, even though `DEPARTMENT_ID` is not referenced in the SQL statement `WHERE` clause. No audit record is created if there are no records for department 10.

The last example does not cause an audit event because the `SALARY` column is not accessed. If the `SALARY` column had not been specified as `AUDIT_COLUMN` when the policy was created, this `SELECT` statement would cause an audit event. For full examples of FGA behavior, refer to My Oracle Support Note 266896.1, “10g: Fine-Grained Auditing.”

Data Dictionary Views

View Name	Description	
DBA_FGA_AUDIT_TRAIL	Pre-12c	All FGA events
UNIFIED_AUDIT_TRAIL	12c	
ALL_AUDIT_POLICIES		All FGA policies for objects that the current user can access
DBA_AUDIT_POLICIES		All FGA policies in the database
USER_AUDIT_POLICIES		All FGA policies for objects in the current user schema



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

If the unified audit is not enabled, the FGA audit entries are logged into a separate table from object and privilege audits. The entries are recorded in the `SYS.FGA_LOG$` table and `DBA_FGA_AUDIT_TRAIL` view.

If the unified audit is enabled, the FGA audit entries are logged into the single unified audit table with all object and privilege audits. The entries are recorded in the `UNIFIED_AUDIT_TRAIL` view.

The other three views listed in the slide contain policy definitions.

DBA_FGA_AUDIT_TRAIL / UNIFIED_AUDIT_TRAIL

```
SQL> SELECT to_char(timestamp, 'YYMMDDHH24MI')
  2          AS timestamp,
  3          db_user,
  4          policy_name,
  5          sql_bind,
  6          sql_text
  7      FROM dba_fga_audit_trail;

TIMESTAMP    DB_USER  POLICY_NAME           SQL_BIND
-----
SQL_TEXT
-----
0808272222 HR        AUDIT_EMPS_SALARY #1(4):1000
SELECT COUNT(*) FROM HR.EMPLOYEES WHERE
DEPARTMENT_ID = 10 AND SALARY > :B1

0808272222 HR        AUDIT_EMPS_SALARY
SELECT salary
FROM hr.employees
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Selecting from the FGA Audit Trail:

The slide displays the two audit rows created by the valid examples from the previous page. The SQL_BIND column has a value of #1(4):1000, which includes the following components:

- #1 indicates that this is the first bind variable in the statement.
- (4) indicates that the bind variable has a length of 4.
- 1000 indicates that the bind variable has a value of 1000.

Note: This example shows a partial list of columns available in DBA_FGA_AUDIT_TRAIL. For more information, see the Oracle Database Reference Guide.

Quiz

Which of the following types of auditing must you implement to audit the access of a specific column?

- a. SQL statement auditing
- b. Object privilege auditing
- c. Fine-grained auditing



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Answer: c

DBMS_FGA Package

- Use DBMS_FGA to maintain FGA policies.
- Grant the EXECUTE privilege only to administrators.
- The DBMS_FGA package includes the following subprograms:

Subprogram	Description
ADD_POLICY	Creates an audit policy by using the supplied predicate as the audit condition
DROP_POLICY	Drops an audit policy
ENABLE_POLICY	Enables an audit policy
DISABLE_POLICY	Disables an audit policy



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

The DBMS_FGA package is the tool to manage FGA functions. The EXECUTE privilege on DBMS_FGA is needed to administer FGA policies. Because the FGA audit trail can contain sensitive information, the EXECUTE privilege on this package must be reserved only for administrators.

To audit usage of the DBMS_FGA package, wrap it with a PL/SQL package that captures user information and then executes the DBMS_FGA procedures. Grant EXECUTE on the wrapper to users that need it. Revoke EXECUTE from the DBMS_FGA package except to the owner of the wrapper package. The PL/SQL wrapper package must use definer's rights.

Enabling / Disabling Dropping an FGA Policy

- Enable a policy: dbms_fga.enable_policy

```
dbms_fga.enable_policy (
    object_schema => 'hr',
    object_name   => 'employees',
    policy_name   => 'audit_emps_salary' );
```

- Disable a policy: dbms_fga.disable_policy
- Dropping a policy: dbms_fga.drop_policy

Note: All procedures use the same parameters



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Enable an FGA to generate audit events. The policy will not generate any log events when it is disabled. By default, the policy is enabled when it is created. The example in the slide shows how to enable and disable a policy. For both procedures, all arguments are required.

FGA Policy Guidelines

Setting policy parameters

- Audit conditions
 - To audit all statements, use a NULL or TRUE condition.
 - If the audit condition syntax is invalid, an ORA-28112 error is raised when the audited object is accessed.
- Audit columns
 - If audit column is set to NULL, all columns are audited.
 - If the audit column name is valid but incorrect, the wrong statements are audited.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Audit Conditions

When creating a new FGA policy, the audit condition defaults to null. An audit condition of null, or a condition that always evaluates to TRUE such as 1=1 are equivalent. Either means that all statements will be audited, including statements resulting in no rows selected. FGA with null (audit all) can be extremely useful in capturing SQL to determine whether tuning (or developer training) is needed. It can also help troubleshoot data corruption from GUI applications by capturing the data manipulation language (DML) statements plus bind variables.

If the audit condition has an invalid syntax, the policy is created without an error. But the following error is raised when the audited object is accessed:

ORA-28112: failed to execute policy function

If the audit condition has a valid syntax, but is incorrect, the wrong rows are audited.

Audit Columns

When the audit column is null, all columns are audited.

If the audit column name is valid but incorrect, the wrong statements are audited.

FGA Policy Errors

- Policy creation errors occur when:
 - The audited table or view does not exist
 - The policy already exists; error ORA-28101 is raised
 - The audit column does not exist
- Audited SQL statements fail when:
 - The audit condition is invalid
 - The event handler does not exist or is invalid



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Errors can occur at the time the policy is created, or when the policy is invoked. Some error policy configuration errors do not appear until an audited SQL statement is executed.

Policy Creation Errors

- **Audited-Object Errors:** The audited table or view must exist when you create the policy. If it does not, you receive the following error:
ORA-00942: table or view does not exist
- **Policy-Name Error:** Policy names must be unique within the database. They have no owner. If a duplicate name is used, you receive the following error when creating the policy:
ORA-28101: policy already exists

- If the audit column does not exist, the ADD_POLICY procedure fails.

Audited SQL Statement Errors

- **Invalid Audit Condition:** An audit condition with invalid syntax causes the SQL statement to fail. No audit record can be created.
- **Invalid Event Handler:** The policy is created even when the policy references a nonexistent or invalid event handler. The SQL statement will fail when the handler is accessed. Unhandled exceptions in the handler cause the SQL statement to fail. The policy is still enforced and the audit records are created.

Maintaining the Audit Trail

Maintaining the audit trail must include:

- Reviewing and storing old records
- Preventing storage problems
- Avoiding loss of records



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Each type of audit trail must be maintained. Basic maintenance includes reviewing the audit records and removing older records from the database or operating system. Audit trails can grow to fill the available storage. If the file system is full, the system may crash or exhibit performance problems. If the database audit trail fills the tablespace, audited actions do not complete. If the audit trail fills the system tablespace, the performance of other operations are affected before audit operations halt.

The audit trail for standard auditing is stored in the `AUD$` table. The audit trail for FGA auditing is the `FGA_LOG$` table. Both these tables are created in the `SYSTEM` tablespace.

Audit records can be lost during the process of removing records from the audit tables. A copy operation that is followed by a delete or truncate operation can remove records that are not captured by the copy operation. A better method is to perform an export of records that is based on a time stamp, and then delete those records that have the same (or an earlier) time stamp.

Summary

In this lesson, you should have learned how to:

- Implement FGA
- Maintain FGA policies
- Implement an FGA audit event handler
- Read FGA audit events from the FGA audit trail



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Practice 22 Overview: Using Fine-grained Audit

This practice covers the following topics:

- 22-1: Creating an FGA policy
- 22-2: Observing audited records in the database audit table
- 22-3: Creating an FGA policy with an event handler



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Error : You are not a Valid Partner use only

Implementing Virtual Private Database Policy Groups



ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to group policies.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Note

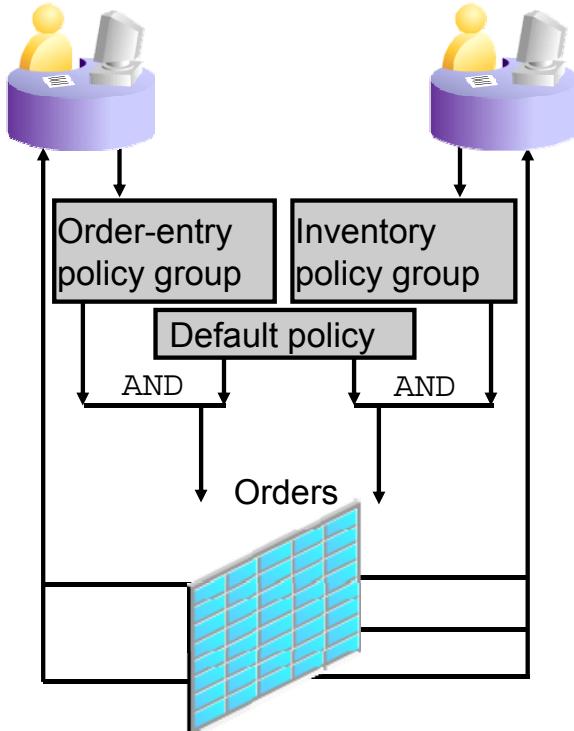
For a complete understanding of VPD grouped policies, refer to the *Tutorial: Implementing an Oracle Virtual Private Database Policy Group* tutorial in the Oracle documentation:

- *Oracle Database Security Guide 12c Release 1*

For more information about the topic covered in the lesson, read the Oracle By Example:

- *Using Virtual Private Database Policy Groups* under Oracle Learning Library

Implementing Policy Groups



- Security policies are application-driven.
- Different policies apply, depending on the active driving context.
- Policies can be developed independently.
- The default policy always applies.
- Policy groups are useful when hosting applications.

ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

By default, all policies defined on a table or view are applied to all SQL statements. The policies written for one application generate predicates for other applications. This requires development groups to collaborate on policies. Partitioned fine-grained access control by using policy groups allows application-driven security policies.

With policy groups, applications can have different security policies based on their individual application needs.

You can use policy groups and a driving application context to partition fine-grained access control enforcement so that different policies apply for each application that is accessing the data. The application sets the active driving context so that only the policy group for that application is applied.

In the example, an order-entry clerk in Company B accesses the ORDERS table through the order-entry application. The order-entry application sets a driving context. Inside the database, a driving context sets the policy group to Order Entry (which enforces access control on customer number and region). There is also a default policy group that enforces data separation for each company because this is a hosted application.

An inventory manager in Company C connects to the inventory application that accesses the same ORDERS table. The driving context sets the inventory manager's policy group to Inventory, which enforces access control according to part number. The default policy group that enforces data separation for each company is still in effect because this is a hosted application.

Each user accesses different data, based on entirely different policies. The security enforcement is transparent to the user—for example, each user may issue the SELECT * from ORDERS SQL statement, but entirely different data sets are returned. With the partitioning of FGAC policies, administrators can specify which policy group the policy belongs to when adding a policy to a table or view by using the ADD_POLICY_TO_GROUP interface.

Developing Policies in a Hosting Environment

In this example, in a hosting environment, Company A can host the ORDERS table for Company B and Company C. The hosting company puts a default policy in place that is always in effect. This policy applies a predicate that allows users from Company B to access only their own data, and not any data owned by Company C.

The ORDERS table is accessed by two different applications (order entry from Company B and inventory from Company C) with two different security policies. To integrate these two policies into the ORDERS table without partitioned fine-grained access would require joint development of policies between the two companies, which is not a feasible option. By defining an application context to drive the enforcement of a particular set of policies to the base objects, each application can implement a private set of security policies.

Grouping Policies

1. Determine the default policies.
2. Set up a driving context for each table:
 - a. Create the context.
 - b. Create the function that sets the context.
 - c. Make the context the driving context.
3. Create a policy group for each application.
4. Add each policy to the appropriate group.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Because multiple applications, with varying security policies, can share the same table or view, it is important to identify which policies should be in effect when the table or view is accessed for each application.

To do this, you can organize security policies into groups. By referring to the driving application context, the Oracle server determines which group of policies should be in effect at run time. The server enforces all policies that belong to that policy group.

The slide outlines the steps to create and implement grouped policies.

Default Policy Group

- A predefined default policy group is always applied.
- The default group is named `SYS_DEFAULT`.
- Each object has a default group.



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Default Policy Always Applies

Some policies should always be applied. These policies belong to the default group.

SYS_DEFAULT Group

There is a predefined policy group named `SYS_DEFAULT`. Policies defined in this policy group for a particular table or view are always executed along with the policy group specified by the driving context.

Default Group

By default, all policies belong to the `SYS_DEFAULT` policy group. The policies defined in this group for a particular table or view are always executed along with the policy group specified by the driving context. The `SYS_DEFAULT` policy group may or may not contain policies.

For example, in a hosting environment, the policy that restricts the customers to seeing their own rows may be in this group.

Multiple Policies in the SYS_DEFAULT Policy Group

If you add policies that are associated with two or more objects to the SYS_DEFAULT policy group, each object has a separate SYS_DEFAULT policy group associated with it.

For example, the EMP table in the SCOTT schema has one SYS_DEFAULT policy group, and the DEPT table in the SCOTT schema has a different SYS_DEFAULT policy group associated with it.

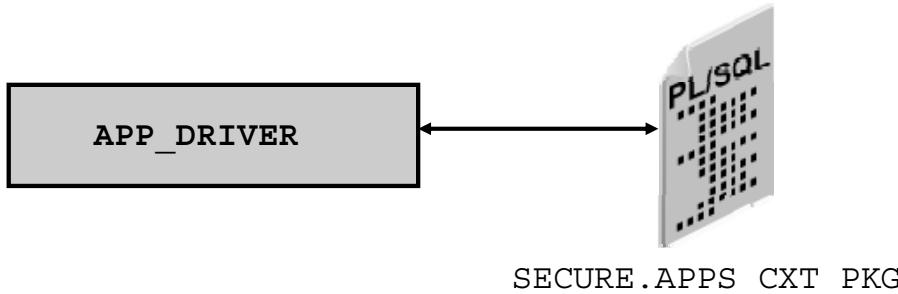
Creating a Driving Context

- Create the context:

```
CREATE CONTEXT app_driver  
    USING secure.apps_ctxt_pkg;
```

- Create the procedure that sets the context:

```
CREATE OR REPLACE PACKAGE BODY secure.apps_ctxt_pkg  
PROCEDURE set_driver( policy_group VARCHAR2) ...
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

A driving application context is a context that holds at least one attribute. This attribute is used to determine which group of policies will be applied. The driving application context is set by the application that is accessing the data. The Oracle data server references the driving application context to determine which policy group to apply.

Create a context and a trusted package to set the attributes in the same way as described in the lesson titled “Using Application Contexts.” The driving context holds the name of the active application context. The name of the attribute is arbitrary, but it is specified in the ADD_POLICY_CONTEXT procedure. The value returned from the driving context attribute defines the driving context of the policy.

The following steps can be performed in any order.

Creating a Context

Create a namespace for the driving context and associate it with a trusted package. For example:

```
CREATE CONTEXT app_driver USING secure.apps_ctxt_pkg;
```

Note: You can use the

/home/oracle/labs/demos/demo_fgac_driving_context.sql to exhibit the example of the slides 8 to 13.

Creating the Procedure That Sets the Context

Create the package that administers the driving context. In this procedure, determine the current application. Techniques for determining the application include the following:

- If the application server determines the application, check the proxy.
- If the user must be in an application table to use the application, set the application only after verifying that the user exists in that table.

In this example, the context is set to the parameter passed to the procedure:

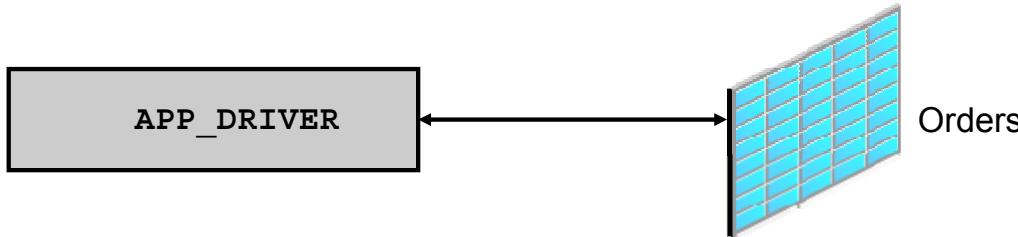
```
CREATE OR REPLACE PACKAGE BODY secure.apps_ctxt_pkg
  PROCEDURE set_driver ( policy_group varchar2 )
  BEGIN
    DBMS_SESSION.SET_CONTEXT
      ('APPS', 'ACTIVE_APP', policy_group );
  END;
END;
```

- If the driving context is a policy group with policies, all enabled policies from that policy group are applied, along with all policies from the SYS_DEFAULT policy group.

Making the Context a Driving Context

Associate the driving context with a table:

```
dbms_rls.add_policy_context(  
    object_schema =>'OE',  
    object_name => 'ORDERS' ,  
    namespace => 'APP_DRIVER',  
    attribute => 'ACTIVE_APP')
```



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Associating the Driving Context with a Table

The ADD_POLICY_CONTEXT procedure in the DBMS_RLS package makes a context a driving application context. The driving context enforces the policies. It is the driving context that determines which application is running.

When using this procedure, consider the following guidelines:

- The driving context can be local or accessed globally.
- If the driving context is NULL, policies from all policy groups are used.
- There must be at least one driving context defined for each object that has FGAC policies; otherwise, all policies for the object are executed.
- Adding multiple contexts to the same object causes policies from multiple policy groups to be enforced.

Making the Context a Driving Context

Define the driving context for the ORDERS table:

```
DBMS_RLS.ADD_POLICY_CONTEXT  
( 'OE' , 'ORDERS' , 'APP_DRIVER' , 'ACTIVE_APP' );
```

Modify the function that sets the predicate. The driving context is set after the user is authenticated as a valid user for the application.

```
CREATE OR REPLACE PACKAGE oe_context IS
    PROCEDURE set_cust_id;
END;
/
CREATE OR REPLACE PACKAGE BODY oe_context IS
    PROCEDURE set_cust_id
    IS
        v_cust_id NUMBER;
    BEGIN
        SELECT customer_id INTO v_cust_id FROM oe.customers
        WHERE cust_first_name || '_' || cust_last_name =
            SYS_CONTEXT('USERENV', 'SESSION_USER');
        DBMS_SESSION.SET_CONTEXT('oe', 'cust_id', v_cust_id);
        apps_context.set_driver('OE_GRP');          -- set the driver
    EXCEPTION
        WHEN no_data_found THEN
            apps_context.set_driver('XX'); -- set the driver
    END;
END;
/
```

The OE_CONTEXT.SET_CUST_ID procedure is called when the order-entry application is entered. If the user is a valid customer, the OE.CUST_ID and APPS.ACTIVE_APP context attributes are set. If not, the driver is set to an invalid driving context that causes an error, preventing the user from accessing any rows.

Instead of an invalid context, you can create a driving context that references a group containing a policy that always returns the 1=2 predicate, which is always FALSE, preventing access to any rows.

Creating a Policy Group

- Create the OE_GRP group:

```
dbms_rls.create_policy_group(
    object_schema =>'OE',
    object_name  => 'ORDERS',
    policy_group => 'OE_GRP' );
```

- Create the AC_GRP group:

```
dbms_rls.create_policy_group
( 'OE', 'ORDERS', 'AC_GRP' );
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Each policy group must have a unique name for each table. No table can have two policy groups with the same name. The CREATE_POLICY_GROUP procedure of the DBMS_RLS package creates the policy group and associates it with a table or a view.

This example creates the OE_GRP and AC_GRP groups:

- DBMS_RLS.CREATE_POLICY_GROUP('OE', 'ORDERS', 'OE');
- DBMS_RLS.CREATE_POLICY_GROUP('OE', 'ORDERS', 'AC');

Syntax

```
PROCEDURE create_policy_group (
    object_schema    IN      VARCHAR2 := NULL,
    object_name     IN      VARCHAR2 := NULL,
    policy_group    IN      VARCHAR2 );
```

where:

- OBJECT_SCHEMA is the owner of the table or view that the policy group is applied to; the current user is the default.
- OBJECT_NAME is the name of the table or view that the policy group is applied to.
- POLICY_NAME is the name of the policy group.

Adding a Policy to a Group

1. Add the OE_SECURITY policy to the OE_GRP group:

```
dbms_rls.add_grouped_policy (
    object_schema    => 'oe',
    object_name      => 'orders',
    policy_group     => 'oe_grp',
    policy_name      => 'oe_security',
    function_schema  => 'sec',
    policy_function  => 'oe_context');
```

2. Add the AC_SECURITY policy to the AC_GRP group:

```
dbms_rls.add_grouped_policy (
    'oe', 'orders', 'ac_grp', 'ac_security',
    'sec', 'ac_context');
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

When you create a policy that is part of a policy group, use the ADD_GROUPED_POLICY procedure of the DBMS_RLS package, instead of the ADD_POLICY procedure.

The first step in the slide adds a policy named OE_SECURITY to the OE_GRP policy group. The SEC.OE_CONTEXT function returns the predicate to enforce security on the OE.ORDERS table.

The second step adds a policy named AC_SECURITY to the AC_GRP policy group. The SEC.AC_CONTEXT function returns the predicate to enforce security on the OE.ORDERS table.

You must also create the appropriate security functions.

ADD_GROUPED_POLICY

This procedure adds a policy for a table or view and associates it with a policy group. The policy group must have been created by using the CREATE_POLICY_GROUP procedure. The policy name must be unique within a policy group for a specific object.

It has the following specification:

```
DBMS_RLS.ADD_GROUPED_POLICY(  
    object_schema      VARCHAR2,  
    object_name        VARCHAR2,  
    policy_group       VARCHAR2,  
    policy_name        VARCHAR2,  
    function_schema    VARCHAR2,  
    policy_function    VARCHAR2,  
    statement_types    VARCHAR2,  
    update_check       BOOLEAN,  
    enabled            BOOLEAN,  
    static_policy      IN BOOLEAN FALSE,  
    policy_type        IN BINARY_INTEGER NULL,  
    long_predicate     IN BOOLEAN FALSE,  
    sec_relevant_cols IN VARCHAR2);
```

The DBMS_RLS package contains the following procedures for managing grouped policies:

- ADD_GROUPED_POLICY
- DROP_GROUPED_POLICY
- ENABLE_GROUPED_POLICY
- DISABLE_GROUPED_POLICY
- REFRESH_GROUPED_POLICY
- CREATE_POLICY_GROUP
- DELETE_POLICY_GROUP

For the specifications for these procedures and usage notes, refer to the *PL/SQL Packages and Types Reference* manual.

Summary

In this appendix, you should have learned how to:

- Group policies



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Error : You are not a Valid Partner use only

b Encrypting Network Traffic

ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Encrypting Network Traffic

- Guideline: Encrypt sensitive network traffic.
- Use the TCPS protocol for TCP/IP with SSL:

```
...
(ADDRESS=
  (PROTOCOL=tcps)
...
```



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Although you may want to avoid the overhead of encrypting and decrypting internal network communication, remember that 70% of security violations are from internal sources. Both SSL and Oracle Net native encryption encrypts all network traffic between a client and a server. Although the SSL solution also provides authentication, it requires the use of certificates. Oracle Net native encryption provides end-to-end encryption in an n-tier environment without a certificate.

Encrypting Sensitive Client-Server Communication with HTTPS

Use the HTTPS protocol to encrypt sensitive data passed between the client computer and the HTTP server.

Encrypting Oracle Net Services Traffic

Use Oracle Net Services to encrypt network traffic between client computers, databases, and application servers.

Oracle Net Services provides data encryption and integrity for all network protocols into the Oracle database, including Oracle Net with native encryption, Oracle Net/SSL, Internet Inter-ORB Protocol (IIOP)/SSL, and Java-based encryption for thin Java Database Connectivity (JDBC) clients.

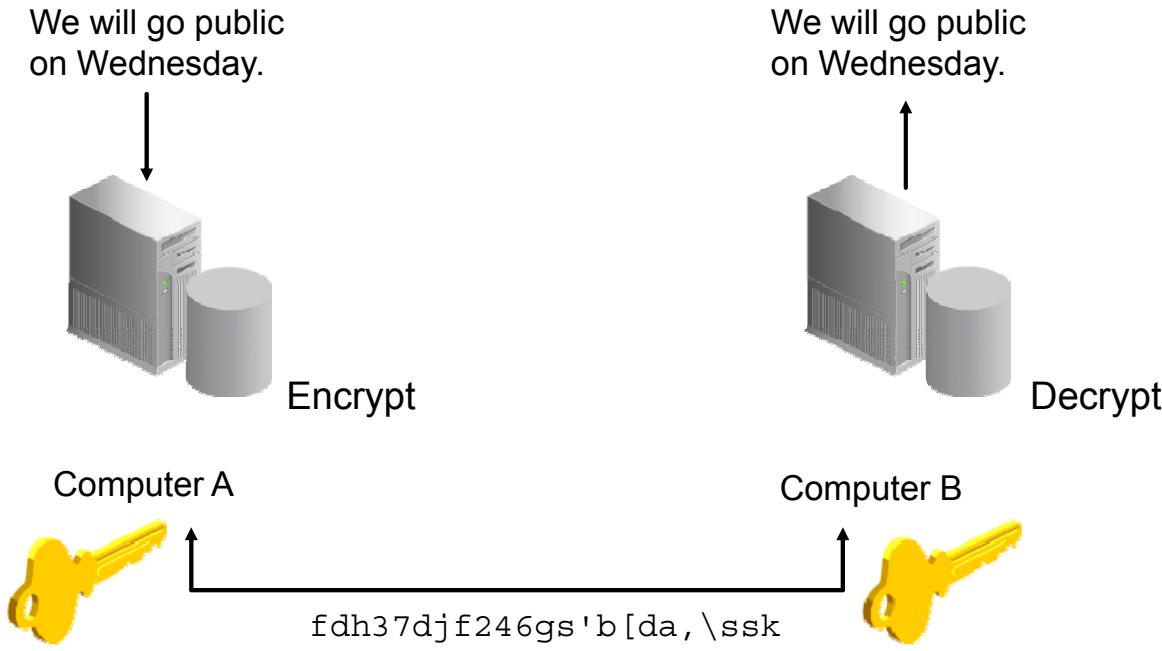
Encrypting Transparent Gateway Traffic

If you are using a transparent gateway to interface to another database, network communications that use Oracle Net Services can be encrypted.

The client computer communicates with the Oracle instance by using Oracle Net Services, the Oracle instance communicates with the transparent gateway by using Oracle Net Services, and the gateway communicates with the other database over an encrypted network connection or a memory-to-memory connection.

How the transparent gateway communicates with the other database is dependent on the options offered by that database vendor.

End-to-End Encryption



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Encryption Techniques

Oracle Network Services supports the following common encryption methods:

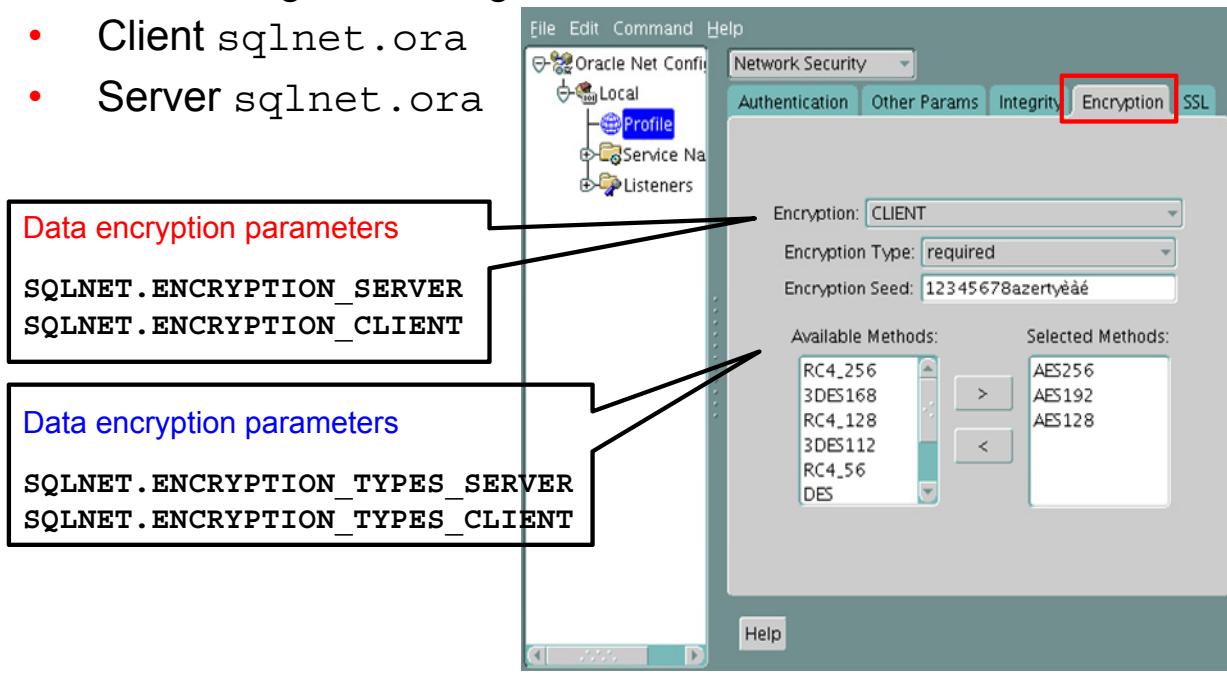
- DES: (not recommended) The U.S. Data Encryption Standard (DES) algorithm uses symmetric key cryptography.
- 3DES: Triple-DES Encryption encrypts message data with three passes of DES.
- RSA RC4: The RC4 algorithm, developed by RSA Security, Inc., uses a secret, randomly generated key that is unique to each session.
- AES: Advanced Encryption Standard (AES) is a new cryptographic algorithm that uses cipher keys with lengths of 128, 192, and 256 bits.

Network encryption provides varying levels of security and performance for different types of data transfers. Longer key length provides stronger encryption and is harder to break, but uses more resources during encryption and decryption.

Configuring Network Encryption

Use Net Manager to configure:

- Client sqlnet.ora
- Server sqlnet.ora



Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

ORACLE

Use Network Manager to configure the sqlnet.ora parameters for network encryption on both the client and server. These parameters may be set manually in the sqlnet.ora file.

The parameters set for the client are SQLNET.ENCRYPTION_CLIENT and SQLNET.ENCRYPTION_TYPES_CLIENT. The default values for client encryptions allow clients to connect however the server is configured.

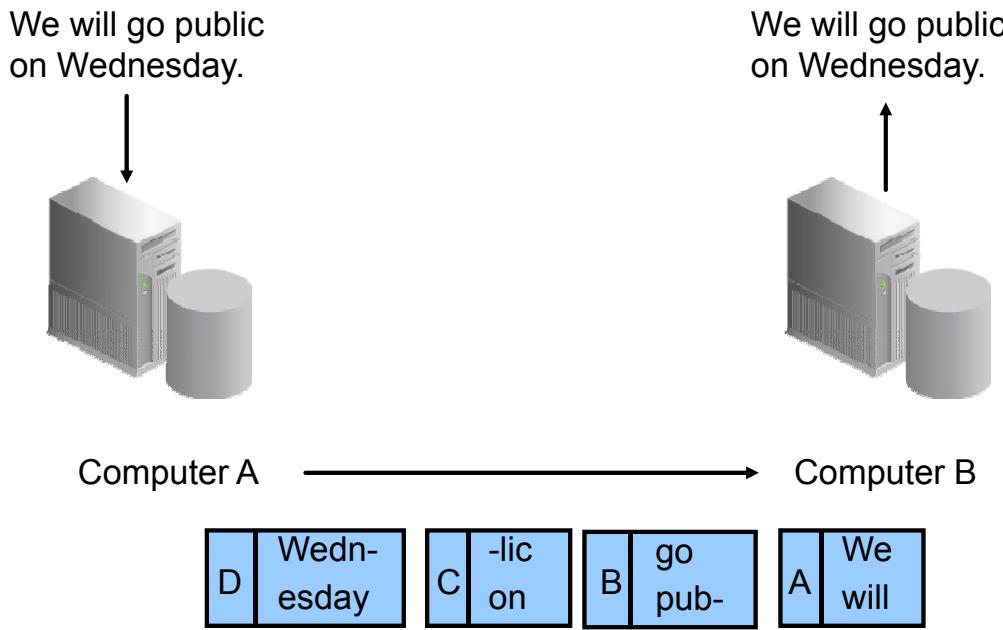
The parameters for the server are similar: SQLNET.ENCRYPTION_SERVER and SQLNET.ENCRYPTION_TYPES_SERVER.

ENCRYPTION_SERVER and ENCRYPTION_CLIENT have four valid values: accepted, rejected, requested, and required. The default value for both is accepted. If the server specifies required and the client is not set to rejected, the encrypted connection will be attempted.

ENCRYPTION_TYPES lists the allowed encryption algorithms. At least one of the types listed on both the client and the server must be the same for the encrypted connect to be set up. The server list is used to find an available algorithm on the client. If no algorithm is specified on the server, all installed algorithms are tried.

For more information about the interaction of these parameters, see the *Oracle Database Security Guide*.

Checksumming



ORACLE®

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Checksumming ensures the integrity of data packets. A checksum or digest is produced by applying the checksumming algorithm to all the data. The same algorithm is applied at the packet destination. If the checksum produced has a different value, the data has been changed during transmission.

To ensure the integrity of data packets during transmission, the checksumming algorithm generates a cryptographically secure message digest and includes it with each message sent across a network. Data integrity algorithms add little overhead and protect against the following attacks:

- Data modification
- Deleted packets
- Replay attacks

Oracle Net Services supports common industry-standard algorithms:

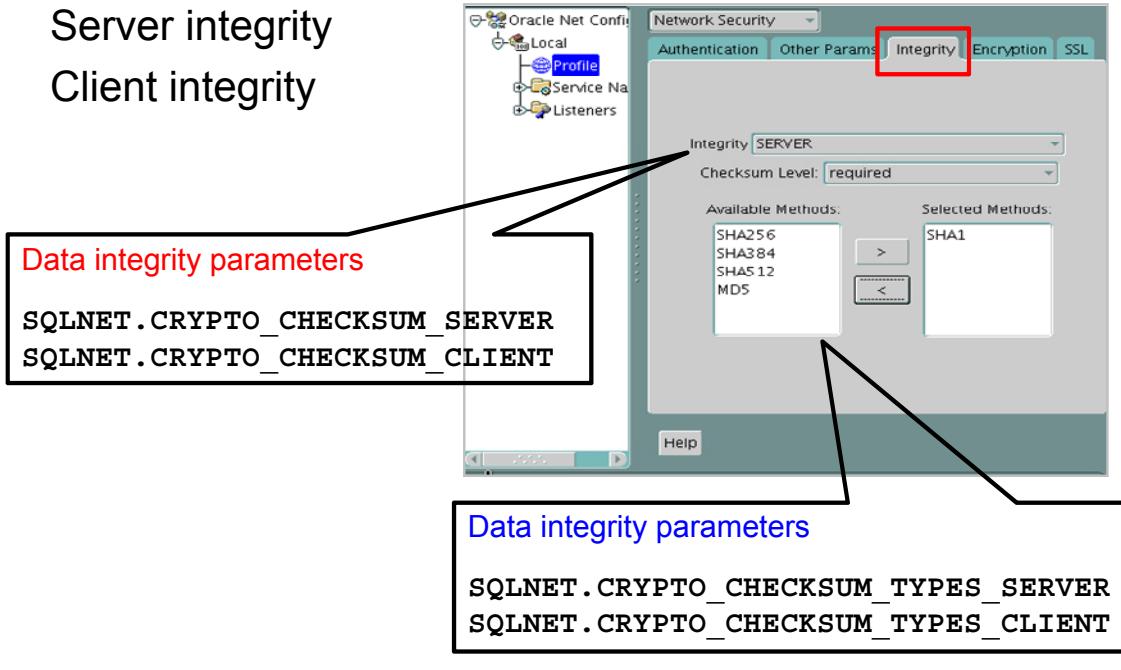
- MD5
- SHA-1

SHA-1 is slightly slower than MD5, but produces a larger message digest to make it more secure against brute-force collision and inversion attacks.

Configuring Checksumming

Use Oracle Net Manager to configure:

- Server integrity
- Client integrity



ORACLE

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Use Oracle Net Manager to configure the `sqlnet.ora` parameters on the client and server that control data integrity (checksumming). These parameters can be set manually in the `sqlnet.ora` file.

The parameters for the server are `SQLNET.CRYPTO_CHECKSUM_SERVER` and `SQLNET.CRYPTO_CHECKSUM_TYPES_SERVER`.

The parameters for the client are `SQLNET.CRYPTO_CHECKSUM_CLIENT` and `SQLNET.CRYPTO_CHECKSUM_TYPES_CLIENT`.

The `CRYPTO_CHECKSUM_CLIENT` and `CRYPTO_CHECKSUM_SERVER` have four valid values: accepted, rejected, requested, and required. The default value for both is accepted. If the server specifies required and the client is not set to rejected, the checksumming on the connection will be attempted.

On both the client and server, `CRYPTO_CHECKSUM_TYPES` determines the algorithms that will be tried. The server will try each algorithm in order until a matching algorithm on the client is found. If no matching algorithm is found, the connect will revert to no checksumming, or fail depending on the setting of `CRYPTO_CHECKSUM` on both. If no algorithms are specified, all installed algorithms will be tried.

For more information about the interaction of these parameters, see the *Oracle Database Security Guide*.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Error : You are not a Valid Partner use only

General Security Reports

ORACLE®

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Using Database Vault Reports

Oracle Database Vault provides two categories of security reports:

- General Security Reports: Privileges, roles, profiles, accounts, initialization parameters, audit, and so on
- Database Vault Reports: Potential configuration issues and violations
 - Configuration Issues Reports
 - Enforcement Audit Reports
 - Database Vault Configuration Changes Audit Reports



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In any system that has security requirements, reports are needed to support the evaluation of the security measures that have been implemented. Database Vault provides a selection of reports that display security-related information from the database. These reports also show custom Database Vault audit event information. If you have unified auditing enabled, then the reports capture the results of your unified audit policies.

The reports are in two categories:

- **General Security Reports:** These reports allow you to check the status of object privileges, database account system privileges, sensitive objects, privilege management, powerful database accounts and roles, initialization parameters, profiles, account passwords, security audits, and other security vulnerability reports.
- **Database Vault Reports:** These reports allow you to check configuration issues with realms, command rules, factors, factor identities, rule sets, and secure application roles. These reports also reveal realm violations, auditing results, and so on. They are grouped in Cloud Control, as shown on the slide.

The appendix covers the General Security Reports only.

General Security Reports

- Cloud Control navigation: Security (main menu) > Reports
- Groups of security-related reports:
 - Database Account Password Reports
 - Privileged Database Accounts and Roles Reports
 - Initialization Parameter and Operating System Directory Permission Reports
 - General Database Privilege and Resource Profile Reports
 - Database Audit and Privilege Reports
 - Object Privilege Reports
 - Sensitive Objects Reports
 - Unified Audit Trail
 - Other Security Reports



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

These reports enable you to check the status of object privileges, database account system privileges, sensitive objects, privilege management, powerful database accounts and roles, initialization parameters and profiles, account passwords, security audits, and other security vulnerability reports. You are required to use Cloud Control, SQL*Plus, or a SQL-based command-line tool to correct many of the issues shown by the general security reports.

The groups of reports are listed in the slide.

Database Account Password Reports

- Database Account Default Password Report
 - Change default passwords
- Database Account Status Report
 - Lock accounts for special schemas
 - Confirm the use of your organizations tablespaces, special passwords and resource profiles (if applicable)



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

This group of reports focuses on password issues. Most of these issues are prevented by implementing a reasonable password policy. Database Vault provides a password policy by default.

- The **Database Account Default Password Report** lists the database accounts that have default passwords. Default passwords are provided during the Oracle Database installation.
- The **Database Account Status Report** provides a quick view of account status for each account, which helps you identify accounts that must be locked or accounts not using organizationally defined default tablespaces or accounts that use external passwords or accounts that are not using special password and resource secure profile (if defined).

Using Powerful Database Accounts and Roles Reports

Each report focuses on a specific vulnerability, such as:

- The ability to grant privileges
- The ability to override protections
- The ability to gain access to another user's objects
- The ability to change the system or session configuration
- The knowledge to more easily guess or crack passwords
- The ability to modify audit records or auditing options
- The ability to export sensitive or protected information to the OS



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Each report in this group focuses on a specific vulnerability, but all these issues have the common thread of giving a user the knowledge or ability to perform an unauthorized action. Some of these reports have to do with privileges that are one level away from the unauthorized action, for example, the AUDIT privileges report. This report shows users that can disable audit options and remove audit records. A user with these privileges could alter audit records, potentially hiding actions that should be audited.

Privileged Database Accounts and Roles Reports

- Database Accounts With EXEMPT ACCESS POLICY Privilege
- Database Accounts With BECOME USER Privilege
- Database Accounts With ALTER SYSTEM OR ALTER SESSION Privileges
- Database Accounts With CATALOG Roles
- Database Accounts With Privileged Roles
- Database Accounts With ANY System Privilege



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

- The Database Accounts With The EXEMPT ACCESS POLICY Privilege Report shows database (but not Oracle Database Vault) accounts and roles that have the EXEMPT ACCESS POLICY system privilege granted to them. Accounts that have this privilege can bypass all Virtual Private Database (VPD) policy filters and any Oracle Label Security policies that use Oracle Virtual Private Database indirectly.
- The Database Accounts With BECOME USER Privilege Report shows all database accounts roles that have the BECOME USER system privilege.
- Database Accounts With ALTER SYSTEM OR ALTER SESSION Privileges Report shows all database accounts and roles that have the ALTER SYSTEM or ALTER SESSION privilege. Oracle recommends that you restrict these privileges only to those accounts and roles that truly need them.
- The Database Accounts With CATALOG Roles Report displays all database accounts and roles that have the %CATALOG% roles granted to them.
- The Database Accounts With Privileged Roles report shows the Database Accounts With Password File Authentication. These are the accounts with special system privileges like SYSDBA/SYSOPER and are authenticated using a password file. Note: Operating system authentication takes precedence over password file authentication if you are a member of the OSDBA or OSOPER group.
- Database Accounts With ANY System Privilege Report shows all ANY system privileges granted to the specified database account or role.

Initialization Parameter and Operating System Directory Permission Reports

- Security Related Database Parameter Settings In init(sid).ora File
- Permissions On Operating System Directory Objects



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

- The Security Related Database Parameter Settings In init(sid).ora File Report displays database parameters that can cause security vulnerabilities, if not set correctly. This report can be used to compare the recommended settings with the current state of the database parameter values.
- The Permissions On Operating System Directory Objects Report shows all directory objects that exist in the database, whether they are available to PUBLIC, and what their privileges are. Directory objects should exist only for secured operating system (OS) directories, and access to them within the database should be protected.

Review Privilege Reports

Privilege reports give essential information for:

- Applying the principle of least privilege
- Reducing dependence on PUBLIC
- Removing the requirement to grant the DBA role
- Removing access to sensitive objects
- Controlling object access



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Privileges are the primary method of access control in the Oracle database. The system and object privileges provide security for most objects in a database. The four groups of privilege reports—object, system, sensitive, and by grantee—provide you with tools to apply the principle of least privilege. Grant only the minimum privilege required for a user to do the assigned task.

Applying the principle of least privilege is difficult without the knowledge of what objects must be accessed and how the privileges are currently granted to the user. There are over 21,000 object grants to PUBLIC in the 11g version of the Oracle Database. Granting privileges to public grants these privileges to every user, even to many that may not need them. The default grants to PUBLIC are generally safe, but this cannot be said for the grants to the DBA role. The DBA role has object and system privileges that make it dangerous in the hands of untrusted users. Any user granted the DBA role should be thoroughly vetted and should also be subject to auditing. If you know the grants that a particular user requires, you can use a role that grants just those privileges instead of the DBA role. This reduces the insider threat.

There are sensitive objects in the database that can provide information about users, roles, and privileges. There are powerful PL/SQL packages. Access to these objects and EXECUTE privilege on these packages should not be available to PUBLIC, but only to users who need to know.

General Database Privilege and Resource Profile Reports

- Privileges Distribution By Grantee
- Privileges Distribution By Grantee, Owner
- Privileges Distribution By Grantee, Owner, Privilege
- Roles And Accounts That Have A Given Role
- Resource Profiles
- System Resource Limits



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The following reports display the count of privileges granted to a database account or role with various groupings. These can provide an insight into accounts and roles that may have powerful privileges.

- The Privileges Distribution By Grantee Report displays the count of privileges granted to a database account or role. This provides insight into accounts and roles that may have powerful privileges.
- The Privileges Distribution By Grantee, Owner Report displays a count of privileges based on the grantee and the owner of the object. This provides insight into accounts or roles that may have powerful privileges.
- The Privileges Distribution By Grantee, Owner, Privilege Report displays a count of privileges based on the privilege, the grantee, and the owner of the object. This provides insight into the accounts or roles that may have powerful privileges.
- The Roles and Accounts That Have A Given Role report displays the database accounts and roles to which a role has been granted. This report is provided for dependency analysis.
- The Resource Profiles Report provides a view of resource profiles, such as CPU_PER_SESSION and IDLE_TIME, that may be allowing unlimited resource consumption.
- The System Resource Limits Report provides insight into the current system resource usage by the database. This helps determine whether any of these resources are approaching their limits under the existing application load.

Database Audit and Privilege Reports

- Database Report On Core Database Audit Trail
- Database Accounts With AUDIT Privileges



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

- The Database Report On Core Database Audit Trail Report returns audit records for the audit policy defined and any auditing records that are generated for audit statements you have defined. This report only displays audit records that are captured if the database initialization parameter AUDIT_TRAIL has been set to DB.
- The Database Accounts With AUDIT Privileges Report displays all database accounts and roles that have the AUDIT ANY or AUDIT SYSTEM privilege. This privilege can be used to disable auditing, which could be used to eliminate the audit trail record of a intruder who has compromised the system.

Object Privilege Reports

- The Object Access By PUBLIC report
- The Object Access Not By PUBLIC report
- The Direct Object Privileges report
- The Object Dependencies report



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

These reports enable you to monitor the source of user privileges and provide information for the design of roles.

- **Object Access By PUBLIC:** This report lists all objects granted to PUBLIC. The report details all the object access privileges that the database account has through grants to PUBLIC. On the Reports Parameters page, you can filter the results based on the privilege, object owner, or object name. This report is useful for tracking the privileges that a particular user has through PUBLIC.
- **Object Access Not By PUBLIC:** This report details all the object privileges that a database account has through direct grants to the account or through a role, but not through PUBLIC. You specify a single database account on the Report Parameters page, and you can filter the results based on the privilege, object owner, or object name.
- **Direct Object Privileges:** This report shows the direct object privileges granted to nonsystem database accounts. It is provided as a tool to help determine where password-protected roles can be implemented.
- **Object Dependencies:** The Object Dependencies Report describes all dependencies in the database between procedures, packages, functions, package bodies, and triggers, including dependencies on views created without any database links. It can help you develop a security policy using the principle of least privilege for existing applications.

Sensitive Objects Reports

- Execute Privileges to Strong SYS Packages
- Access to Sensitive Objects
- Public Execute Privilege to SYS PL/SQL Procedures



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

- **System Privileges by Privilege:** This report displays the database accounts and roles that have the system privilege selected on the Report Parameters page.
- **Execute Privileges to Strong SYS Packages:** This report shows the database accounts and roles that have execute privileges on system packages that can be used to access operating system (OS) resources or other powerful system packages. The following system packages are included:

DBMS_ALERT	UTL_FILE	DBMS_CAPTURE_ADMIN
DBMS_DDL	UTL_HTTP	DBMS_DISTRIBUTED_TRUST_ADMIN
DBMS_FGA	UTL_SMTP	DBMS_RESOURCE_MANAGER
DBMS_JOB	UTL_TCP	DBMS_BACKUP_RESTORE
DBMS_LDAP	DBMS_LOGMNR	DBMS_ORACLE_TRACE_AGENT
DBMS_LOB	DBMS_LOGMNR_D	DBMS_RESOURCE_MANAGER_PRIVS
DBMS_RLS	DBMS_REPAIR	DBMS_OBFUSCATION_TOOLKIT
DBMS_PIPE	DBMS_REPCAT	DBMS_REPCAT_ADMIN
DBMS_RANDOM	DBMS_SESSION	DEBUG_EXTPROC

- **Access to Sensitive Objects:** This report shows the database accounts and roles that have object privileges on system tables or views that contain sensitive information. It includes the following system tables and views:

ALL_SOURCE	LINK\$	DBMS_BACKUP_RESTORE
ALL_USERS	OBJ\$	STATS_SQL_SUMMARY
APPROLE\$	OBJAUTH\$	STATS_SQLTEXT
AUD\$	OBJPRIV\$	STREAMS\$_PRIVILEGED_USER
AUDIT_TRAIL\$	PROFILE\$	SYSTEM_PRIVILEGE_MAP
DBA_ROLE_PRIVS	SOURCE\$	TABLE_PRIVILEGE_MAP
DBA_ROLES	TRIGGER\$	PROXY_ROLE_DATA\$
DBA_TAB_PRIVS	USER\$	PROXY_ROLE_INFO\$
DEFROLE\$	FGA_LOG\$	ROLE_ROLE_PRIVS
USER_HISTORY\$	USER\$	USER_TAB_PRIVS

- **Public Execute Privilege to SYS PL/SQL Procedures:** The Public Execute Privilege to SYS PL/SQL Procedures report shows all the database accounts and roles that have execute privileges on packages owned by **SYS**. This can be used to determine which privileges can be revoked from **PUBLIC** or from other accounts and roles. This reduces vulnerabilities as part of an overall least-privileges security policy implementation.

Unified Audit Trail

- Unified Audit Trail Generic
- Audit Configuration Changes
- Failed Login Attempts
- DDL Actions
- DML Actions
- Oracle Label Security Actions
- User Account Actions
- Datapump Actions
- RMAN Actions
- Privileges Exercised

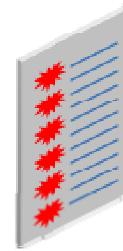


Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

- The Unified Audit Trail Generic Report displays the first 2000 audit records from the unified audit trail. If there are more audit records, it requests user to use “Search” to refine the result list.
- The Audit Configuration Changes Report displays audit records corresponding to the changes made to the audit configuration.
- The Failed Login Attempts Report displays audit records corresponding to the failed login attempts.
- The DDL Audit Actions Report displays audit records corresponding to the DDL actions.
- The DML Audit Actions Report displays audit records corresponding to the DML actions.
- The Oracle Label Security Actions Report displays the audit trail for OLS actions.
- The User Account Actions Report displays audit records corresponding to the user account actions.
- The Datapump Report displays audit records corresponding to the Datapump actions.
- The RMAN Actions Report displays audit records corresponding to the RMAN actions.
- The Privileges Exercised Report displays audit records showing various privileges exercised.

Other Security Vulnerability Reports

- OS Security Vulnerability Privileges
- Java Policy Grants
- Unwrapped Package Bodies
- User Name OR Password Tables
- Tablespace Quotas
- Non-Owner Object Trigger
- Password History Access
- WITH GRANT Privilege Grants
- Objects Dependent on Dynamic SQL



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Each of these reports focuses on a possible security vulnerability:

- The **OS Security Vulnerability Privileges** report shows the database accounts and roles that have the required system privileges to export sensitive or otherwise protected information to the operating system.
- The **Java Policy Grants** report shows the Java policy permissions stored in the database. It helps to reveal violations to the least privilege principle. Look for GRANT, READ, or WRITE privileges to PUBLIC or other accounts and roles that do not necessarily need the privileges. It is advisable to disable Java loading privileges from PUBLIC, if Java is not required in the database.
- **Objects Dependent on Dynamic SQL:** This report shows objects that use dynamic SQL. Such objects can be the target for a SQL injection attack. After determining the objects that use dynamic SQL, you need to check the privileges that client applications (for example, a Web application) have over the object. You also need to check the access granted for the object to PUBLIC or a wider account base. These actions can limit the scope of an attack.
- **Unwrapped PL/SQL Package Bodies:** This report displays PL/SQL package procedures that are not wrapped. Oracle provides a wrap utility that obfuscates code to the point where it cannot be read in the data dictionary. This prevents a hacker from reading source code and determining how to circumvent data protection.

- **User Name or Password Tables report:** This report helps to identify application tables in the database that store usernames and password strings by displaying tables with column_names with the strings USER%NAME or PASSWORD embedded. These tables should be examined to determine whether the information is encrypted, and if not, the code and applications using them should be modified to protect them from being visible to database sessions.
- **Tablespace Quotas report:** This report shows database accounts that have unlimited quotas on one or more tablespaces. These tablespaces can become potential targets for DoS attacks.
- **Non-Owner Object Trigger report:** This report helps to reveal triggers that are owned by a database account that is different from the account that owns the database object on which the trigger acts. If the trigger is not part of a trusted database application, it can steal sensitive data, possibly from the tables protected through Oracle Label Security (OLS) or Virtual Private Database (VPD), and place it into an unprotected table for subsequent viewing or export. This kind of trigger requires that the owner of the trigger have privileges granted directly on the object that the trigger accesses.
- The **Password History Access Report** shows database accounts that have access to the USER_HISTORY\$ table that stores hashed passwords that were previously used by each account.
- The **WITH GRANT Privileges Report** shows all database accounts that have been granted privileges with the WITH GRANT clause.